One-Way Delay Measurement based on Flow Data in Large Enterprise Networks

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde
eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Jochen Andreas Kögel

geb. in Waiblingen

Hauptberichter:	Prof. em. Dr.-Ing. Dr. h. c. mult. Paul J. Kühn
1. Mitberichter:	Prof. Dr.-Ing. Georg Carle, TU München
2. Mitberichter:	Prof. Dr.-Ing. Andreas Kirstädter
Tag der Einreichung:	08. Juni 2012
Tag der mündlichen Prüfung:	19. September 2013

Institut für Kommunikationsnetze und Rechnersysteme
der Universität Stuttgart

2013

To Simone.

Abstract

Today, an efficient and powerful enterprise IT infrastructure provides a vital basis for economic success. As more and more business critical services are network-based, enterprise networks are the foundation for all business-related activities. Such networks need to be well-managed, which means they require thorough monitoring, since any service degradation directly impacts the enterprise's productivity. At best, problems and resource shortages are detected far before they impact user experience. However, network management and monitoring have to deal with limitations, especially employing expensive additional measurement equipment is infeasible due to limited budgets.

There are different approaches for network monitoring. First, there is component monitoring, which monitors parameters, such as CPU or link utilization, or error counts. Another class of monitoring approaches is flow monitoring, which allows for detailed traffic analysis using aggregated measurement data on traffic flows. Both of these approaches are passive, i.e., they do not induce additional traffic into the network. Active measurements come into place when service response times or timing-based network parameters, such as *One-Way Delay* (OWD), *Round Trip Time* (RTT), or jitter are measured. Especially the One-Way Delay is an important indicator for network load and problems, since OWD has direct impact on service response time - both as impact on the RTT itself as well as an influencing part to congestion control. Furthermore, OWD is impacted by queuing delay, which directly relates to network utilization and congestion in router queues.

The flow-based OWD measurement approach has the advantage that it uses flow data as input, which is often already available in enterprise network scenarios, or its creation can simply be enabled by turning on existing router features. Due to these advantages, flow-based OWD measurement does not require additional measurement equipment or heavy changes to device configurations in the network. It requires only additional processing of flow data for extracting the OWD samples. The fundamental question is, however, which accuracy can be gained by flow data based OWD measurements. Furthermore, efficient processing mechanisms are necessary in order to deal with the huge amount of data that has to be processed in large networks. These questions are addressed in this thesis.

Chapter 2 introduces the fundamentals of IP-based networks and delay measurement. A section on router functionality and architecture shows that depending on the router architecture a different number of forwarding engines and queues are present. This has impact on the positions where flow capturing can be performed and which queues can contribute queuing delay before or after a measuring point. Furthermore, network structures are discussed and the differences

between the global Internet and global enterprise networks are highlighted. Enterprise networks typically use MPLS VPNs with several edge routers per branch location for resilience reasons. These edge routers are typical devices where flow capturing is be performed. A section on metrology introduces metrological terms and details the concept of measurement errors and error propagation. Before existing approaches for delay measurement are discussed, the measurands and metrics are clarified. It is shown that there are different definitions for network delay, and a definition for this thesis is specified. The overview of existing delay measurement approaches distinguishes active and passive measurement approaches. It is highlighted that predominantly active measurement technologies are used in today's networks, but passive approaches have certain advantages and are of high interest in research activities. Finally, the problem of clock errors is addressed by first clarifying related terms and then introducing clock synchronization mechanisms and algorithms for removing clock skew or drift from measurement data sets.

Chapter 3 deals with flow capturing, an important mechanism for traffic monitoring in enterprise and provider networks. This chapter starts with giving consistent definitions of the general term flow, more specific flow definitions, and flow capturing related terms. Then, it introduces and compares several flow capturing mechanisms. Here, the main focus is on NetFlow, which is the most widely used de-facto standard for flow capturing today. For delay measurement, the very point at which flow data is captured in routers is important. Thus, flow capturing implementations in routers and probes are discussed in detail. Related work regarding flow capturing is addressed in terms of flow capturing application and deployment, as well as in terms of flow data processing approaches. It is shown that flow capturing is used predominantly for traffic reporting and accounting while little research towards obtaining QoS-related information has been performed. Flow data processing is predominantly performed in an offline processing approach, i.e., data is stored to disk or to a database before being analyzed.

Based on the fundamentals provided by Chapter 2 and Chapter 3, Chapter 4 motivates the flow-based OWD measurement approach in Section 4.1. This approach is compared to other active and passive delay measurement approaches and it is found that this approach is advantageous due to the three criteria one-way measurement, passive measurement and low effort measurement. The following three sections systematically address problems in flow-based OWD measurement that arise from different effects and errors. Section 4.2 starts with considering network effects, Section 4.3 deals with flow capturing effects and errors, and Section 4.4 details timestamp effects and errors in flow capturing devices by introducing a timestamp creation model. Systematically, each effect is evaluated in terms of its impact on OWD extraction, and for each measurement error methods for correction or quantification is been given. Throughout these sections, an OWD extraction model is developed that takes flow records as input and creates OWD samples as output while handling all effects and errors appropriately. It is shown that error correction and quantification requires parameters on flow capturing devices that can be provided by means of an exporter profile.

Chapter 5 details an online processing approach for the efficient implementation of the OWD extraction model. First, the export effects in flow data are analyzed and, by taking processing requirements into account, a multi-stage window-based extraction mechanism is proposed. This mechanism uses a result and a record window that define how long flow records and OWD results are buffered in memory. Second, the window-based processing approach is detailed by evaluating the typical flow record export characteristics and deriving from the latter methods

for window handling and dimensioning. An overview of a design for the window-based processing approach is given. This design was implemented and has proven its technical feasibility. The third section presents a profile-based method for window dimensioning that takes record rates and export jitter distributions into account for giving an estimation on the number of OWD samples and memory requirements with different window sizes. This method especially considers the effects in global enterprise networks regarding different record rates at different times of a day for different locations. The parameters required for dimensioning depend on traffic characteristics and flow capturing device configuration and could be given in an exporter profile.

The exporter profile approach that has been motivated by the need for measurement error related and export effect related parameters in Chapter 4 and Chapter 5 is detailed in Chapter 6. This chapter systematically elaborates the various parameter dependencies and specifies the exporter profile consisting of three parts that contain parameters on the flow capturing device, a certain observation point, or an observation point pair, i.e, a path between observation points. Section 6.3 discusses various profile creation approaches based on parameter dependencies. It is concluded that traffic-dependent parameters can only be obtained by a profile creation approach based on flow data and that this approach can also be used for obtaining the other parameters. Consequently, a flow data based profile creation approach is proposed in Section 6.4. This approach is a seven step iterative offline processing approach that extracts profile parameters from a reference flow data set. Due to the special properties of flow data, three profiling steps demand for the design of special algorithms. These algorithms for record rate estimation, obtaining timestamp resolution, and *Observation Point Pair* (OPP) inference are presented in detail.

Chapter 7 presents how flow-based OWD measurement with profile support was applied using flow data from an enterprise network. Furthermore, this chapter evaluates measurement accuracy. The evaluation is based on NetFlow v5 data and active RTT reference measurements that were collected over five days from three different locations. OWD measurements have been evaluated on a path between two European locations as well as between a European and an Australian location of the global enterprise network. For each exporter of the scenario, profile parameters were obtained. A comparison with active measurement shows that measurement accuracy can be considerably improved using the profile-based correction methods. Furthermore, the flow-based OWD measurement results closely match the active measurement results and random errors can be correctly quantified based on profile parameters. The effect of exporter-internal clock skew is evaluated separately and it is been shown that this effect can lead to a correctable error of up to 10 ms. In terms of profile-based window dimensioning for the online processing approach, it is shown that the amount of samples as well as memory requirements can be well estimated.

In summary, this thesis shows the feasibility of flow-based OWD measurement with profile support. Flow-based OWD measurement with profile support is feasible as addition or replacement for active OWD measurements that are performed today. Furthermore, the approaches for improving timestamp accuracy can be taken as input for any other flow analysis application that relies on flow capturing timestamps.

Kurzfassung

Messung der Einweglaufzeit mittels Flow-Daten in großen Unternehmensnetzen

In Unternehmen ist eine effiziente und leistungsfähige IT-Infrastruktur die Grundlage für wirtschaftlichen Erfolg. Vor allem ein gut funktionierendes Unternehmensnetz ist für geschäftliche Aktivitäten essenziell, da geschäftskritische Anwendungen zunehmend netzbasiert sind. Der Betrieb solcher Netze erfordert sorgfältiges Monitoring, da jede Einschränkung der Anwendungen direkt die Produktivität des Unternehmens beeinflusst. Hierbei sollen einerseits Probleme und Ressourcenengpässe möglichst früh erkannt werden, d. h. bevor die Anwendungsnutzung davon betroffen ist. Andererseits haben Netzbetrieb und das Netzmonitoring mit finanziellen Einschränkungen umzugehen, sodass der Einsatz teurer zusätzlicher Messgeräte aus Kostengründen nicht umsetzbar ist. Dies verlangt nach universellen kostengünstigen Messverfahren.

Beim Netzmonitoring werden verschiedene Ansätze verfolgt. Zum einen gibt es Komponenten-Monitoring, das z.B. die Auslastung von Prozessoren oder Netzverbindungen misst, sowie Fehler erfasst. Zum anderen stellt Flow-Monitoring aggregierte Messdaten zur Verfügung, die detaillierte Verkehrsanalysen ermöglichen. Beide der genannten Ansätze gehören zur Klasse der passiven Messverfahren, d.h. es wird kein zusätzlicher Messverkehr im Netz erzeugt. Letzteres ist bei aktiven Messverfahren der Fall, die oft eingesetzt werden um zeitbasierte Netzcharakteristika zu messen, wie z.B. die Einweglaufzeit OWD (*One-Way Delay*), die Umlaufzeit RTT (*Round Trip Time*) oder Laufzeitschwankungen (Jitter). Die OWD ist ein wichtiger Indikator für die Netzauslastung und für Netzprobleme. Außerdem wirkt sich die OWD direkt auf Antwortzeiten von Anwendungen aus. Dies geschieht einerseits durch die Verzögerung von Anfragen und andererseits durch den Einfluss der OWD auf die Überlastregelung (Congestion-Control).

Die Messung der OWD mittels Flow-Daten hat den Vorteil, dass Flow-Daten in Unternehmensnetzen oft schon für andere Zwecke erfasst werden, bzw. deren Erfassung einfach in Routern aktiviert werden kann. Aufgrund dieser Eigenschaften benötigt Flow-basierte OWD-Messung keine zusätzlichen Messgeräte oder grundlegende Änderungen an der Konfiguration von Netzelementen. Es wird nur eine weitergehende Verarbeitung von Flow-Daten zur Extraktion von OWD-Messwerten benötigt. Die grundlegende Frage ist, welche Genauigkeit mit der Flow-basierten OWD-Messung erzielt werden kann. Des Weiteren sind effiziente Verarbeitungsmechanismen notwendig, um die immensen Datenmengen aus großen Netzen verarbeiten zu können. Diese Punkte werden in der vorliegenden Dissertation behandelt.

Im Folgenden führt Kapitel 1 in das Thema ein. Kapitel 2 stellt dann die Grundlagen von IP-basierten Netzen und von Laufzeitmessungen vor. Ein Abschnitt zur Routerfunktionalität und -architektur zeigt, dass sich Router hinsichtlich der Anzahl von Forwarding-Engines und Warteschlangen unterscheiden. Dies wirkt sich darauf aus, an welchen Punkten Flow-Daten erzeugt werden können und welche Warteschlangen mit Verzögerungen vor oder nach dem Messpunkt zur Gesamtlaufzeit beitragen.f Des Weiteren werden Netzstrukturen behandelt, vor allem die Unterschiede zwischen der Internet-Topologie und Unternehmensnetzen. Unternehmensnetze basieren typischerweise auf MPLS VPNs (Multi Protocol Label Switching Virtual Private Networks) mit mehreren Edge-Routern pro Niederlassung aus Gründen der Ausfallsicherheit. Diese Edge-Router sind typischerweise die Geräte, in denen die Flow-Erfassung (Flow-Capturing) erfolgt. Ein Abschnitt über Metrologie führt metrologische Grundbegriffe ein und behandelt die Konzepte *Messfehler* und *Fehlerfortpflanzung*. Bevor existierende Ansätze für Laufzeitmessungen diskutiert werden, werden Messgrößen und Metriken erläutert. Es wird gezeigt, dass unterschiedliche Definitionen für die Laufzeit in Netzen existieren und es wird eine Definition für diese Arbeit herausgearbeitet. Ein Überblick über existierende Messverfahren unterscheidet zwischen aktiven und passiven Messverfahren. Es wird erläutert, dass heute überwiegend aktive Messtechniken eingesetzt werden, passive Messtechniken aber bestimmte Vorteile haben, weshalb diesen im Forschungsumfeld ein großes Interesse gilt. Zum Schluss wird das Problem der Uhrenfehler behandelt, indem zuerst die verwandten Begriffe und dann die Uhrensynchronisationsmechanismen, sowie Algorithmen zum Entfernen von Uhrenversatz und -drift aus Messdatensätzen betrachtet werden.

Kapitel 3 behandelt die Flow-Erfassung (Flow-Capturing), ein wichtiger Mechanismus für das Monitoring von Verkehrsströmen in Unternehmensnetzen und Netzen öffentlicher Netzbetreiber (Provider). Dieses Kapitel beginnt mit der Darstellung einer konsistenten Definition für den allgemeinen Flow-Begriff und erarbeitet darauf basierend spezifischere Flow-Begriffe und andere Begriffe im Zusammenhang mit Flow-Capturing. Dann werden verschiedene Flow-Capturing-Mechanismen eingeführt und verglichen. Hierbei steht vor allem NetFlow im Fokus, welches der heute am weitesten verbreitete de-facto Standard für Flow-Capturing ist. Für die Messung von Laufzeiten ist der genaue Punkt, an welchem Flow-Daten im Router erfasst werden, wichtig. Deshalb werden Flow-Capturing-Implementierungen in Routern und Mess-Sonden detailliert betrachtet. Verwandte Arbeiten zu Flow-Capturing werden hinsichtlich der Anwendung von Flow-Capturing und der Positionierung von Flow-Capturing-Geräten im Netz betrachtet, sowie hinsichtlich der Datenverarbeitungsansätze. Es wird gezeigt, dass Flow-Capturing vor allem für das Erstellen von Berichten über das Verkehrsverhalten und für Abrechnungszwecke eingesetzt wird, während wenig Forschungsarbeiten zur Extraktion zeitbasierter Netzcharakteristika bekannt sind. Die Verarbeitung von Flow-Daten wird heute hauptsächlich offline vorgenommen, d.h. Daten werden auf Festplatten oder in Datenbanken gespeichert, bevor diese weitergehend analysiert werden.

Basierend auf den Grundlagen von Kapitel 2 und Kapitel 3 stellt Kapitel 4 im ersten Abschnitt den Ansatz der Flow-basierten OWD-Messung vor. Dieser Ansatz wird mit anderen aktiven und passiven Messverfahren verglichen und es wird begründet, dass dieser Ansatz aufgrund der Kriterien *Einwegmessung*, *passive Messung* und *Messung mit geringem Aufwand* vorteilhaft ist. Die folgenden drei Abschnitte arbeiten systematisch Probleme bei Flow-basierter OWD-Messung auf, die aufgrund verschiedener Effekte und Fehler entstehen. Abschnitt 4.2 beginnt mit der Betrachtung von Netzeffekten, Abschnitt 4.3 behandelt Effekte und Fehler

des Flow-Capturings, und Abschnitt 4.4 vertieft Zeitstempeleffekte und -fehler durch die Einführung eines Modells zur Zeitstempelerzeugung. Systematisch wird jeder Effekt bezüglich des Einflusses auf die OWD-Extraktion bewertet und für die Messfehler werden Verfahren zur Korrektur oder Quantifizierung angegeben. Über diese Abschnitte hinweg wird ein OWD-Extraktionsmodell entwickelt, das Flow-Records als Eingangsdaten verwendet und möglichst korrekte OWD-Messwerte unter Berücksichtigung aller Effekte und Fehler erzeugt. Es wird gezeigt, dass für die Fehlerkorrektur und -quantifizierung Parameter über die Flow-Capturing-Geräte (Exporter) benötigt werden, und dass diese in einem Exporter-Profil zur Verfügung gestellt werden können.

Für das zuvor entwickelte OWD-Extraktionsmodell wird in Kapitel 5 ein Online-Verarbeitungsansatz entwickelt, der eine effizienten Implementierung ermöglicht. Zuerst werden Export-Effekte in Flow-Daten analysiert und durch die Berücksichtigung von Verarbeitungsanforderungen wird ein mehrstufiger fensterbasierter Extraktionsmechanismus vorgeschlagen. Dieser Mechanismus verwendet ein Ergebnisfenster und ein Record-Fenster, die jeweils festlegen, wie lange Flow-Records und OWD-Messwerte im Speicher gepuffert werden. Im zweiten Abschnitt wird der fensterbasierte Verarbeitungsansatz vertieft, indem die typischen Exportcharakteristika von Flow-Records ausgewertet werden und daraus Methoden für die Handhabung der Fenster und für die Dimensionierung abgeleitet werden. Um die technische Umsetzbarkeit zu zeigen, wurde dieser Verarbeitungsansatz prototypisch implementiert. Ein Überblick über diese Implementierung wird gegeben. Der dritte Abschnitt stellt ein profilbasiertes Verfahren zur Fensterdimensionierung vor, das Record-Raten und Export-Jitter-Verteilungen berücksichtigt, um die Anzahl der OWD-Messwerte und den Speicherbedarf für unterschiedliche Fenstergrößen abzuschätzen. Dieses Verfahren berücksichtigt insbesondere Effekte in globalen Unternehmensnetzen hinsichtlich unterschiedlicher Record-Raten zu unterschiedlichen Tageszeiten an unterschiedlichen Standorten. Die für die Dimensionierung benötigten Parameter hängen von Verkehrscharakteristika und der Konfiguration von Flow-Capturing-Geräten ab und können über ein Exporter-Profil zur Verfügung gestellt werden.

Der Ansatz des Exporter-Profils wurde durch den Bedarf für Messfehler- und Exporter-Parameter, welche spezifische Effekte beschreiben, in Kapitel 4 und Kapitel 5 begründet und wird nun in Kapitel 6 detailliert dargestellt. Dieses Kapitel arbeitet systematisch die unterschiedlichen Parameterabhängigkeiten auf und spezifiziert das Exporter-Profil, welches aus drei Teilen besteht: Ein Teil mit Parametern des Flow-Capturing-Gerätes, ein weiterer Teil mit Parametern für einen bestimmten Beobachtungspunkt und ein Teil mit Parametern für ein Paar von Beobachtungspunkten, d.h., ein Pfad zwischen Beobachtungspunkten. Abschnitt 6.3 diskutiert unterschiedliche Ansätze zur Profilerzeugung basierend auf Profilabhängigkeiten. Es wird argumentiert, dass verkehrsabhängige Parameter nur durch einen Profilerzeugungsansatz gewonnen werden können, der auf Flow-Daten basiert und dass ein solcher Ansatz auch zur Gewinnung der anderen Parameter verwendet werden kann. Daraus folgt ein Flow-basierter Ansatz zur Profilerzeugung, der dann in Abschnitt 6.4 erarbeitet wird. Dieser Ansatz ist ein Offline-Verarbeitungsansatz aus sieben iterativen Schritten, der Exporter-Profil-Parameter aus einem Referenz-Flow-Datensatz gewinnt. Aufgrund der speziellen Eigenschaften von Flow-Daten wurden für drei der sieben Profilbildungsschritte spezielle Algorithmen entwickelt. Diese Algorithmen zur Flow-Raten-Abschätzung, zur Erfassung der Zeitstempelauflösung und für die Inferenz von Beobachtungspunkt-Paaren werden im Detail behandelt.

Wie die Flow-basierte OWD-Messung mit Profilunterstützung für Flow-Daten eines globalen Unternehmensnetzes eingesetzt wurde, zeigt Kapitel 7. Des Weiteren bewertet dieses Kapitel die Messgenauigkeit. Die Bewertung basiert auf Flow-Daten (NetFlow v5) und den Ergebnissen von aktiven RTT-Messungen als Referenz, die jeweils über fünf Tage für drei unterschiedliche Standorte aufgezeichnet wurden. OWD-Messungen wurden auf einem Pfad zwischen zwei europäischen Standorten sowie zwischen einem europäischen Standort und einem australischen Standort bewertet. Der Vergleich mit aktiven Messungen zeigt, dass die Messgenauigkeit durch die profilbasierten Korrekturverfahren beträchtlich verbessert werden kann. Außerdem ist der Unterschied zwischen den Ergebnissen der Flow-basierten Messung und den Ergebnissen der aktiven Messung sehr klein. Zufällige Messfehler können aufgrund von Profilparametern korrekt quantifiziert werden. Der Effekt des Exporter-internen Uhrenversatzes wird separat bewertet und es wird gezeigt, dass dieser Effekt zu einem korrigierbaren Messfehler von bis zu 10 ms führen kann. Hinsichtlich der profilbasierten Fensterdimensionierung für den Online-Verarbeitungsansatz wird gezeigt, dass die Anzahl extrahierbarer Messwerte sowie der Speicherbedarf gut abgeschätzt werden können.

Zusammenfassend zeigt diese Arbeit die Machbarkeit Flow-basierter OWD-Messungen mit Profilunterstützung. Dieses Messverfahren benötigt keine zusätzlichen Komponenten im Netz und nur geringe Zusatzinformationen zur Konfiguration. Damit stellt es eine einfach zu installierende, universelle und kostengünstige Möglichkeit zur Messung der Einweglaufzeit dar, die zusätzlich zu oder als Ersatz für aktive Messungen einsetzbar ist. Die in dieser Arbeit entwickelten Ansätze zur Verbesserung der Zeitstempelgenauigkeit können zudem auch für andere zeitbasierte Flow-Analysen verwendet werden.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

ACL	Access Control List	8
AS	Autonomous System	10
ASIC	Application Specific Integrated Circuit	9
BGP	Border Gateway Protocol	10
BIPM	Bureau international des poids et mesures	12
BR	Border Router	10
CC	Central Card	8, 46
CDF	Cumulative Distribution Function	107
CE	Customer Edge	11
CEP	Complex Event Processing	54
CPE	Customer Premises Equipment	8
CQL	Continuous Query Language	54
CR	Core Router	10
DBMS	Data Base Management System	53
DC	Data Center	10
DFT	Discrete Fourier Transform	130
DiffServ	Differentiated Services	5
DNS	Domain Name System	7
DoS	Denial of Service	46
DPI	Deep Packet Inspection	25

DSCP	Differentiated Services Code Point	6
DSMS	Data Stream Management Systems	54
E2E	End-to-End	58
EC	Exporter Component	122, 141
ECMP	Equal-Cost Multi-Path	12
ECN	Explicit Congestion Notification	6
ESA	Export-Systime-Alignment	85, 89
FACT	Flow-based Approach for Connectivity Tracking	49
FE	Forwarding Engine	7, 46
FEC	Forwarding Equivalence Class	8
FFT	Fast Fourier Transform	132
FPGA	Field Programmable Gate Array	123
GPS	Global Positioning System	21
GRE	Generic Routing Encapsulation	165
GUM	Guide to the Expression of Uncertainty in Measurement	12
HTTP	Hypertext Transfer Protocol	7
IANA	Internet Assigned Numbers Authority	42
ICMP	Internet Control Message Protocol	7
ICMPv6	Internet Control Message Protocol, version 6	7
IDS	Intrusion Detection System	48
IEC	International Electrotechnical Commission	12
IETF	Internet Engineering Task Force	5
IHL	IP Header Length	6
IntServ	Integrated Services	28
IP	Internet Protocol	5
IPFIX	IP Flow Information Export	25, 33
IPPM	IP Performance Metrics	17
IPSLA	Cisco IOS IP Service Level Agreements	19

IPTD	IP Packet Transfer Delay	19
IPTL Flow	IP Transport Layer Flow	26
IPv4	Internet Protocol version 4	6
IPv6	Internet Protocol version 6	6
IRTF	Internet Research Task Force	2
ISO	International Organization for Standardization	12
ISP	Internet Service Provider	10
ISQ	International System of Quantities	12
JCGM	Joined Committee for Guides in Metrology	12
LAN	Local Area Network	11
LC	Line Card	8, 46
MIB	Management Information Base	34
MPLS	Multi Protocol Label Switching	11
MTU	Maximum Transmission Unit	5
N2N	Node-to-Node	58
NAT	Network Address Translation	8
NMRG	Network Management Research Group	2
NTP	Network Time Protocol	22
OP	Observation Point	27, 29
OPC	Observation Point Component	122, 141
OPP	Observation Point Pair	iii, 61
OPPC	Observation Point Path Component	122
OW	One-Way	58
OWAMP	One-Way Active Measurement Protocol	19
OWD	One-Way Delay	i, 1, 17
PDF	Probability Density Function	13
PE	Provider Edge	11
PPS	Pulse Per Second	21

PSAMP	Packet Sampling	42
PTP	Precision Time Protocol	22
QoS	Quality of Service	5
RFC	Request for Comments	5
RMON	Remote Network Monitoring	32
RSVP	Resource Reservation Protocol	28
RTC	Real-Time Clock	21
RTFM	Real Time Flow Measurement	28, 32
RTT	Round Trip Time	i, 1, 18
RX	Receive	7
SC	Service Card	46
SCTP	Stream Control Transmission Protocol	6
SEC	Start-End-Correspondence	64
SLA	Service Level Agreement	11
SNMP	Simple Network Management Protocol	32
SRL	Simple Ruleset Language	34
TCP	Transmission Control Protocol	6
TLV	Type Length Value	40
ToS	Type of Service	6
TTL	Time to Live	6
TW	Two-Way	58
TWAMP	Two-Way Active Measurement Protocol	20
TX	Transmit	8
UDP	User Datagram Protocol	6
UTC	Coordinated Universal Time	21
VIM	Vocabulaire International de Métrologie	12
VPN	Virtual Private Network	11
WAN	Wide Area Network	11
WG	Working Group	32
XRMON	Extended RMON	32

Symbols

b	Byte count of a flow record	31
$b_{\mathrm{p,max}}$	Maximum packet size (MTU) in bytes	74
$b_{\mathrm{p,min}}$	Minimum packet size in bytes	73, 74
c_{x}	Correction for the measured value x	14
$c(t)$	Clock value at time t, general clock	21
$\mathrm{C_x}$	Exporter-internal timestamp conversion	80
d	Flow record duration	31
d^*	Flow duration	27, 31
D	Exporter-internal timestamp processing delay	80
$\mathrm{D_1}$	Exporter-internal start timestamp processing delay	93
$\mathrm{D_2}$	Exporter-internal system uptime timestamp processing delay	93
$\mathrm{D_3}$	Exporter-internal UNIX timestamp processing delay	93
$e_{\mathrm{r,b}}$	Byte count error per record	74
e_{e}	Record end time error (single sample)	82
e_{first}	Error of r_{first} (single sample)	88
e_{last}	Error of r_{last} (single sample)	88
e_{su}	Error of r_{su} (single sample)	89
e_{b}	Error of t_{b} (single sample)	89
$e_{\mathrm{r,x}}$	Random (measurement) error of value x	13
e_{s}	Record start time error (single sample)	82
$e_{\mathrm{s,x}}$	Systematic (measurement) error of value x	13
e_{x}	Measurement error for a single sample of value x	13
$\mathbf{F}$	Flow	25
$\mathbf{F}_s$	Subflow	27
f_{K}	Flow key function	26
f_{M}	Measurement function (general)	15
$f_R(J)$	Export jitter PDF per record	114

$f_S(J)$	Export jitter PDF per OWD sample	115
$h_R(J_i)$	Export jitter histogram per record	115
$h_S(J_i)$	Export jitter histogram per OWD sample	115
j	Export jitter of an observation point (variation of export delay)	106
J	Export jitter of an observation point pair (difference between export times)	109
k	Byte count sensitivity threshold	73
$\mathbf{K}$	Flow key	25
L	Record window size	104
l_{L}	Propagation delay	17
l_{P}	Processing delay	17
l_{Q}	Queueing delay	17
l_{T}	Transmission delay	17
n_{full}	Number of full packets in byte count error estimation	74
p	Packet count of a flow record	31
$\mathbf{R}$	Flow record	31, 62
r_{first}	Raw timestamp `first`	40, 79
r_{last}	Raw timestamp `last`	40, 79
r_{nsec}	Raw UNIX nano seconds	40
r_{sec}	Raw UNIX seconds	40
r_{su}	Raw system uptime	40, 79
$\lvert\mathscr{R}_L\rvert$	Amount of records in the record window	116
$\mathscr{R}_{\mathrm{OP}}$	Records exported from a certain OP	126
s	Empirical standard deviation	16
S	Sample count	115
t	(true) time	21
t_{b}	Exporter boot time	40, 79
t_{e}	Record end timestamp	25, 31
t_{e}^*	Flow end timestamp	27, 31

T_j	Time of day of interval with index j	117
t_r	Record cache removal time	36
t_s	Record start timestamp	25, 31
t_s^*	Flow start timestamp	27, 31
t_{uwe}	Time of the upper window edge	108
t_x	Record export time	31, 40, 79
$\mathbf{U}$	Packet properties	25
$u(t)$	UNIX time reported by an exporter at time t	79
$\mathbf{V}$	Flow record attributes	30
x_t	True (quantity) value of x	13
y(t)	System time reported by an exporter at time t	79

α	Significance level	16
β_x	Known systematic error (bias) of value x	14
$\beta_{\Delta t_e}$	Known systematic error (bias) of r_{last}	89
$\beta_{\Delta t_s}$	Known systematic error (bias) of Δt_e	89
β_e	Known systematic error (bias) of record end time	89
β_{first}	Known systematic error (bias) of r_{first}	89
β_{last}	Known systematic error (bias) of Δt_s	89
β_{nsec}	Known systematic error (bias) of r_{nsec}	89
β_s	Known systematic error (bias) of record start time	89
β_{su}	Known systematic error (bias) of r_{su}	89
δ_e	Export delay, measured from record end	32
δ_s	Export delay, measured from record start	31
Δb	Byte count difference	62
Δp	Packet count difference	62
Δt_e	End time difference	62
Δt_s	Start time difference	62

γ	Confidence level	16
κ	Resolution alignment offset (wrt. aligned resolution)	85
λ_R	Record rate	116
$\lambda_{R,\max}(L, T_\mathrm{j})$	Maximum record rate per window size and time of day interval	117
$\omega_c(t)$	Clock offset of clock c at time t	21
ω_0	Clock offset at time t=0	21
ω_u	Offset of UNIX timestamp to true time	80
$\omega_{0,u}$	Clock offset of UNIX time at time t=0	81
ω_y	System time offset. Difference between internal clocks y and u.	79
$\omega_{0,y}$	System clock offset constant	81
$\varphi(t)$	Clock skew	21
$\varphi_c(t)$	Clock skew of clock c at time t	21
φ_0	Clock skew time at time t=0	21
$\varphi_{0,u}$	Constant skew of UNIX time	81
$\varphi_{0,y}$	System clock skew constant	81
ρ	Resolution of a timestamp	80
ρ_a	Aligned resolution	84
ρ_d	Resolution of the record duration d	87
ρ_e	Resolution of the record end time t_e	84
ρ_first	Resolution of the raw timestamp r_first	84
ρ_last	Resolution of the raw timestamp r_last	84
ρ_nsec	Resolution of the raw timestamp r_nsec	84
ρ_s	Resolution of the record start time t_s	84
ρ_sec	Resolution of the raw timestamp r_sec	84
ρ_su	Resolution of the raw timestamp r_su	84
ρ_u	Unaligned resolution	84
ρ_x	Resolution of the export time t_x	84
σ	Standard deviation	16

τ	Range of a distribution	15
ϑ_{act}	Active timeout	36
ϑ_{inact}	Inactive timeout	36

Nomenclature

x denotes a scalar, $\mathbf{x}$ a vector, and $\mathcal{X}$ a set.

1 Introduction

Today, an efficient and powerful enterprise IT infrastructure provides a vital basis for economic success. As more and more business critical services are network-based, enterprise networks are the basis for all business-related activities. Such networks need to be well-managed, which means they require thorough monitoring, since any service degradation directly impacts the enterprise's productivity. At best, problems and resource shortage is detected far before it impacts user experience. However, network management and monitoring has to deal with limitations, especially employing expensive additional measurement equipment is infeasible due to limited budgets.

There are different approaches for network monitoring. First, there is component monitoring, which monitors parameters, such as CPU or link utilization, or error counts. Another class of monitoring approaches is flow monitoring, which allows for detailed traffic analysis at yet coarse grained measurement data. Both of these approaches are passive, i.e., they do not induce additional traffic into the network. Active measurements come into place when service response times or timing-based network parameters, such as *One-Way Delay* (OWD), *Round Trip Time* (RTT), or jitter are measured. Especially the One-Way delay is an important indicator for network load and problems, since OWD has direct impact on service response time - both as impact on the RTT itself as well as an influencing part to congestion control. Furthermore, OWD is impacted by queuing delay, which directly relates to network utilization and congestion in router queues.

This thesis proposes a method for OWD measurement, which is based on flow data. The next section gives an overview of this measurement approach and the contributions of this thesis. The second section presents the thesis outline.

1.1 Overview: Flow-based Delay Measurement with Exporter Profiling

While today OWD measurement relies mostly on active measurements, passive measurement does not inject additional traffic and provides measurements of the traffic that directly relates to the network services. The approach presented in this thesis solely relies on flow data that is created by flow capturing mechanisms. This OWD measurement approach has the following advantages:

- Flow capturing is implemented in many routers. Therefore, the creation of flow data can be enabled by turning on this router feature.

- Flow data is often already collected for accounting and reporting purposes. Hence this delay measurement method may not require any changes to router configuration at all.

- Compared to active measurements, there is no need to select paths on which measurements are performed and to setup measurement probes on these paths. OWD from flow data can be calculated between any two points from which flow data is available if there is traffic between these points.

Due to these advantages, flow-based OWD measurement does not require additional measurement equipment or heavy changes to device configurations in the network. It requires only additional processing of flow data for extracting the OWD samples. The fundamental question, however, is which accuracy can be gained by flow data based OWD measurements. Furthermore, efficient processing mechanisms are necessary in order to deal with the huge amount of data that has to be processed in large networks.

This thesis systematically analyzes the problems of the flow-based OWD measurement approach. It describes the problems that result from network effects, flow capturing effects, as well as from errors in the flow capturing process. Furthermore, a timestamp creation model is developed with parameters that describe the resulting timestamp accuracy effects and errors of a flow capturing device. An efficient processing approach that takes into account network and flow capturing effects has been developed as well.

In order to provide reliable measurements, errors have to be detected, corrected and/or quantified. Additionally, properties of the data arrival process have to be known in order to perform efficient flow data processing. These properties depend on the flow capturing devices (flow exporters). This work presents an approach for creating an exporter profile that describes these errors and shows how this profile improves online extraction of OWD from flow data in terms of accuracy and efficiency. The exporter profile approach is depicted in Figure 1.1. As shown in the figure, this principle consists of two phases: in phase 1 exporter profiles are created, and in phase 2 the profile values are used for OWD calculation. Profile creation works in an offline fashion, i.e., it uses a reference data set of a typical working day that is processed iteratively. This phase is not time critical, hence exporter profile creation can use large amounts of data and also perform complex processing tasks. These profiles are then input in phase two where flow data is directly received from the flow capturing devices and OWD values are computed immediately in an online processing approach. Profile values are used for dimensioning the online processing chain as well as for online error detection, compensation, and quantification. Since the profile values are already available in phase 2, online processing can be very efficient.

The ideas and contributions of this thesis have been published in several papers and presentations. The initial ideas on using NetFlow timestamps for performance measurements have been published in [1]. Further results and an overview of the OWD measurement approach have been published at *Internet Research Task Force* (IRTF)*Network Management Research Group* (NMRG) meetings [2, 3, 4] that were organized as expert workshops for flow data analysis. For processing the flow data at high throughput, a flexible java framework has been developed. Its design and a performance evaluation have been published in [5]. The exporter profiling concept for improving OWD measurement has been published as a conference paper in [6]. In addition to these publications closely related to the thesis, the author was involved in

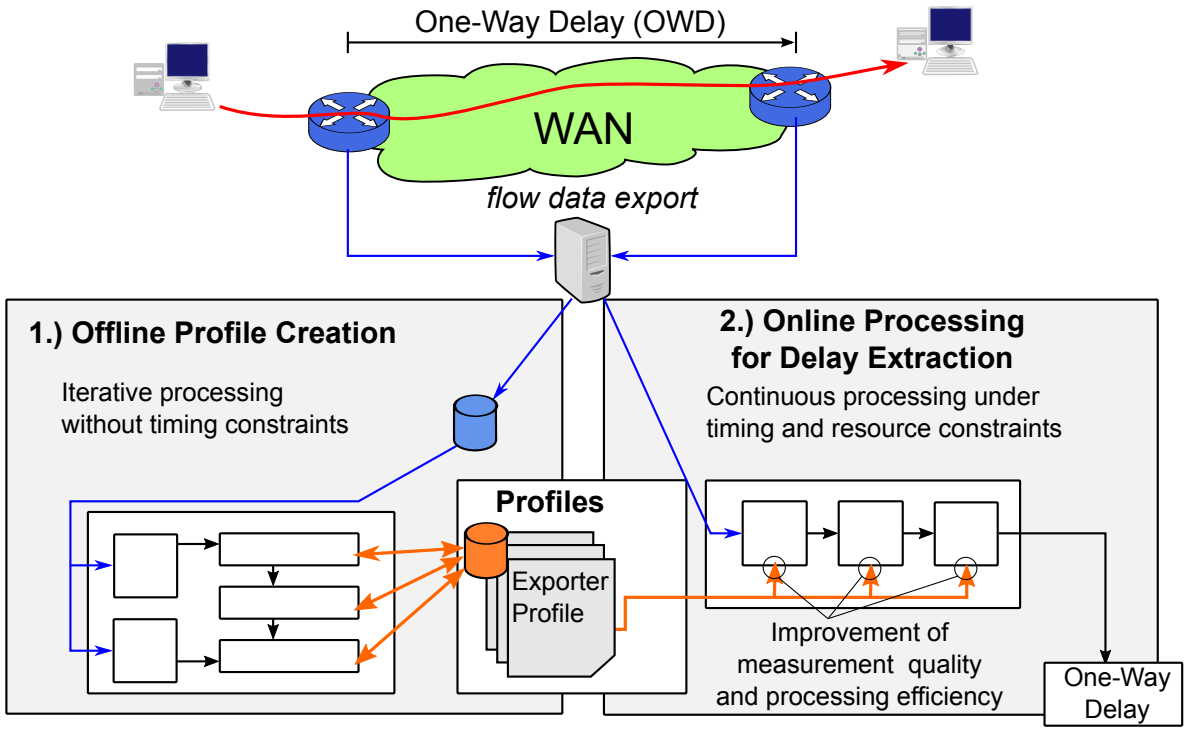


Figure 1.1: Overview of the flow-based OWD measurement method with profile support. It consists of the block for exporter profile creation (left) and delay extraction (right), which makes use of the previously obtained exporter profiles.

joined research on flow-based tracking of connectivity problems [7, 8]. Several student projects in the field of flow analysis and processing were guided by the author [7, 9, 10, 11, 12].

1.2 Thesis Outline

In this thesis, a method for flow-based One-Way Delay measurement method is proposed and evaluated. Chapter 2 and Chapter 3 provide fundamentals on network measurement and on flow capturing. Chapter 4 presents the OWD extraction mechanism and Chapter 5 an online processing implementation of it. Chapter 6 details the exporter profile creation and Chapter 7 evaluates the methods based on measurements in an enterprise network.

Chapter 2 first introduces fundamentals of IP-based networks. Besides the Internet protocol stack it focuses on router functionality and architectures as well as on network structures and routing, which both are fundamentals required for the understanding of flow capturing and network measurements. This chapter also introduces fundamental metrological terms, especially measurement errors. Finally, state of the art in delay measurement is addressed by first introducing measurands and metrics, second by providing an overview on active and passive measurement approaches and third, by highlighting the intrinsic problem of clock errors.

Flow capturing that delivers the input data to the proposed method is introduced in Chapter 3. This chapter first defines the term flow and related terms as well as flow capturing terminol-

ogy. Furthermore, it gives an overview on flow capturing mechanisms and protocols. Different possibilities of flow capturing implementations are also highlighted, since this impacts the properties of the flow data obtained. Additionally, an overview of applications that use flow data and typical deployments of flow capturing in networks is given. Finally, approaches for flow data processing are discussed.

Chapter 4 starts with giving a rationale for flow-based delay measurement based on the fundamentals of the previous chapters. It then derives step by step an OWD extraction model by analyzing characteristics and errors of flow capturing. Where applicable, measures for dealing with effects and errors are developed. As a first step, this chapter considers network effects, i.e., it considers perfect flow data. Second, flow capturing effects and errors are discussed. Third, timestamp errors are considered and a timestamp creation model is presented.

Chapter 5 takes the OWD extraction model of the previous chapter as input and develops an online processing approach for OWD extraction. First, the problem statement and requirements are presented that lead to a multi-stage extraction mechanism, which buffers data in memory based on a window-based mechanism. Then, window handling is discussed in detail based on flow data export characteristics. An overview of the implementation of the integrated window-based delay extraction is given. The last section of this chapter proposes a profile-based window dimensioning method, which estimates the memory consumption and OWD sample count that results from different window sizes.

While Chapter 4 and Chapter 5 highlight the need for certain profile parameters, Chapter 6 details the overall exporter profile approach and profile creation. It analyzes parameter dependencies and gives a specification of the profile. Furthermore, it systematically elaborates different approaches for profile creation and motivates the flow-based profile creation approach that has been selected. A seven step iterative profile creation process is presented and algorithms specifically designed for certain profiling steps are detailed.

Chapter 7 shows the applicability and evaluation of the flow-based OWD measurement approach based on data collected in a global enterprise network. It describes first the evaluation scenario consisting of three locations in Germany, France and Australia. Second, the accuracy of flow-based OWD measurement is addressed by evaluating the profile-based error compensation and quantification methods. This includes the creation of error related profile parameters and the comparison of flow-based OWD measurements with active reference measurements. Furthermore, methods for profile-based confidence interval calculation are evaluated. The third part of this chapter evaluates the profile-based window dimensioning method for the estimation of memory requirements and the number of OWD samples.

Chapter 8 concludes this thesis and gives an outlook to future work.

2 Delay Measurement in IP-based Networks

The first section of this chapter introduces the fundamentals of IP-based network protocols, routers, and network structures. The second section presents metrological basics on measurement errors and their propagation, which will be input for error considerations throughout the following chapters. The third section presents delay measurement approaches for IP-based networks.

2.1 IP-based Networks

This section introduces the fundamentals of Internet protocols, router architectures, and network structures. More detailed information is available in computer network literature [13, 14].

2.1.1 Internet Protocol Stack

The Internet protocol stack allows running applications with different requirements and behavior over different network technologies. It consists of the *Internet Protocol* (IP) layer, the transport layer and the application layer. As the well-known hourglass model highlights, IP is the convergence layer between different sub-IP layer network technologies and different transport layer protocols. Internet protocols are specified by the *Internet Engineering Task Force* (IETF) and published in *Request for Comments* (RFC) documents.

The IP layer itself provides a connectionless packet-based communication service. In terms of addressing, IP addresses define a network endpoint, which is a host interface. Packets are forwarded in a hop-by-hop fashion by routers, which determine routes through meshed networks using routing protocols. Components that provide functionality beyond forwarding (e.g., changing addresses, processing higher layer protocols) are called *middleboxes* [RFC 3234]. IP packets are of variable length. The maximum possible amount of data the sub-IP layer can handle in one packet, the *Maximum Transmission Unit* (MTU), limits the maximum IP packet size. This can lead to the effect that packets have to be fragmented, when the MTU on a link becomes smaller on the path. In such cases IP packets are reassembled at the destination. Several mechanisms for providing certain levels of *Quality of Service* (QoS) are defined in the IP layer. The QoS mechanism *Differentiated Services* (DiffServ) [RFC 2474], where traffic is assigned to different traffic classes, is most widely used.

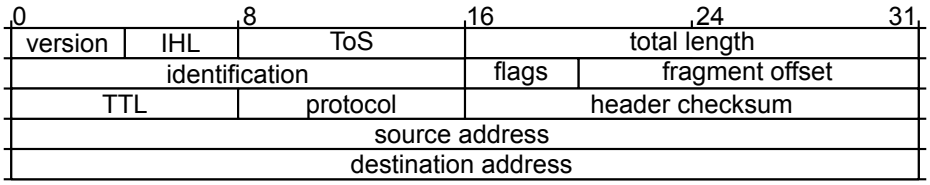


Figure 2.1: IPv4 header format

In order to understand network monitoring and especially flow capturing mechanisms, details about IP are introduced. Figure 2.1 shows the *Internet Protocol version 4* (IPv4) packet header [RFC 791] with each row representing a 32-bit word. The first word contains the IP version number, the *IP Header Length* (IHL), the *Type of Service* (ToS) byte, and the total length of the IP packet. The ToS byte initially indicated the QoS parameters that apply to the packet. While this field is still referred to as ToS byte, its definition has been changed: Six bits are defined as *Differentiated Services Code Point* (DSCP) [RFC 2474], which indicates the DiffServ class, and two bits are defined as *Explicit Congestion Notification* (ECN) [RFC 3168] flags. ECN is a mechanism that allows routers to mark packets in case of network congestion. Fields of the second word of the header are dedicated to fragmentation: an identification that is used for reassembly at the destination, flags defining whether or not to fragment a packet, and the fragment offset. The third word of the header starts with the *Time to Live* (TTL) value, which is decremented at each router. If its value reaches zero, routers have to drop the packet in order to prevent routing loops. The protocol field defines the protocol type that is carried by the IPv4 packet. The header checksum allows for IP header integrity checks. Finally, the fourth and fifth words give the source and destination address. In addition to the format shown, options can be appended by adding further 32-bit words. However, in most cases no options and the minimum IP header of 20 bytes length are used.

Since the address space of the currently widely deployed IPv4 is almost exhausted, the successor *Internet Protocol version 6* (IPv6) [RFC 2460] is increasingly used. While one of the main advantages of IPv6 over IPv4 is the extension of the address length from 32 bit to 128 bit, also the header format and other specifications changed. Figure 2.2 shows the format of the IPv6 header. It contains fewer fields than the IPv4 header since some fields became optional and have been moved to IPv6 extension headers. The first 32-bit word of the header contains the version number and the traffic class, which corresponds to the IPv4 ToS byte. The remaining 20 bits are defined as flow label in order to tag packets that belong to a flow independently of transport protocol port numbers (see Section 3.1.1 for details). The second word gives the payload length, next header, and hop limit. The next header indicates the type of the first extension header, if any, otherwise the payload type. The hop limit corresponds to the IPv4 TTL. The remaining words contain source and destination address. Without extension headers the IPv6 header has a size of 40 bytes.

There are different transport protocols, which can run on top of IP, that are implemented in the operating system of end systems. The most important transport protocols are the *User Datagram Protocol* (UDP) [RFC 768], the *Transmission Control Protocol* (TCP) [RFC 793], and the *Stream Control Transmission Protocol* (SCTP) [RFC 4960]. UDP defines an unreliable datagram service and is typically used for real-time applications, for which packet retransmis-

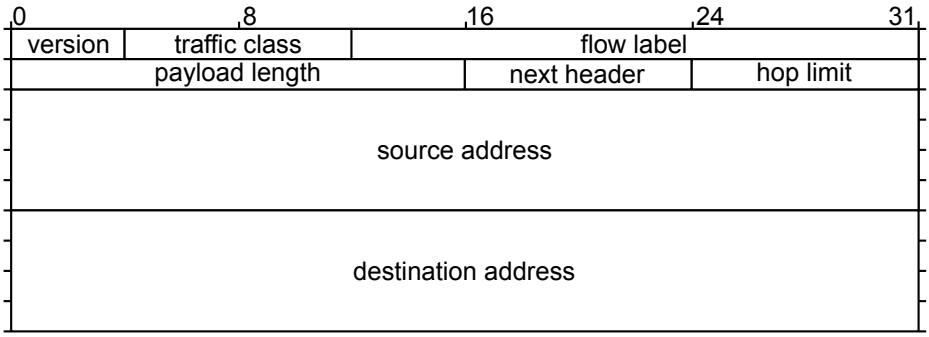


Figure 2.2: IPv6 header format

sion is infeasible, and signaling applications where only few messages are exchanged (e.g., the *Domain Name System* (DNS) [RFC 1035]). In contrast, TCP provides a reliable stream transmission service with flow control, congestion control, and guaranteed in-order delivery (e.g., for the *Hypertext Transfer Protocol* (HTTP) [RFC 2616]). TCP is connection oriented and performs a three-way handshake at connection setup. TCP is the most widely used transport protocol today employed for interactive applications as well as for bulk data transfers. SCTP provides a datagram service and has several options such that the degree of in-order delivery and reliability [RFC 3758] can be tuned to fit the application's needs. Additionally, it provides multihoming support. SCTP is hardly deployed today, but due to its features it might become an important protocol for flow data transport in future. All transport protocols add 16-bit source and destination port numbers as addressing layer in addition to the IP address for being able to distinguish different applications running on the same host. Transport protocol specifications, especially port numbers, are not affected by the transition from IPv4 to IPv6.

IP packets can also carry protocol messages of the *Internet Control Message Protocol* (ICMP) [RFC 792] or *Internet Control Message Protocol, version 6* (ICMPv6) [RFC 4443]. These companion protocols serve signaling purposes and belong to the IP layer itself, not to the transport layer. Routing and fragmentation problems as well as echo requests and responses (ping) are signaled by ICMP and ICMPv6, respectively.

2.1.2 Router Functionality and Architectures

Routers are network nodes that operate in the IP layer and forward packets mainly based on their destination address to a next hop router or to local networks. It is a property of the Internet architecture to keep forwarding as simple as possible in order to allow high throughput with limited effort. When more packets have to be sent on a link than bandwidth is available, routers buffer packets in queues in order to deal with temporary congestion.

[RFC 1812] defines the tasks a router has to perform. These can be separated in forwarding tasks that have to be performed for every packet, and control tasks that deal with route calculation, management, and monitoring [RFC 3654]. Figure 2.3 highlights these different task classes. The chain of forwarding tasks starts with receiving packets from the ingress interfaces in the *Receive* (RX) stage and delivering them to the *Forwarding Engine* (FE) that performs actions

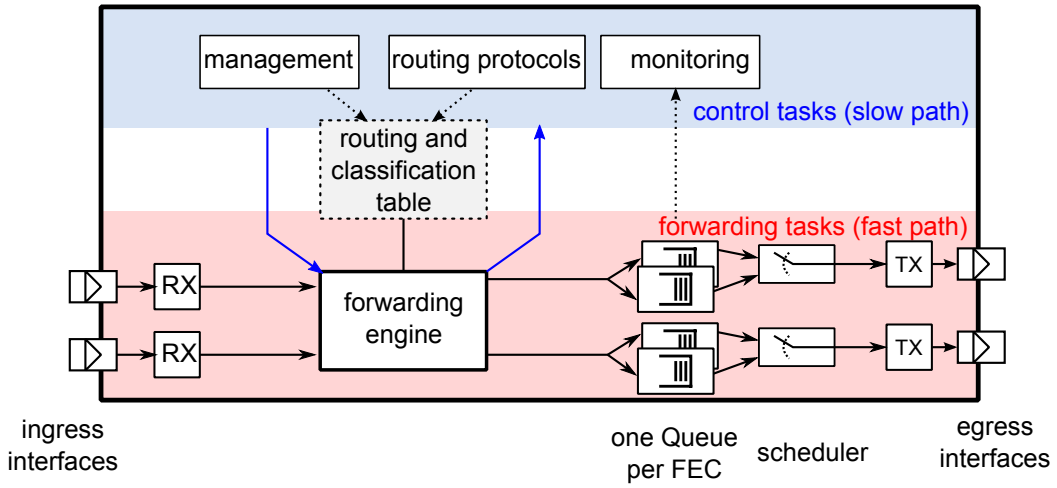


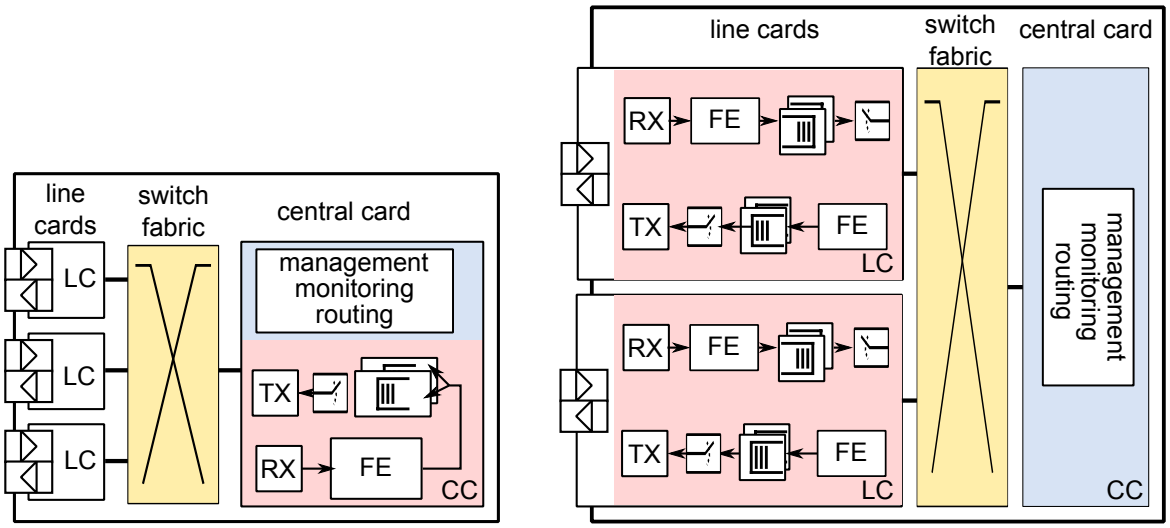
Figure 2.3: Router functions

on the packets. These actions take the routing and classification table into account and result in an association of a packet with a certain *Forwarding Equivalence Class* (FEC), which defines the egress interface and the priority of the packet. The FE puts all packets of the same FEC into the same queue[1]. A scheduler takes the packets from the queues according to their priority by applying a defined scheduling algorithm and sends them to the *Transmit* (TX) stage, which sends the packets on the egress interface.

Basic router functionality means performing forwarding decisions based on a routing table. However, in most scenarios much more information is taken into account for packet classification, such as policies defined in an *Access Control List* (ACL), or QoS classes. The forwarding engine might even change fields in the packet headers (e.g., for *Network Address Translation* (NAT)). In general, all actions of the forwarding engine are based on rules stored in the routing and classification table, which is managed by control functions, such as routing protocols and configuration from network management. Another interface between control tasks and forwarding tasks is the packet path between the two, which is indicated by the blue arrows in Figure 2.3. This path is necessary for network communication of the control tasks themselves or packets that cannot be classified or handled by the forwarding engine. From an implementation perspective, certain router tasks are implemented in a very efficient way by using specialized components with limited functionality (*fast path*), while other tasks are implemented on general purpose components (*slow path*). In most cases forwarding tasks are realized in the fast path and control tasks in the slow path.

There is a wide variety of routers in IP-based networks, ranging from small *Customer Premises Equipment* (CPE) routers dealing with some Mbit/s up to large core routers handling multiple Tbit/s. Obviously, this results in different performance and scalability requirements, which leads to different architectures. Small to medium size routers are implemented based on a monolithic architecture (Figure 2.4 *a*), where all control and forwarding functionality is implemented on a *Central Card* (CC). The latter consists of a single CPU or multiple processors and specialized hardware for performing the tasks. Each *Line Card* (LC) performs only basic func-

[1] This is the ideal case if a router implements enough queues. There might be limitations.



a) monolithic: central card with FE

b) distributed: forwarding on line cards

Figure 2.4: Router architectures

tionality and is connected to the CC by a switching network (e.g., from a simple bus in CPE up to backplanes with multi-stage switching networks in medium size routers).

Large high performance routers are realized based on distributed architectures: forwarding is not performed on the CC, but directly on each LC (Figure 2.4 *b*). Each LC has a separate ingress and egress path each containing FE, queues, and scheduler. In high performance routers the forwarding engine is based on *Application Specific Integrated Circuit* (ASIC) components or on network processors, which are specialized processors that allow for feature extensibility by software updates [15]. In order to provide high switching capacity, the switch fabric is realized as multi stage switching network[2]. In distributed architectures, control functions are performed on the CC, which writes classification table updates and other configuration data to the LC. An overview of queuing strategies and architectures is presented in [17]. Details on implementation of a high performance router can be found in [18] and [19]. The monolithic and distributed architecture are corner cases and any partially distributed architecture is possible as well.

The considerations of router architectures have shown that there are different paths a packet can take on its way through a router. Depending on the classification result, it is put in different queues, which might be subject to different queuing disciplines and scheduling priorities. For network measurement, which relies on information generated in a router, it is important to take into account the position of the measurement function, especially in distributed architectures. Section 3.3 details where flow capturing functions can be located and how this can impact delay measurement.

[2]There are even routers with extensible switching networks [16].

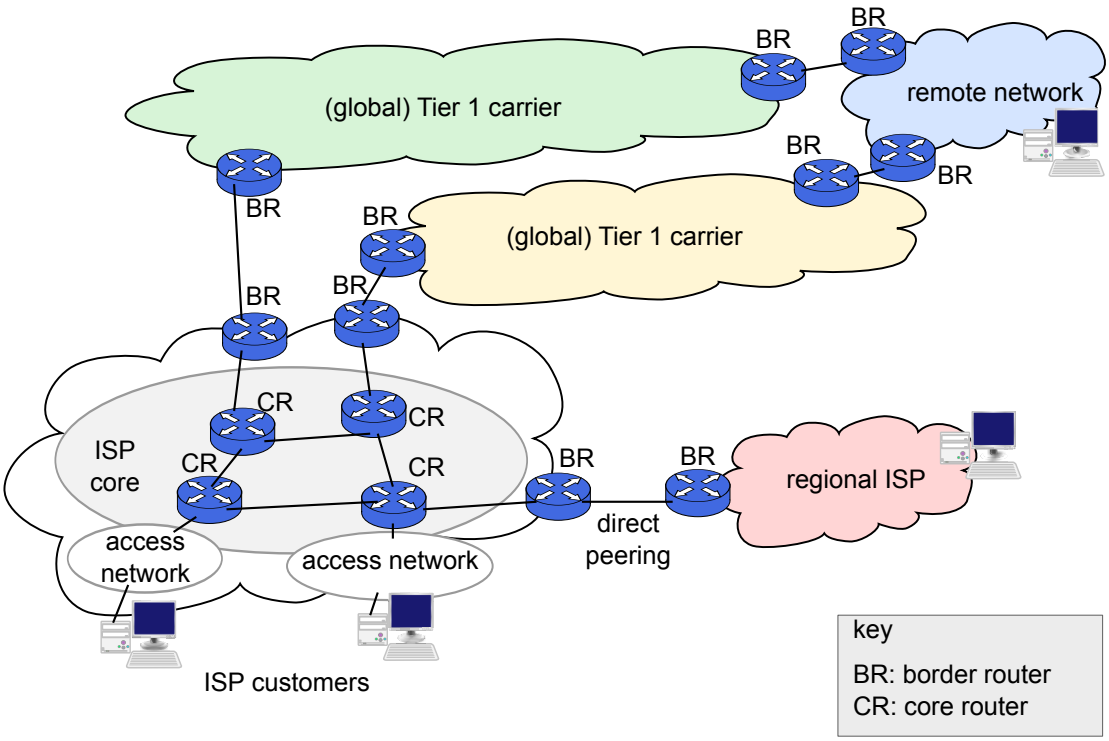


Figure 2.5: Internet topology principle

2.1.3 Network Structures and Routing

The Internet consists of independent networks. Each of them is called an *Autonomous System* (AS). Connectivity between those networks is achieved by interconnections at peering points. Since an *Internet Service Provider* (ISP) can only establish direct peerings with a few AS, it peers with global carriers (Tier 1 carriers) that provide transit service to remote networks (see Figure 2.5). A router at a peering point is called *Border Router* (BR) and communicates with other BRs using the *Border Gateway Protocol* (BGP) [RFC 4271] for exchanging routing information. Figure 2.5 shows that ISPs typically connect to several transit carriers for redundancy and path optimization purposes. The path taken by the outgoing packets to a remote network can be different from the path taken by the incoming packets from this remote network. Such asymmetric routing [8] is caused by the fact that the source AS decides which carrier to use. In contrast to a BR, a *Core Router* (CR) is located in the ISP's core network that connects the different access networks among each other as well as the border routers.

Although enterprise networks are also IP-based and provide global connectivity between enterprise branch locations, they are not part of the Internet, but form a closed network. Figure 2.6 shows an exemplary enterprise network topology that connects branch locations to a *Data Center* (DC). Since network-based applications in enterprises are often business critical, ensuring proper network connectivity and performance is essential. The depicted network provides full router and link redundancy, i.e., if a single router or link fails, there will still be full connectivity in the enterprise network.

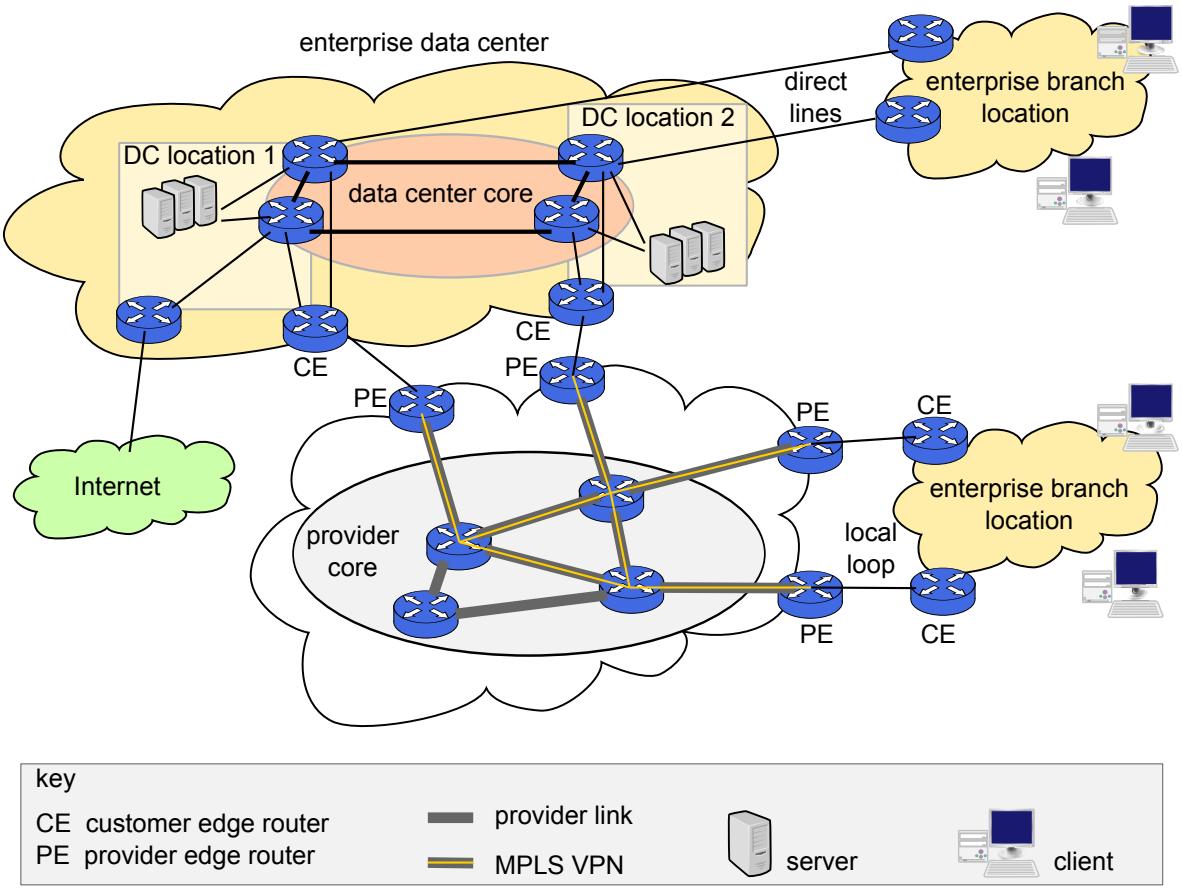


Figure 2.6: Enterprise network topology principle

The enterprise network depicted in Figure 2.6 uses a *Multi Protocol Label Switching* (MPLS) *Virtual Private Network* (VPN) for connecting the lower branch location to the data center. MPLS [RFC 3031] is a sub-IP layer protocol that provides connection-oriented transport between edge routers. As shown in Figure 2.6, the provider sets up MPLS paths for the enterprise network in its *Wide Area Network* (WAN) up to the network edge where the *Provider Edge* (PE) routers reside. These are connected via the local loop to the *Customer Edge* (CE) routers, which provide the interconnection point for the enterprise to its *Local Area Network* (LAN). From the enterprise perspective there is direct connectivity between CE routers and the enterprise routers do not have to care about the underlying MPLS network. CE routers are often owned and managed by the MPLS provider, but enterprise staff can still obtain monitoring data from them. Another possibility to connect branch locations is using direct lines, as depicted for the upper location in Figure 2.6. The quality of the transport service the provider offers to the enterprise is specified in a *Service Level Agreement* (SLA) that defines the maximum throughput, availability, delay and packet loss. SLAs can also be specified between enterprise departments that offer services to each other (e.g., End-to-End network performance or server response time).

Data centers of large enterprises are actually realized using at least two locations in some distance to provide geographical redundancy in case of disasters. A data center core network with high bandwidth connections enables the servers of each location to keep their data synchronized

with each other. This is a requirement for failover (hot standby). In order to reach the Internet, an enterprise network has several connections to ISPs, secured with firewalls.

If there is no failure in the network, there are at least two disjoint paths for reaching remote sites. In order to use the available bandwidth efficiently, routers can be configured to use multiple routes (multipath routing) instead of only one for load balancing. A simple scheme that performs load balancing on per packet basis is called *Equal-Cost Multi-Path* (ECMP) routing, which has several issues [RFC 2991] (e.g., regarding packet reordering) and is therefore considered as deprecated. Alternatives are flow-based load balancing, which e.g., performs path decision based on hash values obtained from header fields [RFC 2992]. More information on enterprise network design can be found in [20] and [21].

2.2 Metrology

Measurements are important for correctly quantifying the amount and/or the quality of goods or services for both, manufacturing as well as trading. Thus, measurements have been for centuries of great economic importance. Metrology is the "science of measurement and its application" [22] that covers definitions and basic principles of measurement. In the following, the first subsection details the fundamental terms of measurement. The second subsection details the principle of error propagation and confidence in measurements.

2.2.1 Fundamental Terms

In order to define worldwide consistent terms and concepts, several standardization bodies are concerned with metrology. An important standardization organization in this field is the *Bureau international des poids et mesures* (BIPM) [23] that was created in 1875 in the *Convention of the Metre* treaty. The task of the BIPM is "to ensure world-wide uniformity of measurements and their traceability to the International System of Units (SI)" [23]. This well-known system is based on the *International System of Quantities* (ISQ), which itself is defined in standards of the *International Organization for Standardization* (ISO) [24] and the *International Electrotechnical Commission* (IEC)[25].

The SI standards and the ISQ, however, do not define metrological terms itself, but the relation between quantities and their units. Therefore, the aforementioned bodies as well as additional organizations compose the *Joined Committee for Guides in Metrology* (JCGM) [26]. This committee has according to its charter [27] two main work items that also form its two working groups:

- The *International Vocabulary of Metrology*
  (*Vocabulaire International de Métrologie* (VIM)) [22, 28]
- The *Guide to the Expression of Uncertainty in Measurement* (GUM) [29]

The following definitions and citations of selected metrological terms are taken from the VIM [22, 28].

Measurement is the "process of experimentally obtaining one or more quantity values that can reasonably be attributed to a quantity"[28]. So measurement is about quantities and values. A *quantity* is a "property of a phenomenon, body or substance where the property has a magnitude that can be expressed as a number and a reference" and the "quantity intended to be measured" is called *measurand*[28]. This thesis considers mainly measurands like time, delay, and loss ratio where the reference is a *measurement unit* (and not reference material or something else). The *measurement unit* is "a real scalar quantity, defined and adopted by convention" (e.g., second).

Measurement obtains the *measured value*, which is "the quantity value representing a measurement result"[28], denoted by x. In general, repeated measurements of the same quantity even under the same conditions result in different measured values. For interpreting and handling these effects, there are two fundamentally different approaches, which are according to the VIM the *error approach* and the *uncertainty approach*. With the error approach, the true value is in general unknown and the measured result is a random variable. Systematic and random error can be distinguished. Contrary, in the uncertainty approach the quantity is considered as a random variable whose values are represented by a *Probability Density Function* (PDF) derived from available knowledge whereas the measured values are considered facts [30, 31]. Users of the error approach are called *Frequentists* and users of the uncertainty approach are called *Bayesians*. The approaches mainly differ in terminology, but this also led to confusion since the first versions of the GUM did not clearly state this difference, which demanded for correct interpretation based on the Frequentists' and Bayesians' view [32].

Since this thesis focuses on quantities measured in networks based on digital devices, which often allow for clear separation of random and systematic errors, the "traditional" error approach has been chosen. According to this terminology, there is a *true quantity value* (or briefly *true value*), which is the "quantity value consistent with the definition of a quantity"[28]. This true value x_t can never be known or measured due to the *measurement error* e_x (2.1).

$$e_\mathrm{x} = x - x_\mathrm{t} \tag{2.1}$$

A higher error leads to a lower *accuracy*, which is the "closeness of agreement between a measured quantity value and a true quantity value of a measurand"[28]. Measurement errors, however, do not include *mistakes* or *blunder* [33], which are *illegitimate errors* that must be avoided and/or detected in the measurement process. Figure 2.7 depicts the relations between quantity values and errors. As shown in Figure 2.7 the measurement error e_x can be separated into *systematic error* $e_\mathrm{s,x}$ and *random error* $e_\mathrm{r,x}$:

$$e_\mathrm{x} = e_\mathrm{s,x} + e_\mathrm{r,x} \tag{2.2}$$

The *random error* is the component that in "replicate measurements varies in an unpredictable manner"[28]. The random error therefore follows a PDF, as depicted in the example of Figure 2.7 for a normal distribution. As shown the random error has zero mean and adds a certain measurement *uncertainty*.

In contrast to the random error, the *systematic error* "remains constant or varies in a predictable manner"[28]. Due to knowledge about the measurement, one part of the systematic error, the

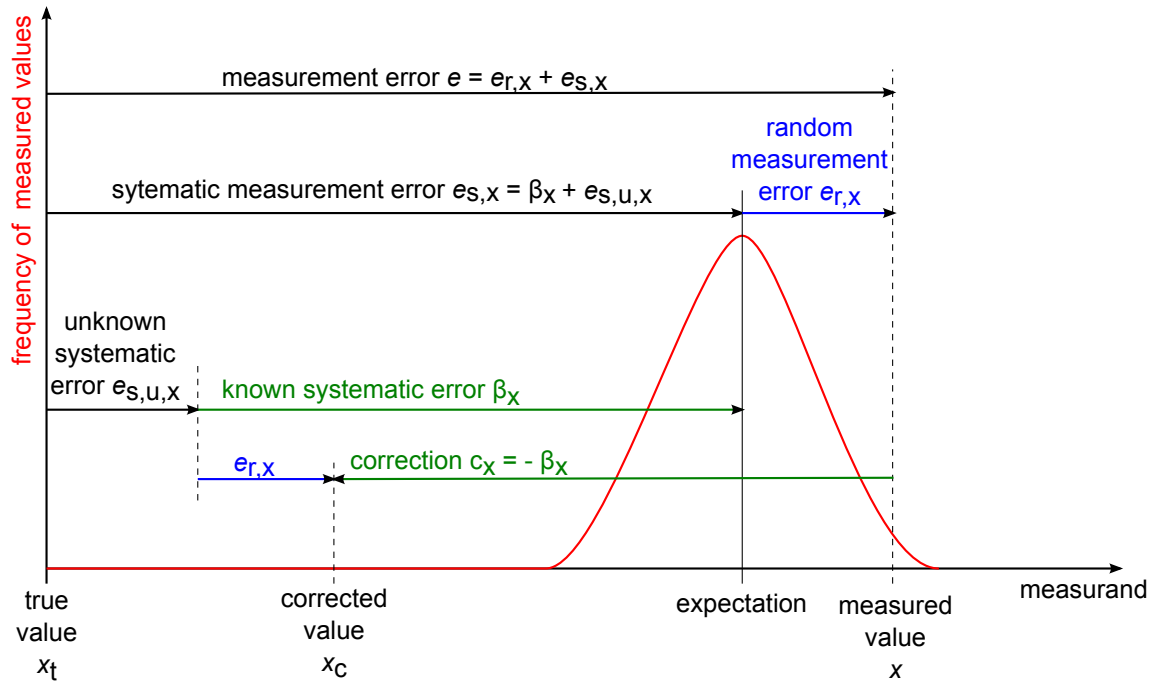


Figure 2.7: Illustration of measurement values and errors; based on [DIN 1319-1]

bias β_x, can be known and potentially corrected. However, there will be always an unknown part ($e_{s,u}$) of the systematic error $e_{s,x}$:

$$e_{s,x} = \beta_x + e_{s,u} \tag{2.3}$$

The known or estimated systematic error β_x can be compensated using a *correction* c_x= -β_x . This results in the corrected value x_c:

$$x_c = x + c_x \tag{2.4}$$

With more and more measurement samples, the mean value of the samples will approach the expectation of the random error PDF due to the law of large numbers. Thus, the random error can be eliminated by collecting a large number of samples under the same conditions. A low random error means high *precision*, which is defined as the "closeness of agreement between ... measured quantity values obtained by replicate measurements on the same or similar objects..."[28]. However, precision must not be confused with accuracy, since measurements with high precision might still be inaccurate due to systematic error. Precision is often reduced by limited *resolution*, which is defined as "smallest change in a quantity being measured that causes a perceptible change in the corresponding indication"[28].

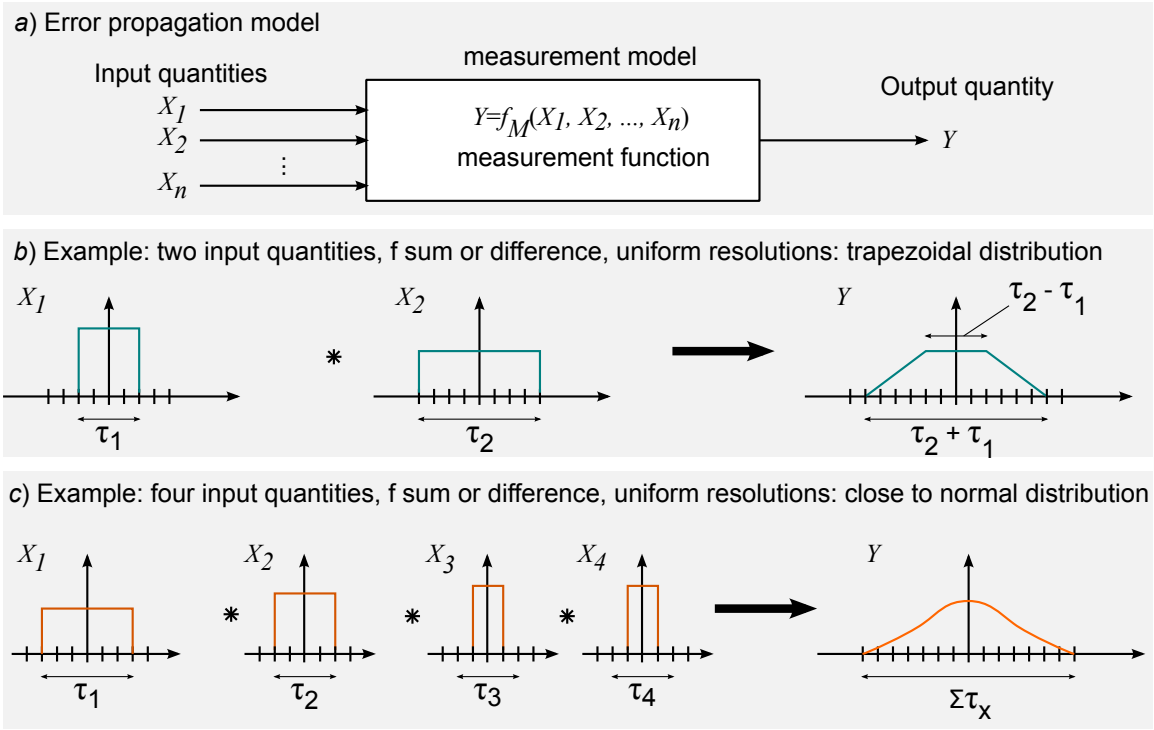


Figure 2.8: Error propagation in a measurement system

2.2.2 Error Propagation and Confidence

Measurands are often not obtained from single measurements, but a quantity is derived from several other measured quantities. This general model is illustrated in Figure 2.8 *a*), where an output quantity Y is obtained from several input quantities $X_1..X_n$. The relation between input and output quantities is defined by a *measurement model*, which can be described by a general *measurement function* $Y = f_M(X_1,...,X_n)$. Measurement errors in obtaining the input quantities affect the output quantity depending on the measurement function according to the laws of *error propagation*.

If the measurement function performs addition or subtraction of input quantity values, the resulting distribution of the random error corresponds to the *convolution* of the PDFs of the input quantities [34]. This only holds if the errors of the input quantity are *independent*. Figure 2.8 illustrates this in the sections *b*) and *c*) for two examples using uniform distributions for random errors, which are common in digital systems. The example in Figure 2.8 *b*) shows that in the case of two input quantities the convolution of two uniform distributions results in a trapezoidal distribution. This distribution has a range τ, which is the sum of the ranges of the input quantity distributions.

The more input quantities, the closer the resulting distributions gets to the normal distribution, as illustrated in the example of Figure 2.8 *c*). The central limit theorem describes this effect. Such a distribution cannot be as easily calculated as the trapezoidal one and requires methods such as the *Laplace-Stieltjes Transform* [35].

A lot of measurements exhibit random errors that can be modelled using the normal distribution. This is an unbounded distribution, i.e., an absolute error cannot be determined. The standard deviation σ of the distribution is given as *standard error* (or *standard uncertainty*). Due to the convolution of PDFs, the variances of the distributions are added if the measurement functions consist of summation or subtraction only. This allows for calculating the standard deviation of an output quantity σ_Y easily from the standard deviations of the input quantities σ_{X_i}.

$$\sigma_Y = \sqrt{\sum_{i=1}^{n} \sigma_{X_i}^2} \tag{2.5}$$

If there are systematic errors in input quantities, these errors propagate according to the measurement function. For example, if the measurement function is a summation, the systematic errors sum up to an overall systematic error of the output quantity. For measurement functions other than summations or differences, the input quantity values itself impact the resulting distribution. This requires taking the partial derivatives of the measurement function into account, which is described by a general law of error distribution. More information on such cases can be found in [33] and [34].

Often several measurement samples are obtained for a quantity in a measurement series in order to obtain the best estimate of the true value as mean value from all measurement samples. If the distribution of the random error is known, an interval can be given in which the true value can be found with a certain *confidence level* γ. Such an interval is called *confidence interval* of the mean value and typically a confidence level of 95% is chosen. The *significance level* α ($\gamma = 1 - \alpha$) is the probability that the true value is not within the calculated inverval.

In general, there is an infinite number of possibilities to specify a confidence interval for an expectation given a mean value from a measurement series. Most commonly, a symmetric confidence interval is calculated with the mean value in the middle. Another possibility is to calculate the shortest confidence interval, which differs from the symmetric one in the case of skewed distributions.

A well-known formula for a symmetric confidence interval uses Student's T-distribution (or simply T-distribution) [36]. This is suitable for cases where the random error follows a normal distribution with unknown standard deviation σ. The interval calculation (Equation 2.6) takes the number of samples n into account, more precisely the $1 - \alpha/2$ quantile of the t-distribution with $n-1$ degrees of freedom ($t_{1-\alpha/2}(n-1)$) is part of the formula. Furthermore, the t-based confidence interval formula contains the mean value of the n collected samples $\overline{X}$ and the empirical standard deviation s .

$$\left[\overline{X} \pm t_{1-\alpha/2}(n-1)\frac{s}{\sqrt{n}}\right]. \tag{2.6}$$

If the standard deviation σ is known, another formula based on the normal distribution applies (Equation 2.7). Instead of the t-distribution quantiles it contains the $\alpha/2$ quantile of the nor-

mal distribution ($z_{1-\alpha/2}$). Furthermore, the known standard deviation σ replaces the empirical standard deviation s compared to the previous formula.

$$\left[\overline{X} \pm z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}\right] \tag{2.7}$$

For an unknown distribution Equation (2.7) gives an approximate confidence interval, if more than 30 samples are obtained [36] and σ is known.

In case of non-normal distributions and few samples, analytical approximations, e.g., according to the uncertainty guidelines of the GUM [29], can be used. Alternatively, [30] introduces a Monte-Carlo method to obtain confidence bounds for arbitrary distributions.

2.3 Delay Measurement

This section details delay measurement approaches by first introducing measurands and metrics used in literature and measurement systems. The second and third subsection present active and passive measurement approaches deployed in networks and current research in this area. The last subsection highlights clock synchronization, which is an important point in delay measurement. Extensive information on network measurement in general can be found for example in [37] and [38].

2.3.1 Measurands and Metrics

One-Way Delay (OWD) on a network path is composed of several contributing delays, as shown in Figure 2.9 (see also [39, 40, 41]). First, there is the path delay consisting of the *transmission delay* l_{T} for sending/receiving the packet and the *propagation delay* l_{L} on the link. Second, there is the router delay, which is the sum of *processing delay* l_{P} and *queuing delay* l_{Q}. l_{P} is the time that is spent for processing packets in the Forwarding Engine (FE), which is mainly independent of network load. l_{Q} is the time that packets spend in queues, which depends on queue levels and therefore on network load. In modular routers with distributed architectures (see Section 2.1.2), there are several FEs and queues that contribute to the overall l_{P} and l_{Q}. Delays in end systems can be considered in a similar way. There, the IP stack in the operating system corresponds to the FE that causes l_{P} and the interface queues cause l_{Q}. Inside routers or operating systems there is no notion of transmission time since packet arrival and departure times are discrete events. Further information on OWD components and typical values for routers can be found in [42].

Figure 2.9 highlights that a definition for measuring OWD is required that clearly defines between which points of network elements or end systems OWD is measured in order to know which delay components contribute to the OWD. There exist several standards for network measurement, especially delay measurement, from the IETF and ITU-T.

In the IETF, the *IP Performance Metrics* (IPPM) working group [43] has released several RFCs on metrics, their aggregation, and associated protocols. The most important IPPM standards in

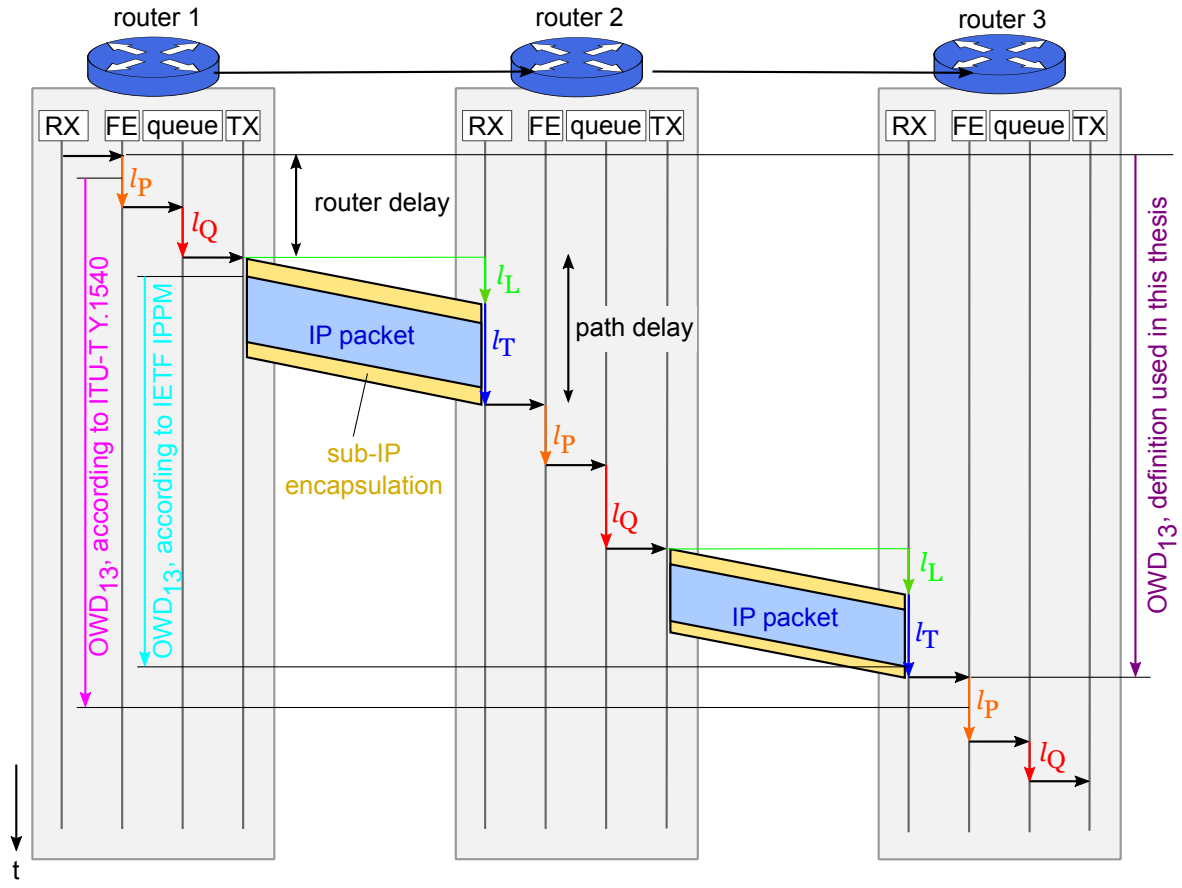


Figure 2.9: One-way delay definitions

the context of this thesis are the One-way Delay Metric [RFC 2679] and the Round-trip Delay Metric [RFC 2681] that are all based on a common framework document [RFC 2330]. All these metrics take into account that measurements depend on the characteristics of the packets used for measurement. This is generally denoted as that a metric is consistent within the same type of packets (”Type-P“). Therefore, the definition reflects the fact that different packets might be treated differently in routers on a network path (e.g., different ToS values, different TCP port numbers).

[RFC 2679] defines how to measure single samples of a “Type-P-One-way-Delay” across Internet paths. Additionally, it specifies a metric consisting of several samples from Poisson sampling and statistics for such measurements. Delay values are assumed to be calculated between two hosts *Src* and *Dst*, i.e., for an end-to-end path. [RFC 2679] defines One-way Delay as follows: “For a real number dT, »the *Type-P-One-way-Delay* from Src to Dst at T is dT« means that Src sent the first bit of a Type-P packet to Dst at *wire-time* T and that Dst received the last bit of that packet at wire-time T+dT.” Wire-time refers to the time at which the bits appear *at the network interface*. Figure 2.9 illustrates this definition for OWD_{13}, which is the OWD between router 1 and router 3. This figure also shows that a precise interpretation of the IPPM definition leads to the fact that timestamps of IP packet bits and not of the larger sub-IP encapsulation frames have to be obtained. It is practically impossible to measure such a timestamp accurately directly at the interface. The *Round Trip Time* (RTT) (in IPPM called “Round-trip Delay Met-

ric") is defined analogously in [RFC 2681], while it additionally specifies the requirement, that the destination immediately sends back a response Type-P packet.

The ITU-T defines OWD (called *IP Packet Transfer Delay* (IPTD)) as time between two packet transfer events [Y.1540]. The latter corresponds to packet observations at measurement points. This is a similar metric definition compared to IPPM, however, the timestamps are not defined on a per-bit basis at the interface, but from an IP stack point of view (but allows for approximations using timestamps from physical interfaces). However, [Y.1540] does not define exactly if the measurement point is before/after the FE/queues[3]. Figure 2.9 shows the ITU-T definition for OWD_{13} and indicates that the exact location for obtaining the timestamps is not clearly specified by showing the Y.1540 timestamps taken at *some* point in the FE. From Figure 2.9 it becomes clear that the IETF and ITU-T OWD definitions differ.

In the context of this thesis, OWD is defined based on timestamps that are taken when the packet *enters the ingress FE* of a router: *"The OWD from SRC to DST for a packet at T is dT" means that the packet enters the ingress forwarding engine of SRC at T and enters the forwarding engine at DST at time T+dT.* This definition takes flow capturing properties into account: Since flow capturing is performed in the FE and is a router implemented feature, this definition avoids any ambiguities when considering different router architectures.

2.3.2 Active Measurement

Active measurement approaches actively generate packets that are sent through the network and collect send and receive timestamps for calculating OWD or RTT. While the IPPM RFCs [RFC 2679, RFC 2681] define related metrics, these standards do not define the measurement methods and devices themselves.

Besides the rudimentary well-known "ping" program that is based on ICMP echo requests/replies, there are a lot of sophisticated active delay measurement tools used for network monitoring today. One example is *Cisco IOS IP Service Level Agreements* (IPSLA) [44, 37], which is implemented in router software. It can perform various active network layer measurements (e.g., OWD, RTT, jitter), and even application layer measurements (e.g., DNS and HTTP). With IPSLA, a router configured as *IPSLA probe* sends test packets to a router configured as *IPSLA responder*. The responder takes timestamps directly after receiving the packet and includes them in a response packet. If both routers' clocks are properly synchronized, this mechanism allows for accurate OWD measurements on the forward and return path between probe and responder. IPSLA measurement results are retrieved via SNMP by network monitoring systems for compiling network performance reports and alerts. In large deployments, probes are realized on *shadow routers*, which are dedicated to IPSLA measurements only. This allows for setups where an IPSLA probe located in a data center can perform measurements to several thousand responders at branch locations (star topology).

In addition to proprietary active measurement systems, the IETF IPPM WG defined a *One-Way Active Measurement Protocol* (OWAMP) in [RFC 4656]. This standard defines an OWAMP

[3] [Y.1540] notes that "The exact location of the IP service MP [means: Measurement Point] within the IP protocol stack is for further study"

test protocol for performing one-way measurements between a sender and a receiver as well as an OWAMP control protocol for setting up such measurements and fetching the results. The OWAMP control protocol allows defining measurement schedules and provides different security features, which allows for reliable measurements even in untrusted environments (e.g., interdomain scenarios).

OWAMP is defined to perform measurements in one direction, which also enables to perform measurement in both ways of a path by setting up two OWAMP measurements. In order to allow for measurements of two-way metrics in case the clocks are not synchronized, the *Two-Way Active Measurement Protocol* (TWAMP) [RFC 5357] has been defined. TWAMP is based on OWAMP, but defines the functionality of a *reflector* that receives and returns test packets.

With active delay measurements, delay samples can be obtained between defined measurement points at scheduled time instants with known traffic. Thus, a complete and deterministic delay analysis of network paths can be obtained. However, active measurements have considerable drawbacks. First, they require that additional traffic is injected into the network, which requires some effort at probing devices and increases network load especially on low bandwidth links. It might therefore impact the network behavior itself. Second, the production traffic itself is not monitored, but only the artificially created probe traffic. Thus, active measurement results might not be directly related to network problems that production traffic experiences. Especially, measurement and production traffic can take different routes, might experience network characteristics at different times, or might be handled differently in routers. The latter is often countered by performing measurements using different probe traffic (ICMP, UDP and TCP as well as different QoS classes). However, this increases the effort even more and still does not provide the view on traffic as passive measurements do.

2.3.3 Passive Measurement

In contrast to active measurement, passive measurement approaches do not inject additional probe traffic into the network, but perform measurements based on the traffic that uses the network for productive purposes. For performing passive OWD measurement, timestamps of *the same packet* at different network locations have to be gathered, which is a challenging task due to packet similarities. Performing passive OWD measurement for all transmitted packets is impossible due to the high effort. Hence, only a fraction of packets is selected (sampled) for measurement.

RTT measurement based on observations at one location require that timestamps of packets are compared that have been immediately send back by the receiver. Typically, passive RTT measurement selects only packets for which this requirement holds (e.g., packets from the TCP handshake carrying SYN or SYN-ACK flags).

There are several commercial products for passive measurement that are typically attached to central switches and operate on the complete traffic for extracting performance metrics. Such sniffers (e.g., [45]) calculate network delays based on RTT obtained from the TCP handshake as well as other information on application performance from the request-response timings in the packet trace. However, such analysis cannot measure OWD and requires processing of

the complete packet trace, which leads to a high processing effort. Thus, they are often too expensive for being deployed at every location for comprehensive network monitoring.

In the last years, a lot of work has been conducted in packet sampling based measurement of OWD. The idea is to sample the same packets at two different locations in the network and to deliver packet descriptions and receive timestamps to a central location for OWD calculation. A use case for packet sampling is described in [RFC 5472] as "coordinated packet selection". [46] describes the design and implementation of a system for OWD calculation based on packet sampling. [47] presents a similar system, which uses the IPFIX protocol for transferring packet IDs and timestamps. Kompella et al. [48] propose a special processing scheme for calculating OWD with microsecond granularity from packet samples, which is focused on being implementable in hardware.

As presented, current approaches for passive delay measurement either can determine only the RTT from a packet trace or require changes to router hardware for coordinated sampling. This is a problem from the deployment point of view, especially in network scenarios with many locations for which OWD measurement is required, such as, e.g., in enterprise networks.

2.3.4 Clock Errors and Compensation

Clocks in digital devices are mostly realized using a *Real-Time Clock* (RTC) component that is based on a quartz oscillator. These clocks are inherently imprecise and without synchronization mechanisms their timing information neither corresponds to a global reference time (such as *Coordinated Universal Time* (UTC)), nor their frequency is synchronized. This leads to offset, skew, and drift that are defined as follows (aligned with [RFC 2330]).

Let t be the true time and $c(t)$ be the time of the clock considered. Then the *offset* $\omega_c(t)$ of the clock at a particular point in time is defined as

$$\omega_c(t) = c(t) - t. \tag{2.8}$$

A clock is accurate at a particular moment, if $\omega_c(t)$ is zero. As mentioned above, there is often a frequency difference, which can be quantified by the *skew* being the first derivative of ω_c with respect to t:

$$\varphi_c(t) = \omega_c(t)'. \tag{2.9}$$

If the skew is constant, the offset $\omega_c(t)$ is a linear function. This can be modeled by the constant φ_0 and the offset $\omega_0 = \omega(0)$ at $t = 0$. If skew is not constant, but varies, this effect is termed *drift*, which is defined as the second derivative of $\omega_c(t)$ with respect to t. A major source of drift is change of hardware temperature [49], which is unlikely to happen in well managed data centers. Thus, drift is negligible.

Several possibilities exist for synchronizing device clocks to global reference time. Besides the expensive solution of using atomic clocks, one of the most accurate methods is attaching a *Global Positioning System* (GPS) receiver that synchronizes to the atomic clocks of GPS satellites. Such receivers provide timing information directly to the device, e.g., via a *Pulse Per Second* (PPS) signal [RFC 1589, RFC 2783] over a serial interface.

Using GPS receivers requires that there is reception of signals from several GPS satellites, which is often not the case inside buildings. Additionally, this requires extra wiring to every device that should be synchronized. In order to reduce the effort, time servers are deployed that synchronize to GPS receivers and distribute timing information to other devices within a data center or network domain using time synchronization protocols over the existing IP-based network. Such protocols are the *Precision Time Protocol* (PTP) defined in the standard IEEE 1588 [50] intended for LANs and the Network Time Protocol (NTP) Network Time Protocol (NTP) version 3 [RFC 1305] or version 4 [RFC 5905] for LAN and WAN usage.

Time synchronization protocols and clients implementing these protocols do not only compensate the offset every time they get a reference time. They also try to synchronize frequencies by calculating skew and applying corrections gradually [51]. Further information on these three approaches including considerations for OWD measurements has been compiled by De Vito et al. in a survey paper [41]. This paper also summarizes the expected synchronization accuracy, which is $1\mu s$ for directly attached GPS, less than $1\mu s$ for IEEE 1588, 1 ms for NTP in LANs and 10-20 ms for NTP in WANs.

Hong et al. [52] recently evaluated the accuracy of public NTP servers on the Internet in more detail by comparing timestamps obtained by NTP with GPS timestamps. They found that the NTP median error is 2-5 ms. However, there is a long tail due to the stratum 1 time servers that actually should be well synchronized to external clocks (e.g., GPS) are badly synchronized and due to asymmetric delays that NTP assumes to be symmetric. Compared to the public Internet, there are less such effects in well managed enterprise networks and carrier networks. Therefore, an error of 2-5 ms without major deviations for NTP synchronized devices can be expected.

Despite the existence of several clock synchronization mechanisms, unsynchronized clocks in network measurement are still a problem. Especially if measurement relies on exact timestamps from two independent devices, as it is the case for OWD measurement. The following paragraphs highlight algorithms that compensate clock skew in data obtained for OWD measurement. The fundamental problem is that clock offset in principle can be calculated by taking delay measurements in both directions of two measurement points into account, however, the delay varies.

Paxson [53] addresses the problem of active OWD measurements with unsynchronized clocks. In the studied data sets clock skew was mostly linear, i.e., drift was negligible. The aim of this work was determining the skew for OWD measurements only without caring about the relative offsets. It is argued that relative offset is of less importance since the aim is capturing *network dynamic*, i.e., delay variations over time. Paxson's algorithm determines minimum values for a certain amount of time intervals, determines the median slope of all possible slopes between minimum values and does a statistical test for checking whether the skew determined from the median slope is plausible. Furthermore, detection of clock adjustment events has been considered.

Moon et al. [54] developed a linear-programming based algorithm for skew estimation. This algorithm is compared to three algorithms: Paxson's algorithm, a linear regression algorithm, and a piecewise minimum algorithm. Evaluations show that the linear-programming based algorithm and Paxson's algorithm perform best in estimating the skew. The linear-programming

based algorithm effort is assumed to have less variance than Paxson's algorithm based on simulation results.

Zhang et al. [55] propose a convex hull algorithm that is more complicated, but can deal with clock resets and gradual frequency adjustments, as performed by NTP. Khlifi and Grégoire [49] compare this algorithm with a simpler algorithms that can deal with clock resets. Additionally, they developed an algorithm that works online and can directly remove skew from data. Khlifi and Grégoire also argue that Paxson's algorithm would perform poorly with highly variable data according to Moon et al. [54], which is, however, not a result of the work of Moon et al..

Gurewitz et al. [56] present optimization algorithms to solve the problem if there are no synchronized clocks at all and path characteristics are asymmetric. The aim is to estimate the OWD values (not only the skew) between all nodes. The proposed algorithm is evaluated based on simulation. However, it is questionable if such scenarios and measurement setups with completely unsynchronized nodes are feasible for network monitoring. While the paper gives an example for implementation using features of routers from Cisco Systems, it also states that the focus is on overlay networks.

As a conclusion from the presented work, it can be stated that despite many ideas to improve Paxson's algorithm [53], Paxson's algorithm or the other *simple* algorithms developed by Khlifi and Grégoire [49] are sufficient for skew estimation in most scenarios. Additionally, measurement data presented in the publications shows that drift is in general negligible.

2.4 Summary

This chapter presented the fundamentals of IP-based networks, metrology, and delay measurement. Due to the characteristics of packet based networks, network delays depend on network load, and packet handling in routers. In both network scenarios that have been highlighted (ISP and enterprise), service delivery depends on the transport services offered by different network providers. Therefore, measuring network performance, especially delay, is vital for performance diagnostics.

The metrology section showed how errors can be compensated and/or quantified, which are fundamental concepts for delay measurement. OWD can be measured by active approaches that inject additional probe traffic into the network or passive measurement approaches that make use of extracting characteristics from production traffic. While there are commercial products available that implement certain active or passive delay measurement mechanisms, there is current research that addresses the two main challenges: obtaining better results and reducing the effort. Furthermore, clock synchronization is still an issue despite the high accuracy that can be provided by GPS clocks today and several methods for compensating skew are available.

In the following chapters the fundamentals of this chapter will be applied to develop and study a flow capturing based approach for passive delay measurement. Especially the topics router architecture, network topology, metrology concepts, and delay measurement concepts are important. Moreover, it is crucial to consider the characteristics of flow capturing itself, which are highlighted in the next chapter.

3 Flow Capturing

This chapter deals with flow capturing, which is an approach to retrieve information about traffic flows for monitoring purposes. The first section introduces the basic terminology used within this thesis. Afterwards, the second section presents different flow capturing approaches currently in use and/or in standardization as well as current research directions. The third section details implementation of flow capturing in routers and probes, while the fourth section presents flow capturing applications and deployment. In the fifth section the processing of flow data is addressed. The last section provides a summary of the chapter.

3.1 Terminology

There are several definitions of the term *flow*. Therefore, a consistent definition of a flow and related terms will be introduced. A general flow definition serves as starting point from which more specialized flow definitions, which are relevant in the context of this thesis, will be defined in the first subsection. The second subsection compares flow terminologies in literature and the last subsection introduces the terminology of flow capturing mechanisms.

3.1.1 Flow Definitions

First, the *general flow* will be introduced. A *flow* $\mathbf{F}$ describes a *set of packets* which are observed in a certain *time interval* $[t_s, t_e]$. All packets belonging to a flow share a common *key* $\mathbf{K}$, which is generated from the *packet properties* $\mathbf{U}$. Consequently, there are two degrees of freedom for a flow: the length of the time interval and the creation of the key from the flow properties. Hence, a flow is defined as $\mathbf{F} = (\mathbf{K}, t_s, t_e)$. This definition is similar to the terminology defined by the IETF *IP Flow Information Export* (IPFIX) Working Group [RFC 3917].

Flow Key

Figure 3.1 shows from top to bottom the general and three more specific flow definitions, which are most often used when dealing with flows. The top case in the figure shows all possible properties that can be taken into account for $\mathbf{U}$. $\mathbf{U}$ contains *packet data*, i.e., all header fields, and payload properties. The latter holds in cases where flow classification is based on *Deep Packet Inspection* (DPI), i.e., classifications beyond protocol headers. Furthermore, the $\mathbf{U}$ contains *network metadata*, e.g., the interface number at which the packet was received at a router,

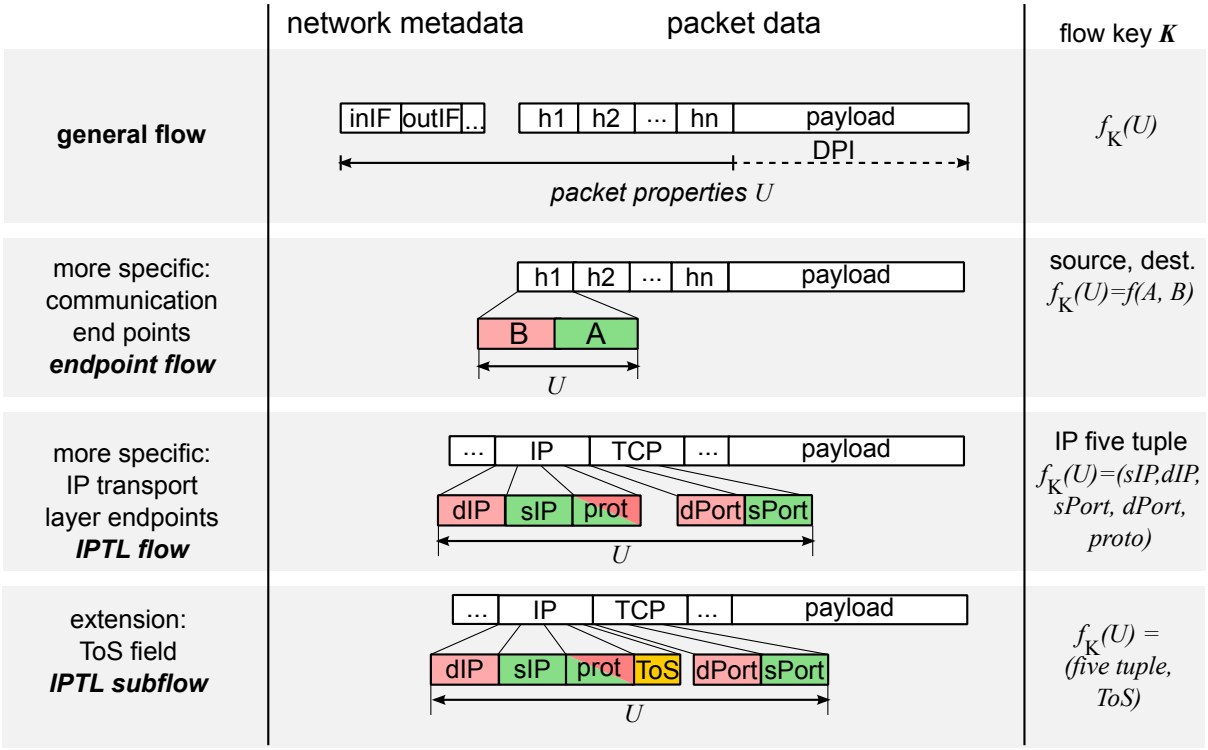


Figure 3.1: Flow definitions

or information from routing. In order to obtain a specific flow definition, certain properties from **U** are selected and transformed by the function f_K in order to obtain the flow key **K**. The combination of these two operations (selection and transformation) allows the creation of specific flow definitions for different applications. Often properties are directly put into the flow key without transformation by f_K. However, applying transformations in f_K is useful for considering aggregated traffic flows, i.e., calculating subnet prefixes from IP addresses or aggregating on interface groups.

Derived from the general flow, a useful and often used flow definition is the *endpoint defined flow*, which is shown in Figure 3.1 in the second box from top. Here, only source and destination addresses are taken into the flow key ($\mathbf{K} = f(A,B)$). The selection of the layer from which the addresses are taken depends on the scenario and the technology used (Ethernet, IP, ...). An endpoint defined flow describes a unidirectional data transmission from source A to destination B. Source and destination can describe subnets or device groups, such that this flow definition is also valid for considering multicast groups or traffic between subnets.

Starting from the abstract endpoint defined flow, source and destination can be defined more specifically. This leads to flow definitions on different protocol layers (Layer-2-Ethernet-Flow, IP-Flow, UDP-Flow) and different aggregation levels (subnet flows, multicast flows, port range flows). In IP networks it is useful to define source and destination as transport layer endpoints, leading to an *IP Transport Layer Flow* (IPTL Flow). Since the Internet architecture does not use separate transport layer and network layer addresses (ports), the flow key must contain source and destination addresses from the IP and transport layers as well as the transport protocol number (Figure 3.1). Therefore, this five tuple defines the IPTL flow key directly without any

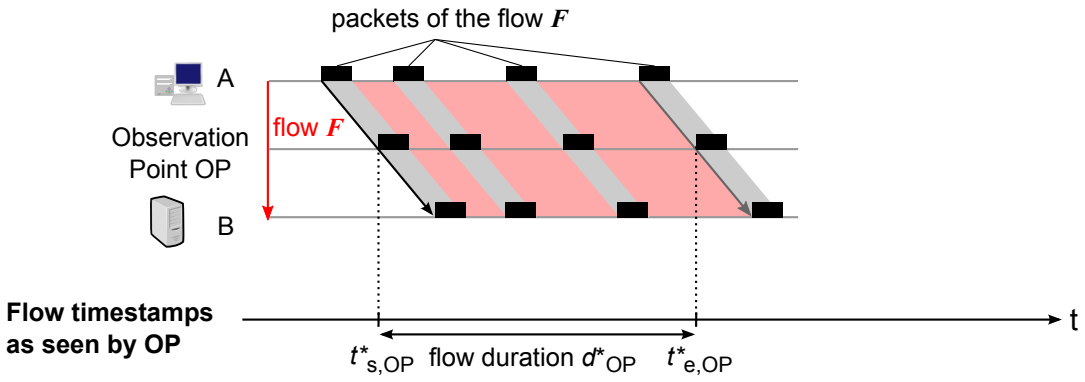


Figure 3.2: Temporal flow definition

transformation: $\mathbf{K} = (sIP, dIP, sPort, dPort, prot)$. In literature, the term *flow* often implies an IPTL flow (see Section 3.1.2).

Due to the fact that queues in network elements and flow capturing mechanisms distinguish flows by more properties than the five tuple, the *IPTL subflow* $\mathbf{F}_s$ is introduced. For an IPTL subflow the IPTL flow key is extended by other fields, such as the ToS value ($\mathbf{K} = (sIP, dIP, sPort, dPort, ToS, ...)$). An IPTL flow can therefore consist of several subflows, altogether belonging to the same data transmission between two transport layer endpoints.

Flow Duration

As introduced above, the flow definition not only covers the *flow key*, but also the *time interval* during which packets are associated with a flow. The IPFIX definition [RFC 3917] does not cover this and allows for an arbitrary time interval.

In this thesis the term flow duration is defined for endpoint defined flows as being the *duration of the data transmission*. This is illustrated in Figure 3.2 for a flow from endpoint A to endpoint B, where flow start and end times correspond to the times when the first, respectively last packet of a data transmission are observed. Due to network delays, these timestamps depend on the *Observation Point* (OP) and therefore are only valid in an OP context.

The actual data transmission, i.e., the flow, lasts from the first bit of the first packet to the last bit of the last packet. In most systems, however, packet start and end times are not available separately, but only a single timestamp value per packet is given (e.g., “packet reception time”, “packet switching time”). In order to be consistent with this implementation characteristic and in order to simplify the notation in figures, this thesis defines the flow start time t_s^* as the time when the first bit of the first packet is observed and the flow end time t_e^* as the time when the first bit of the last packet is observed (see also Figure 3.2). d^* is the flow duration measured between t_s^* and t_e^*.

Using the definition of a flow that is related to the duration of the data transmission leads to the problem that in the general case flow end times (and even flow start times) cannot be de-

termined without additional knowledge. Especially with connectionless communication, there is no possibility to detect this by just looking at packets. How flow capturing mechanisms deal with this problem is covered in Section 3.2.

Bidirectional Flows: Forward and Reverse Flow

While not inherently necessary, packet based communication is predominantly bidirectional. In case of connection oriented transport protocols, this results already from handshakes at connection setup. However, even with connectionless transport protocols, there is often bidirectional communication due to the request-response principle used in applications.

With bidirectional communication, there is the possibility to create bidirectional flows from two endpoint defined flows. The two flows of a bidirectional flow are called forward flow $\mathbf{F}_{\mathrm{fw}}$ and reverse flow $\mathbf{F}_{\mathrm{rv}}$. The forward flow initiates communication. Thus, the reverse flow has switched endpoint addresses and starts after the forward flow: $(A_{\mathrm{rv}} = B_{\mathrm{fw}}) \wedge (B_{\mathrm{rv}} = A_{\mathrm{fw}}) \wedge (t_{\mathrm{s,fw}} < t_{\mathrm{s,rv}})$. This definition corresponds to the IPFIX definition "Direction by Initiator" of the RFC on bidirectional flows [RFC 5103].

3.1.2 Flow Terminology in Literature

One of the first flow terms was defined in 1986 by Jain et al. [57] in their model of "packet trains". This model calls a bidirectional flow a "packet train", which itself consists of unidirectional flows ("tandem trailers"). All packets transmitted on a "railroad track" between two network nodes are aggregated into a packet train. Hence, this definition is a bidirectional flow, defined by endpoints (network nodes). The flow (train) duration is determined by the parameter "maximum allowed intercar gap" and is therefore independent of the duration of the complete data transmission.

IETF documents mention specific flow definitions for the first time in the context of QoS mechanisms in the 1990s. The *Integrated Services* (IntServ) Architecture [RFC 1633] mentions a flow as "stream of related datagrams that results from a single user activity and requires the same QoS". Formally, the term *flow* is defined along with the IntServ *Resource Reservation Protocol* (RSVP) [RFC 2205]. There, a flow is described as RSVP session and an associated "filter spec". This corresponds in terms of UDP and TCP to the definition of a unidirectional flow, defined by the five tuple. IntServ assigns QoS-Parameters (called "flow specification" [RFC 1363]) to such a flow. Flow duration is tied to an RSVP session.

Claffy et al. [58] examined methods for obtaining flow-based information from Internet traffic in 1995. They used unidirectional endpoint defined flows and compared different aggregation concepts for endpoints (transport layer addresses, network layer addresses, address ranges). In addition to a flow defined by two endpoints, they also used "single endpoint flows", which aggregate all traffic originating from or sent towards an endpoint into one flow. Furthermore, the study evaluated the impact of different timer mechanisms on the amount of flow data caused.

At the end of the 1990s the IETF working group *Real Time Flow Measurement* (RTFM) specified an architecture for the measurement of flow characteristics [RFC 2721, RFC 2722]. Here,

a flow is defined as a “portion of traffic” that is defined by start and end timestamp and belongs to a “traffic group”. The latter corresponds to the flow key, which can be created from interface numbers, sub-IP-layer addresses, IP addresses, or transport layer addresses. Since this definition covers network metadata (interface numbers), it goes beyond the endpoint defined flow, but does not cover the general flow definition. Flow duration is called “lifetime” and corresponds to the duration of the data transmission as it is measured from the first to the last packet at the observation point.

In IPv6 the header format was redesigned and a *flow label* of 20 bit size was added [RFC 3697]. The standard defines a flow as “a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow.” Thus, a flow is defined by the triple of source address, destination address and flow label. A correspondence of such a flow to an IPTL flow is suggested, but not demanded. In contrast to a five tuple defined flow, the flow label allows flow-based classification in routers even if the five tuple is not available due to missing port numbers in case of fragmentation or encryption. However, the flow label is currently hardly used and subject to discussion on changes in the standard [59].

The most general flow definition is given in the already mentioned IPFIX RFC [RFC 3917]. Here, a flow is defined as “set of IP packets passing an observation point in the network during a certain time interval”. The packets of a flow “have a set of common properties”, which are defined by “applying a function” to values obtained from packet headers, packet characteristics or from “packet treatment”. This flow definition is clearly limited to IP packets. Meanwhile, however, this was noticed as unfeasible by the IPFIX WG and there are discussions on whether to use the more general term “packets” only [60]. The flow definition of IPFIX is also used by other standardization organizations. E.g., an ITU-T recommendation from 2007 [Y.1543] is referencing the IPFIX requirements RFC [RFC 3917].

After having a look at the different flow definitions, it can be seen that flow terminologies evolved and are still subject to change. The general flow definition used in this thesis is close to the currently discussed changes to IPFIX. However, it covers network metadata in general and is not limited to data from ‘packet treatment”. Furthermore, almost all definitions in literature use endpoint defined flows or even an IPTL flow definition, since this contains the details required for most applications that are based on flow data.

3.1.3 Terminology of Flow Capturing Mechanisms

Flow capturing describes the complete process of the retrieval of flow information (*metering*), *collection* of this information, and delivery of flow information to *applications*, which analyze the data to get information for different purposes. Consequently, the architecture is divided into three components: *flow meter*, *flow data collector* and *application*, as shown in Figure 3.3.

The meter observes traffic at an *Observation Point* (OP) and creates information about traffic flows. Meters can be realized as standalone devices (probes) or integrated into other network elements, such as routers. Flow data is then transferred from the meter to the collector, which stores flow data and forwards it to the applications. Typically, several meters are deployed at remote sites in networks and report to a single collector. These general terms are adopted from [RFC 5470] and [RFC 1272].

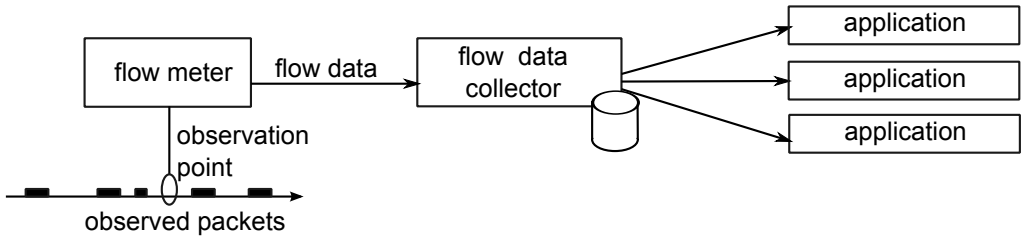


Figure 3.3: Flow capturing components

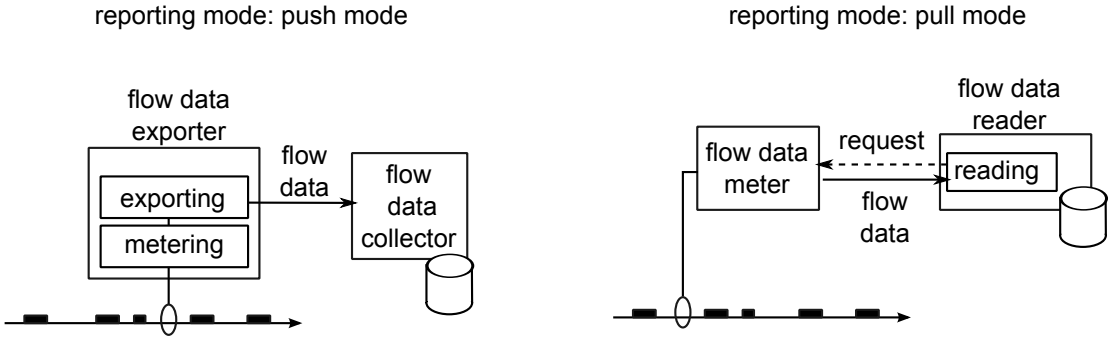


Figure 3.4: Reporting modes for flow capturing

In terms of implementation of this general flow capturing architecture, there are several degrees of freedom. Most importantly, these are the *reporting mode*, *flow state keeping*, and *sampling*. The reporting mode defines which entity is the active part for transferring flow data between meter and collector. Here, *push mode* and *pull mode* can be distinguished, as shown in Figure 3.4. In push mode, the meter component is called *flow data exporter* and contains a metering process and an exporting process. The latter is the active part that pushes flow data to the collector. In pull mode, the collector component is called *flow data reader* and contains a reading process that requests flow data from the meter.

Considering flow state keeping, the meter can either keep flow state or not. As shown in Figure 3.5 a meter that keeps flow state (*stateful metering*) uses a flow cache that contains all currently observed flows. The meter aggregates certain flow attributes (e.g., packet count) by updating counters per flow key **K** based on packets observed. These aggregated flow attributes are denoted as **V**. In general, it can contain every attribute of the packet property set **U** and does typically not contain attributes that are taken into account for the flow key.

If the reader does not keep flow state (*stateless metering*), it does not need to have a flow cache and creates flow data that is directly sent to the exporter. Since sending flow data for every packet to the collector results in a high amount of monitoring data, only a fraction of packets is selected for monitoring by a sampling process as shown in Figure 3.5. Therefore, stateless flow capturing is also called *packet sampling*. Sampling can also be used with stateful metering for reducing the flow data rate and is typically used if meter, exporter, or collector performance does not allow for unsampled stateful flow capturing.

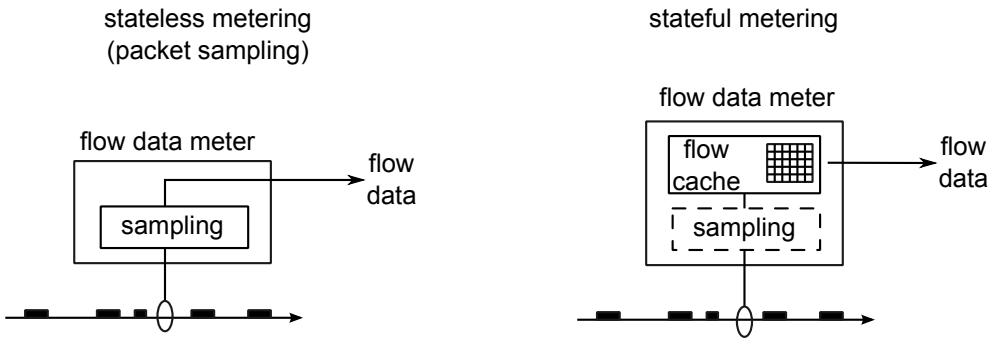


Figure 3.5: Flow state keeping

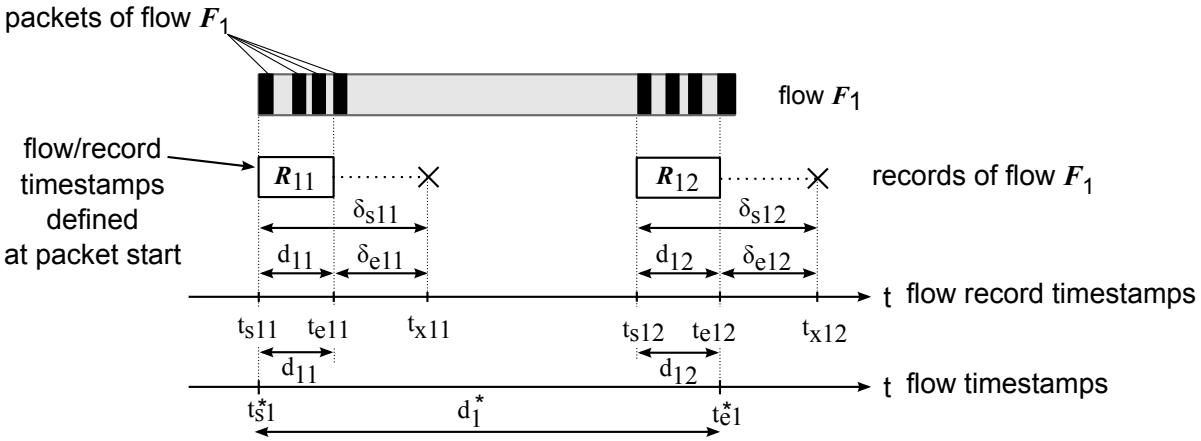


Figure 3.6: Flow record terminology

Flow data created by meters consists of *flow records* that contain information about a flow of a certain time interval. For stateful metering, information on the flow is aggregated in the flow cache before it is transferred in a flow record. Thus, flow records can cover large time intervals of the flow lifetime. If sampling is applied, flow records contain only information on a fraction of packets of the flow. In case of stateless metering the time interval of flow records covers only a single packet of the flow.

Figure 3.6 shows the terminology of flow records used in this thesis. For the flow $\mathbf{F}_1$ there are two flow records $\mathbf{R}_{11}$ and $\mathbf{R}_{12}$ created by flow data meters. The flow records contain the aggregated attributes $\mathbf{V}$ that describe properties of the flow for the time interval considered, such as the packet count p and the byte count b. Timestamps of flow start and flow end are defined to correspond to packet start times, i.e., the time at which the first bit of a packet is seen. Flow start and end times are denoted by $t^*_{s\,1}$ and $t^*_{e\,1}$, while the records start times are t_{s11} and t_{s12}, and the record end times t_{e11} and t_{e12}, respectively. Flow and record duration are defined as the time interval between start and end, denoted as d^* and d, respectively. The time when the flow records are exported from the flow capturing device (either pushed or pulled) is termed *export time*, indicated by an X-symbol in figures (see Figure 3.6) and denoted by t_x. Since records are typically exported some time after t_e, this results in an *export delay*. The export delay measured from record start is denoted as δ_s and the export delay measured from record

end as δ_e. Shortly after export, the flow data packets arrive at the collector. The network delay on this path is negligible compared to the export delay.

As shown flow metering and transferring flow records are in principle different concerns. However, requirements for the protocol used for transferring flow records strongly depend on the flow metering and vice versa. Thus, the protocols that have been developed are linked directly to the capturing mechanisms they are intended for, as it will be shown in the next section.

3.2 Flow Capturing Mechanisms and Protocols

There are several standards for flow capturing today and there is high activity in product development and research. In order to understand the differences of and dependencies between different flow capturing approaches, first the history of the most important flow capturing developments will be presented. Afterwards, each of the approaches will be introduced and an overview on current research in this area will be given. At the end, a comparison of the different approaches shows their differences.

3.2.1 Historical Development

The historical time line is illustrated in Figure 3.7. The first research on flow capturing was done in 1986 by Jain et al. [57], who introduced the "packet train" model. The first IETF activity in this field was the IETF Accounting (IETF ACCT) Working Group (WG) [61], which was established in 1990. This WG released one RFC on "Internet Accounting: Background" [RFC 1272], which defines meter services and usage reporting. The meter service "records aggregate counts of packets belonging to FLOWs between communication entities", i.e., it is a flow capturing device. An Internet Draft describing an architecture was published in 1992, however, no further RFCs were released and the WG was concluded in 1993.

The concepts of the ACCT WG have been realized in a flow capturing tool called NeTraMet (Network Traffic Meter) that was released in 1993 and documented in [RFC 2123]. This tool and the ideas of the ACCT WG were the basis for the IETF Real Time Flow Measurement (RTFM) WG [62] that started in 1996 and concluded in 2000. This WG released a series of RFCs in 1999 that specify the architecture and its components. RTFM specifies a stateful flow capturing approach where flow data is pulled from the reader using the *Simple Network Management Protocol* (SNMP)[RFC 3410]. Details on RTFM are given in Section 3.2.2.

Stateless flow capturing was introduced in 1993 by Hewlett-Packard as Extended RMON (XRMON)[1]. The company InMon developed the *sFlow* standard that contains a protocol for transferring flow records from packet sampling [RFC 3176]. sFlow was used by an increasing number of switch manufacturers who joined the *sFlow forum* [63]. The sFlow forum extended the sFlow protocol and published sFlow version 5 in 2004 [64]. sFlow is described in Section 3.2.3.

[1] Remote Network Monitoring (RMON) [RFC 2819] is a network monitoring protocol mainly for traffic monitoring based on SNMP.

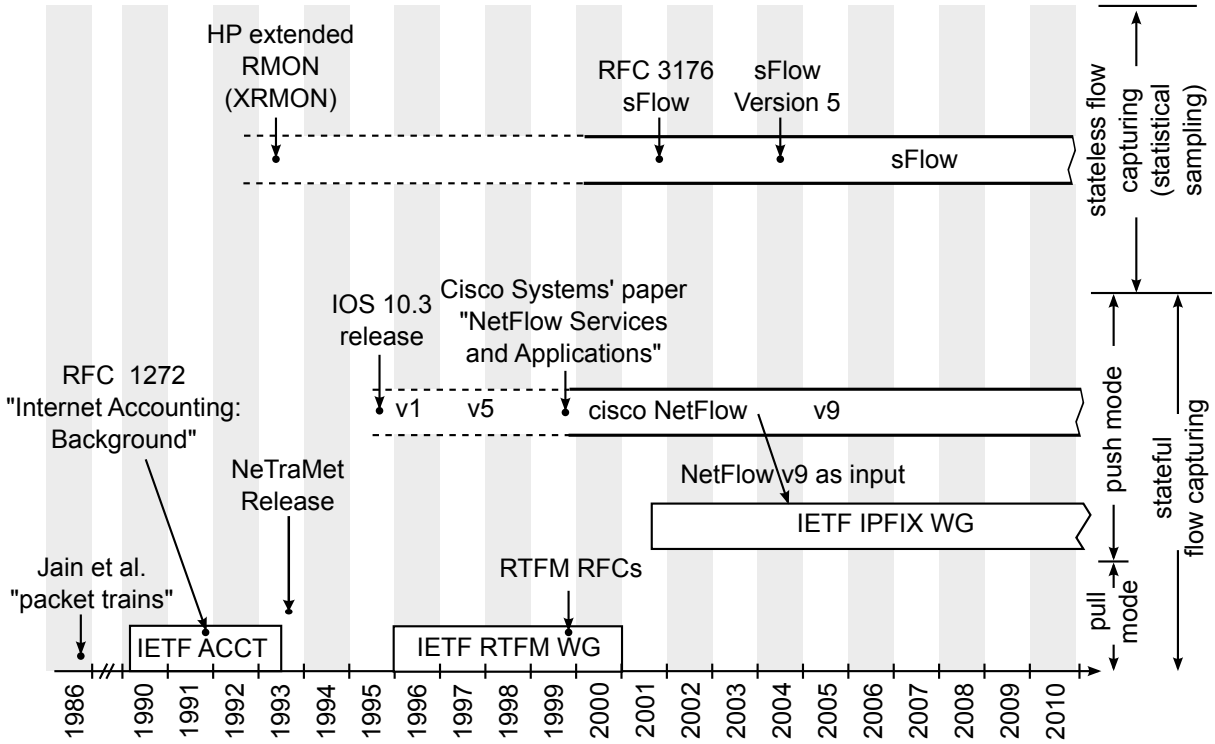


Figure 3.7: Historical development of Flow Capturing

Cisco Systems introduced a rudimentary flow capturing mechanism in its routers with the operating system release IOS version 10.3 [65] around 1995. The flow cache itself was introduced for performance improvements when handling a high number of routes and forwarding rules. This technology was called “NetFlow Switching”. By exporting the flow cache content, the flow capturing mechanism “NetFlow Export” was established, later only called “NetFlow”. After some early versions, the currently still widely used NetFlow version 5 was developed. This version underwent further changes throughout IOS releases until 1998 [66]. In 1999 Cisco Systems published the white paper “NetFlow Services and Applications”[67], which gives an overview of NetFlow versions and implementations. This white paper is a major reference cited in the scientific community when it comes to NetFlow. The next major step was the development of NetFlow version 9 with a flexible flow record format. This is the most recent version available in components currently deployed. More details on NetFlow will be presented in Section 3.2.3.

In 2001 also other ideas on flow capturing mechanisms emerged in addition to the three approaches presented here. The IETF realized that this situation makes development of generalized analysis applications difficult. Therefore, the IP Flow Information Export (IPFIX) working group started in 2001 in order to specify a common terminology and protocol. After defining the requirements and comparing existing protocols, the working group decided to take NetFlow v9 as input and to develop an IPFIX protocol based on it. The IPFIX WG is very active and has produced a high number of RFCs. IPFIX and its current stage are presented in Section 3.2.5.

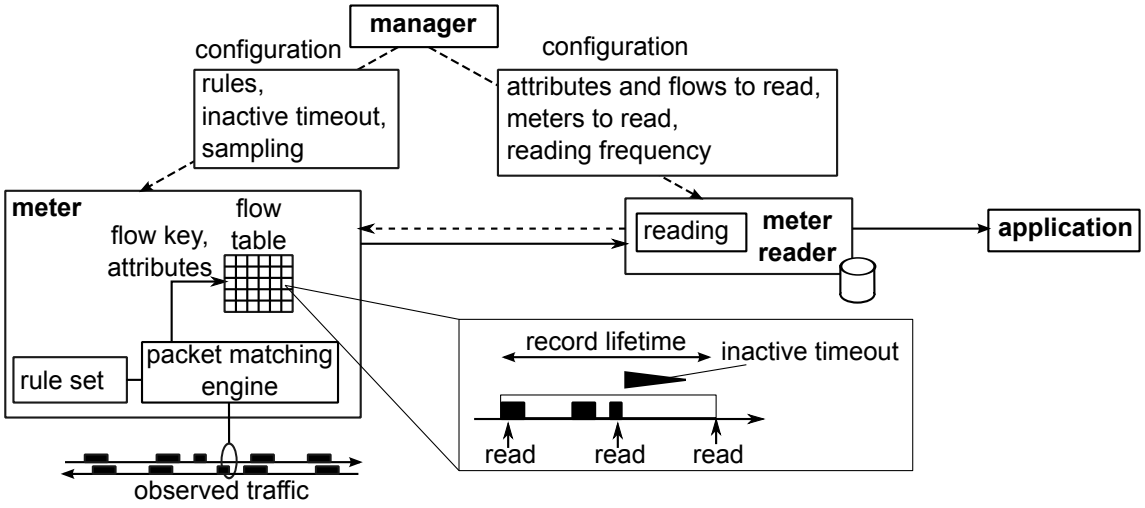


Figure 3.8: RTFM Architecture

3.2.2 Real Time Flow Measurement (RTFM)

The IETF RTFM WG specified an architecture [RFC 2722] (see Figure 3.8) consisting of *meter*, *meter reader*, *manager*, and *application*. The meter creates flow records by keeping flow state in a flow table. For each packet it observes, its packet matching engine creates a flow key and determines attributes that are measured based on a rule set. This means that the flow definitions depend on the rules and are flexible so that depending on the network scenario the *metered traffic groups* can be chosen.

One or more meter readers pull flow records from the meter in regular intervals. Which flow records and which attributes are read is described in the request that is done via SNMP with a specified Meter *Management Information Base* (MIB) [RFC 2720]. A meter reader can read data from several meters and a meter can be read from several meter readers. The meter reader forwards flow records to the traffic analysis application.

The problem of determining the flow lifetime is solved by applying an inactive timeout in the meter (see diagram in Figure 3.8). If no more packets are observed for a certain flow for a certain time, the meter can remove the flow record from the flow cache, but it additionally checks that all meter readers read the final flow record before it is removed. The flow attributes that are counted or aggregated are not reset when the data is read, but the counters for a flow record are always increasing.

Meters and meter readers are configured by the manager. The manager loads rules, the timeout value, and possibly sampling parameters into the meter. Additionally, the manager also configures the meter reader with information on which meter to read at which frequency.

RTFM meters typically capture bidirectional flows. If a rule on a packet does not match in the first run, the endpoint addresses are switched and the rule is applied again. Based on whether a match occurred in the first or second run, forward and reverse flow can be distinguished. [RFC 2723] specifies a Simple Ruleset Language (SRL) for defining rules that can then be compiled into the format that a manager loads into the meter.

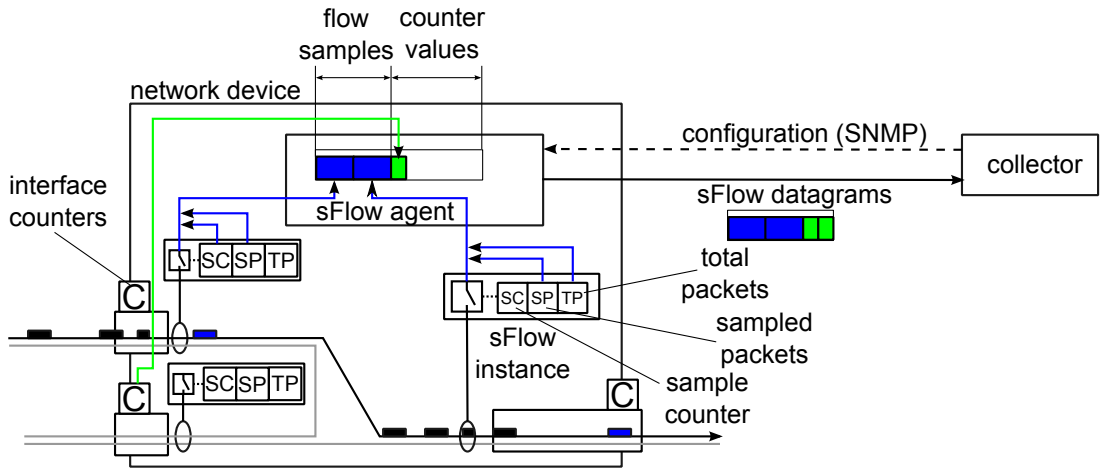


Figure 3.9: sFLow statistical packet sampling

The RTFM architecture was implemented in the NeTraMet tool, which can run on UNIX or DOS computers. Other implementations are not known, especially there is no implementation suitable for high-speed routers that run at several Gigabit/s.

3.2.3 InMon Corporation's sFlow

sFlow is a flow capturing mechanism that works stateless based on packet sampling. It is explicitly designed as low cost mechanism that can be implemented with low effort in hardware (e.g., in Ethernet switches). The sFlow specifications [RFC 3176, 64] provide a definition of the sampling mechanism (including an SNMP MIB for management) and the protocol for transferring the data via *sFlow datagrams* in push mode to the collector. As additional feature that is not related to flow capturing, values of interface counters are piggybacked on sFlow datagrams.

Figure 3.9 shows the components of the sFlow architecture. A network device (switch or router) runs an sFlow agent that is configured from the collector and sends captured data using sFlow datagrams to it. Interfaces in the network device are associated with *sFlow instances* that sample the traffic passing by. Depending on implementation and configuration sFlow instances observe ingress or egress traffic at an interface, or both directions. Each sFlow instance keeps a sample counter that is initialized with a number corresponding to the sample rate and decremented for each packet observed. If the counter reaches zero, the packet information is put into a flow sample and the counter is reset to the initial value. Additional counters of an sFlow instance count the whole number of packets observed and the number of packets sampled.

When a flow sample is created in an sFlow instance, the information in the packet header together with the number of packets sampled, the number of total packets, and the sampling rate are added to the flow sample and sent to the sFlow agent. Figure 3.9 shows that the sFlow agent compiles an sFlow datagram containing several flow samples. sFlow does not only report flow data, but the agent also adds counter samples from interface counters to the sFlow datagram. These counter values can be added to the flow data in order to fill the datagram if there is space

available. The sFlow specification defines how often counter values should be read and which degree of freedom the agent has to add them to the datagram. The sFlow agent can wait for several samples to fill a datagram but must not delay samples by more than one second.

Due to the counter-based sampling approach, no flow state has to be kept by sFlow enabled devices. The flow sample includes addresses of the IP-Layer, the transport layer and/or the sub-IP layer, depending on the sFlow format used. Since there is no flow cache, there is no definition of a flow or flow key. It is therefore up to the collector to handle these samples and aggregate them to the required flow definitions.

sFlow is often deployed in data centers and at peering points like the AMS-IX [68] and DE-CIX Internet exchange. In such scenarios mainly Ethernet switches are deployed, which typically do not have a flow cache and thus cannot implement stateful flow capturing.

3.2.4 Cisco Systems NetFlow

The *NetFlow services export* feature and protocol were introduced to export the content of the flow cache, which has been used in some routers of Cisco Systems for flow-based switching. NetFlow is a highly efficient and implementation friendly protocol where the flow capturing mechanisms and the protocol are closely related. It transfers flow records from an exporter to a collector in push-mode. This section presents the details of NetFlow version 5 (NetFlow v5), which is the version most widely in use today, and NetFlow version 9 (NetFlow v9). Versions 2 to 4 and 6 were never published, while Version 7 and 8 mainly add aggregation features and are not be covered here. These NetFlow versions are described in [67]. Although NetFlow is Cisco proprietary, it became a de facto standard for flow capturing due to its wide deployment. However, due to Cisco's rights on NetFlow, other vendors call it differently (e.g. cflow, jflow).

In the following, the timer-based flow cache management and the NetFlow protocol are introduced. Since the OWD measurement approach proposed in this thesis is based on NetFlow, details and symbols are presented, which are used throughout the following chapters.

NetFlow keeps flow state in the flow cache and updates flow records for the packets observed for these flows. The metering process decides when records expire and removes them from the flow cache at cache removal time t_r for sending them to the collector at t_x.There are several expiration criteria that determine t_r. For each criterion, different t_r formulas based on timeout parameters and flow record start/end timestamps t_s/t_e apply. The criteria are given in the following list and depicted in Figure 3.10.

- **Inactive Timeout** ϑ_{inact}
  If no packet is seen for the considered flow record for a certain time (i.e., the flow is considered inactive), the record is expired: $t_{r,inact} = t_e + \vartheta_{inact}$

- **Active Timeout** ϑ_{act}
  If the flow record contains flow data about more than a given interval (i.e., the flow is considered active for more than a given time), the record is expired: $t_{r,act} = t_s + \vartheta_{act}$. This ensures that data about flows with long duration is exported early, i.e., flow data is not only available when the flow terminates.

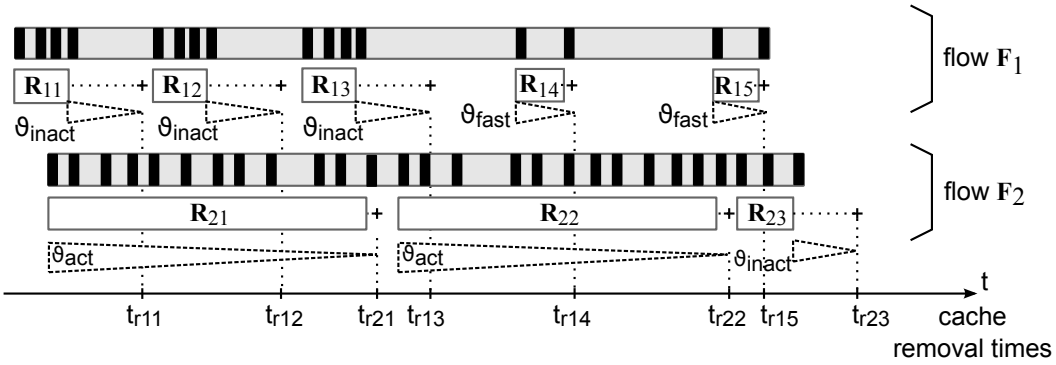


Figure 3.10: NetFlow timeouts

- **Fast Timeout ϑ_{fast} and p_{fast}**
  If only a small number of packets p is observed for a flow record in a given time interval, the flow is expired: $t_{\text{r,fast}} = t_{\text{e}} + \vartheta_{\text{fast}}$ if $p < p_{\text{fast}}$. The purpose of this mechanism is to free the cache from flow records that only contain a single packet (e.g., DNS requests/responses or failed TCP connection setups).

- **TCP connection termination**
  Some NetFlow meters observe TCP flags and expire flow records when a FIN or RST flag is observed in a flow. In this case the flow is expired when it ends: $t_{\text{r,tcpterm}} = t_{\text{e}}$.

- **Cache Full Events**
  If the flow cache runs full, NetFlow expires flow records earlier than given by the other criteria at $t_{\text{r,cache}}$.

Which criteria can be applied depends on device capabilities and configuration. The criterion that matches first will expire the flow record, i.e.,

$$t_{\text{r}} = min(t_{\text{r,inact}}, t_{\text{r,act}}, t_{\text{r,fast}}, t_{\text{r,tcpterm}}, t_{\text{r,cache}}). \tag{3.1}$$

Figure 3.10 shows an example with the three timeout mechanisms. For both flows the packets, the resulting records, and the cache removal time t_{r} at which the records are removed from the cache are shown. The latter is indicated by a plus-sign in the figure in contrast to the X-sign that indicates record export, which has been used in Figure 3.6. Flow $\mathbf{F}_1$ has several breaks that are longer than the inactive timeout, which results in the three records $\mathbf{R}_{11}$, $\mathbf{R}_{12}$, and $\mathbf{R}_{13}$ expired by this criterion. Then, $\mathbf{F}_1$ contains two bursts of two packets each with sufficient time in between, such that the fast timeout criteria matches and the flow records $\mathbf{R}_{14}$ and $\mathbf{R}_{15}$ result. In $\mathbf{F}_2$ packets are transferred without large breaks in between, such that the first two records, $\mathbf{R}_{21}$ and $\mathbf{R}_{22}$, expire due to the active timeout. Then the flow terminates leading to a last record $\mathbf{R}_{23}$ that expires due to inactive timeout.

The timeouts are configurable in a certain range[2], as given in Table 3.1. This table also gives the default values as well as typical values from a network operator's router configuration [9].

[2]taken from [69], the values vary between implementations.

parameter	min. value	max. value	default value	typical value
ϑ_{inact}	10 s	600 s	15 s	10 s
ϑ_{act}	1 min	60 min	30 min	5 min
ϑ_{fast}	1 s	128 s	32 s	4 s
p_{fast}	1	128	100	2

Table 3.1: Ranges and default values for NetFlow timeout parameters

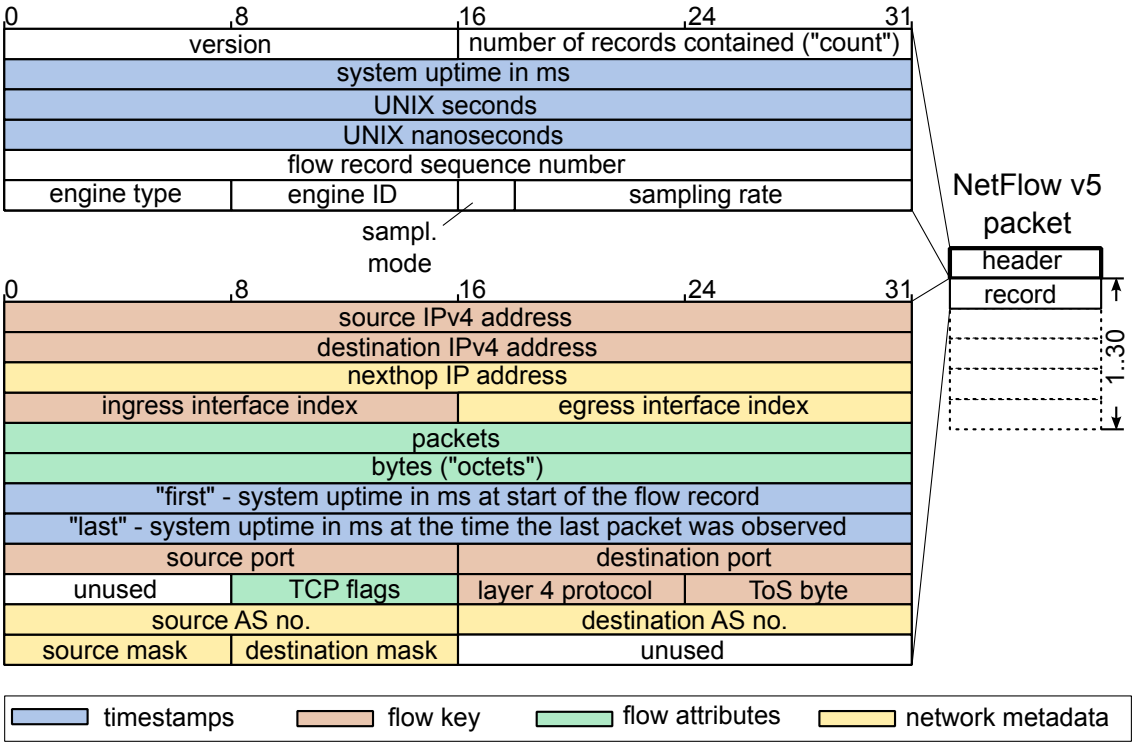


Figure 3.11: NetFlow version 5 packet format (header and record)

NetFlow v5 compiles packets from several expired records and sends them to the configured collectors. Exporters send NetFlow v5 packets via UDP and there is no application layer mechanism for compensating packet loss.

Officially, there is no port number assigned to NetFlow and it is configurable. Typically, UDP port numbers in the range between 9991 and 9999 are used, with port 9995 being most commonly used. This collides with the official port number assignments [70], but is not a problem in practice.

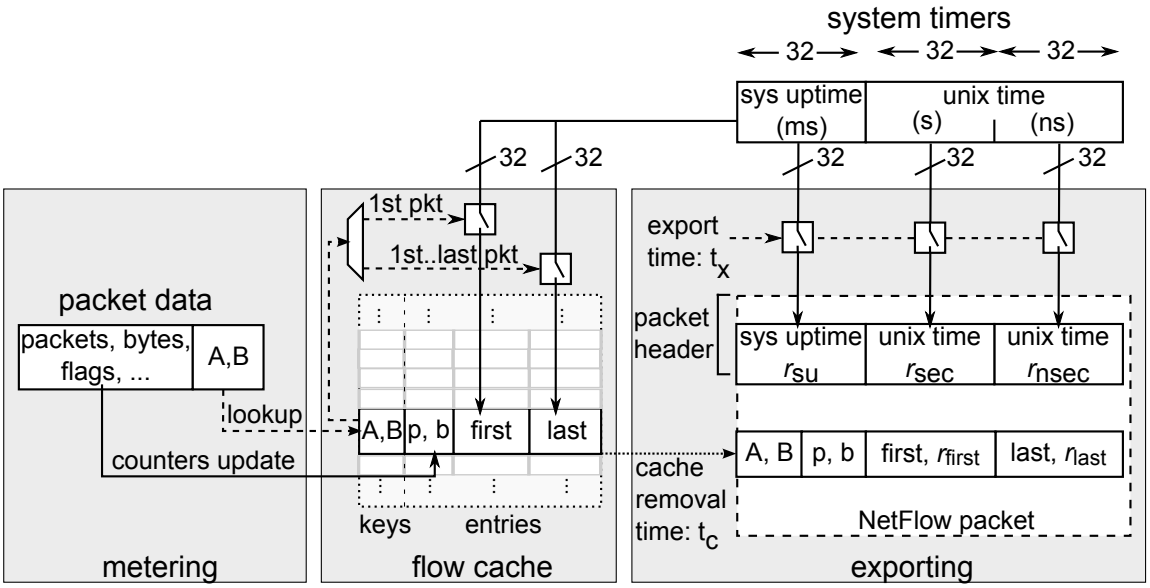


Figure 3.12: Model for NetFlow flow capturing and timestamp creation

The NetFlow v5 packet format[3] is depicted in Figure 3.11. NetFlow v5 packets have a fixed format and consist of a header and up to 30 NetFlow records. The header starts with the NetFlow version number and the amount of records contained in the packet. Then, several timestamps follow that can be used to calculate the system uptime and the time when the packet left the exporter. The record sequence number contains the overall amount of flow records that have been expired from the flow cache, i.e., it can be used at the collector to calculate how many records have been lost due to lost NetFlow packets. The last part of the header contains information about which type of NetFlow engine is used and how sampling is configured.

Fields of the flow records can be distinguished, as shown in Figure 3.11, into fields of the flow key, timestamp fields, flow attributes, and network metadata taken from routing information. NetFlow v5 uses the IPTL five tuple, the ToS byte, and the ingress interface number as flow key. Since the five tuple and ToS byte are contained, every IPTL subflow can be identified. Furthermore, a unique identifier for the observation point can be created from the ingress interface number and the exporter's IP address (obtained from the IP source address in the UDP packet). Flow attributes in NetFlow v5 are the amount of packets, the amount of bytes, and a bitmask containing a cumulative OR of the TCP flags observed for the time interval of the flow record. Furthermore, there are fields for the network masks of the source and destination network, as well as for AS numbers. t_s and t_e of the record are contained in millisecond granularity (fields *first*, *last*) relative to the system uptime given in the NetFlow header.

How the timestamps are efficiently created using this packet format is shown in Figure 3.12. This figure depicts a model of the NetFlow implementation, which has been reverse engineered based on the NetFlow v5 protocol and system knowledge by the author. The model consists of

[3]The format specified in Cisco's document [71] is partly wrong and inconsistent. The format presented here has been validated with NetFlow packets exported from a router. It is consistent with the format specified on another website [72].

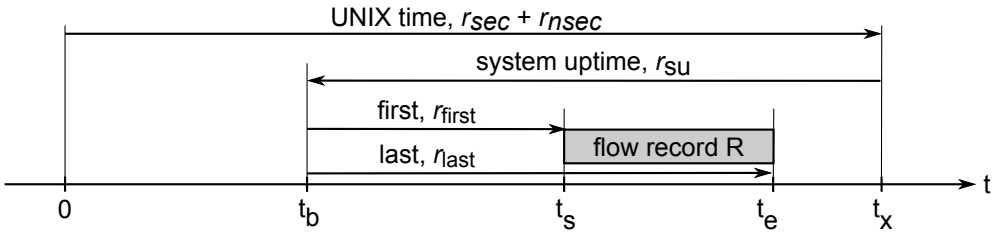


Figure 3.13: NetFlow packet and record timestamp relations

the metering block, the flow cache, and the exporting block. The metering block creates flow keys for packets and updates the flow cache accordingly. The flow cache handles the timers, removes flow records at t_r, and forwards them to the exporting block on the right hand side of Figure 3.12. The exporting block compiles NetFlow packets and sends them to collectors.

The metering part checks for each packet whether there is already a flow record in the flow table. If not, it creates a new flow record and puts the current value of the system uptime counter into the NetFlow fields `first` and `last`. Additionally, it updates the counters of the flow attributes for this record. For every subsequent packet of this flow, the metering part finds the flow record and updates the `last` field with the current system uptime and updates the counters as well. Once a criterion for record expiration matches, the record is removed from the cache without changing any of the values. The exporter process puts several flow records into a NetFlow packet and adds the values of the system counters to the field in the packet header at export time t_x before it sends the packet to the collectors. In the following the `sysUptime` value in the header is denoted as r_{su} , while the UNIX time values are denoted as r_{sec} and r_{nsec}. For the record values `first` and `last` the symbols r_{first} and r_{last} are used.

Timestamps for t_s and t_e are created based on three 32-bit system values that hold the system uptime in milliseconds, the UNIX time in seconds and the nanosecond fraction of the UNIX time. Since t_s and t_e are given relative to the router's uptime, they have to be calculated by taking the timestamp values in the NetFlow header into account, as shown in Figure 3.13. From the UNIX time values that describe t_x, r_{su} is subtracted, which yields the boot time t_b. Adding r_{first} and r_{last} to the boot time yields t_s and t_e. The advantage of this implementation is that the flow cache does not have to store the complete record timestamps in milliseconds of UNIX time, but only a 32-bit value. However, care has to be taken at the collector when the 32-bit counters overflow.

All NetFlow versions up to version 8 have a static record format, i.e., the record field types and their length cannot be changed for exporting more or different information. NetFlow version 9 [RFC 3954] is a complete redesign that provides a flexible format, which can export arbitrary data elements and is extensible towards new data types with different lengths. A well-known approach in protocol design for flexible formats is *Type Length Value* (TLV) encoding where for each protocol field a type number, the field length, and the value is encoded. By specifying a table in the protocol standard that describes the field semantics, new protocol fields can be added by extending this table. Since the length of each field is also encoded, the field's length can be adapted to the scenario's needs (e.g., typical values to expect in counters) and implementations that do not know certain field types can simply jump to the next type. In case of NetFlow, a clas-

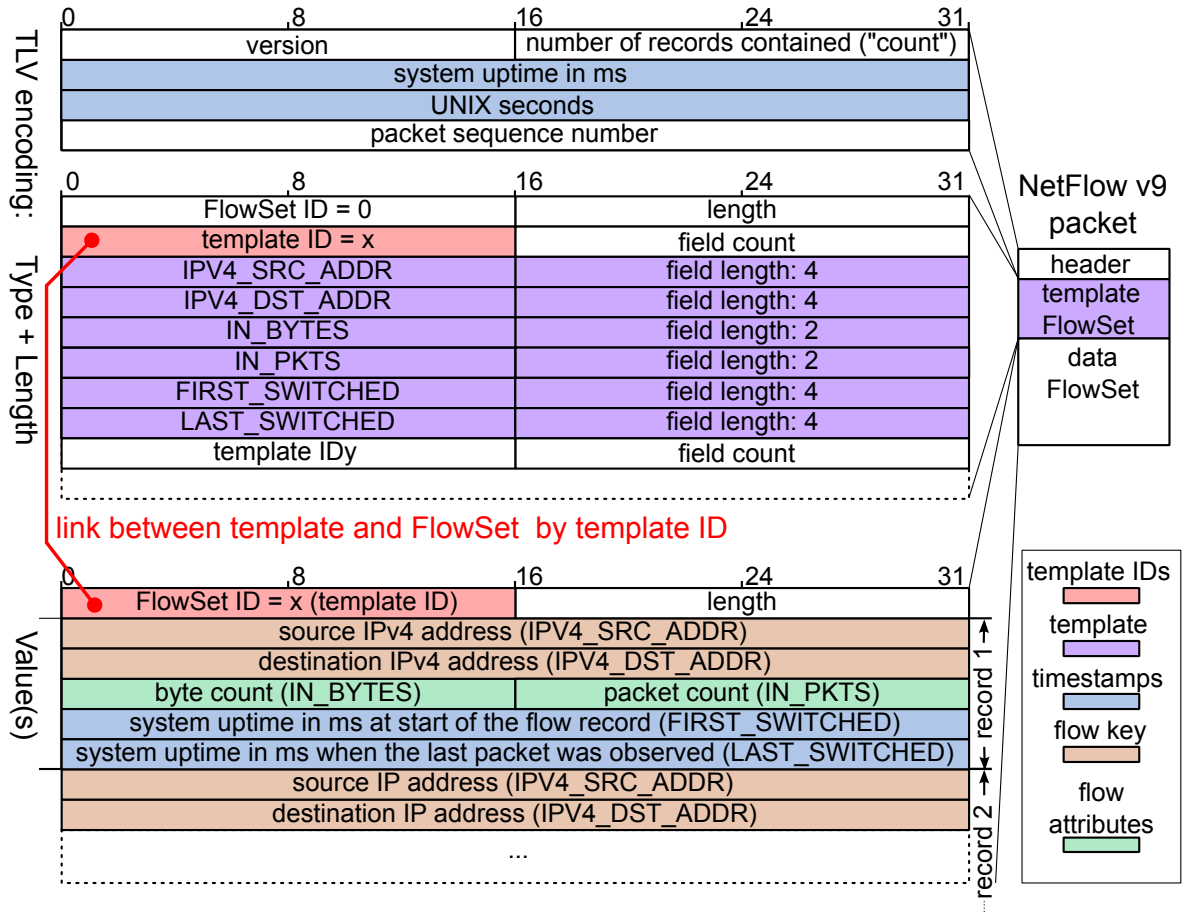


Figure 3.14: NetFlow version 9 packet format (exemplary template shown)

sical TLV encoding would add tremendous amount of redundant type and length information to each record, which is infeasible due to the typically high record rates. NetFlow v9 therefore separates the type and length information from the value by putting the field definitions into *templates* and the values into *data FlowSets* (see Figure 3.14). Additionally, each template contains an ID that links it to the corresponding data FlowSet. When the collector receives a data FlowSet, it can look-up previously received template information for decoding. In the example given in Figure 3.14, the template with ID "x" defines six fields together with their length, which allows the proper decoding of the following data FlowSet with the same ID.

One advantage of the template mechanism is that the collector caches the template and therefore template information has to be sent infrequently only. Additionally, an exporter can send different types of flow records using different template IDs. Furthermore, on reconfiguration of flow capturing for exporting additional fields, new templates containing additional fields are created. This possibility is a precondition for flow capturing implementations that provide a flexible way of defining flow keys and flow attributes (e.g., flexible NetFlow [73]).

Record timestamps in NetFlow v9 are encoded similarly to NetFlow v5: record timestamps are relative to the export timestamps given in the header, i.e., the same calculation rules apply.

However, while NetFlow v5 contains UNIX seconds and nanoseconds in the header, there is no nanoseconds value in the NetFlow v9 header. Hence, the export timestamp is given only with a resolution of one second. Due to the relations of timestamp values shown in Figure 3.13, the flow record timestamps can also only be calculated with a resolution of one second in NetFlow v9. Omitting the nanosecond field is most likely a mistake in the protocol design, since NetFlow v9 implementations seem to obtain the timestamps also as shown in Figure 3.12 while simply dropping the nanosecond value. However, it is possible to compensate this error either in the exporter (adding the millisecond part of missing nanosecond value to r_{su}) or by applying the compensation method described in [74] at the collector. This method is based on the fact that the r_{su} values are monotonically increasing in time and provide millisecond resolution. Some devices also export absolute millisecond timestamps *per record*.

3.2.5 IP Flow Information Export (IPFIX)

IPFIX is an IETF WG that defines standards for flow capturing. While its main work initially targeted towards the export protocol, it meanwhile covers also flow data processing, flow capturing configuration, and sampling. This development is shown in Figure 3.15, which highlights the documents published. After selecting NetFlow v9 as starting point for IPFIX in 2004 [RFC 3955], the base protocol was released in 2008. Then, several extensions and implementation related documents have been released. Additionally, an overview over of the architecture was published in [RFC 5470], which indicates the extension of IPFIX work into the domain of flow data processing. This focus is currently subject to further work in terms of the IPFIX mediation framework [RFC 5982] (see more details in Section 3.5). Besides the work on configuration of flow capturing, IPFIX also works on packet and flow sampling. The latter started independently in the *Packet Sampling* (PSAMP) working group, which was concluded in June 2009 and continues its work in the IPFIX WG.

The IPFIX protocol [RFC 5101] is a major enhancement of NetFlow v9 and is developed as an open standard. IPFIX features the template mechanism of NetFlow v9. However, the data types used in templates are no longer kept in a proprietary table, but the information elements are initially defined in [RFC 5102] and future extensions are to be handled by the *Internet Assigned Numbers Authority* (IANA). In order to allow for custom extensions, IPFIX also defines enterprise specific information elements, which allow for adding protocol extensions in a simple way since they are only valid in a limited scope (enterprise domain).

In terms of transport, IPFIX is specified to work over UDP, TCP and SCTP with partial reliability extensions (PR-SCTP, [RFC 3758]). Compared to UDP, which is used for NetFlow, TCP, and SCTP add reliability and congestion control, while the partial reliability extensions can limit the sender buffer size required for handling retransmissions. The IPFIX protocol uses port number 4739, which has been officially assigned [70].

Regarding the record timestamp encoding, IPFIX provides several formats, which all differ from NetFlow v9. Most importantly, the IPFIX header contains the export timestamp seconds only and no export nanoseconds or system uptime values. Different information elements allow encoding flow record timestamps either as absolute value, relative to the export time, or relative to system uptime. For encoding start/end timestamps as an absolute value there are information elements defined for providing millisecond, microsecond or nanosecond resolution, each using

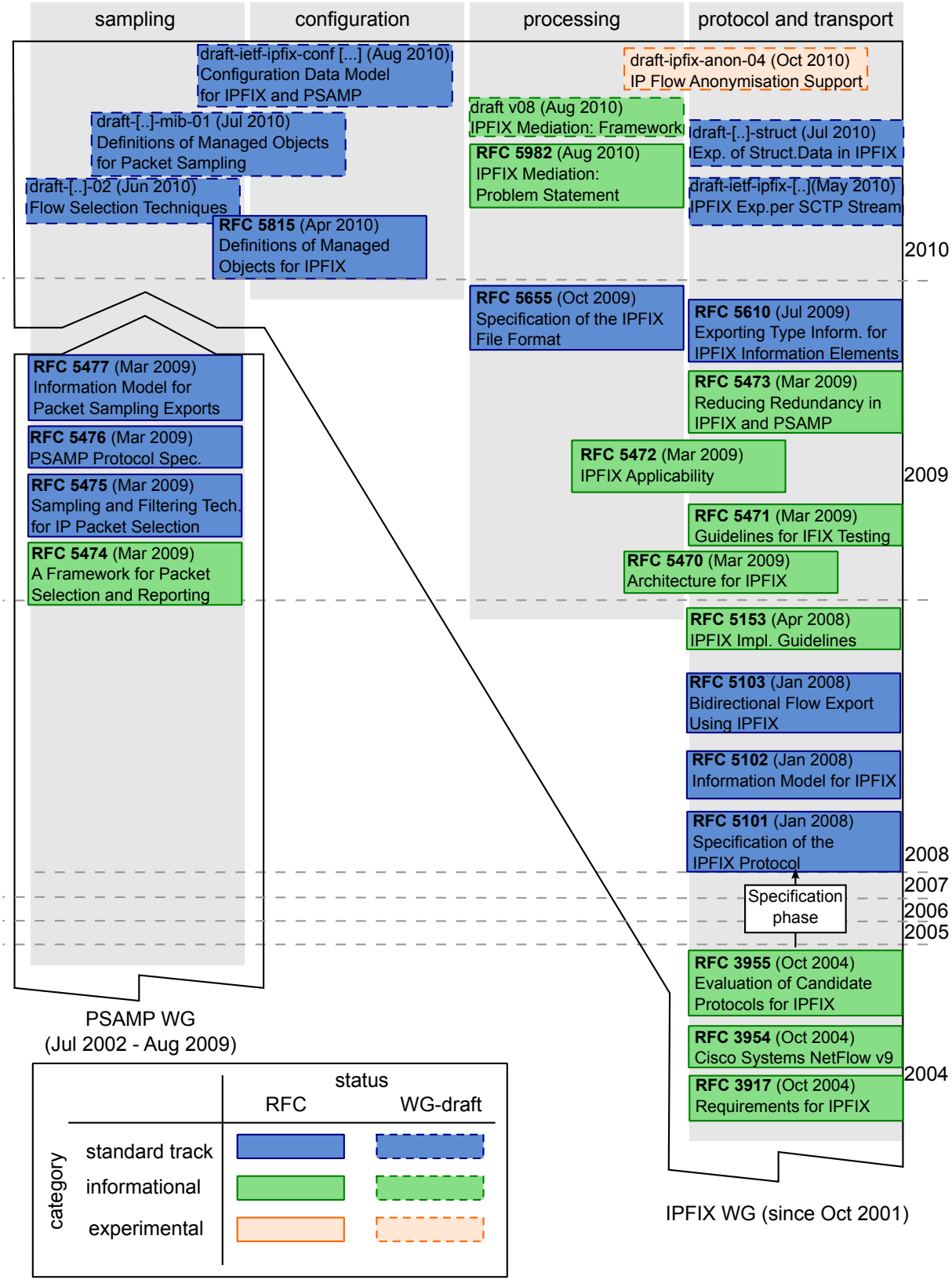


Figure 3.15: IPFIX WG and PSAMP WG documents as of October 2010

64-bit data types. In order to encode the record timestamps relative to the *export time*, there are the `flowStartDeltaMicroseconds` and `flowEndDeltaMicroseconds` information elements, each of 32 bits length. Compared to using absolute timestamps, the latter reduce the amount of data that has to be sent. Encoding timestamps relative to system uptime works by using the `flowStartSysUpTime` and `flowStartEndSysUpTime` information elements. Since the system uptime is not part of the header in IPFIX, it has to be added using the record field `systemInitTimeMilliseconds`. In this way, timestamp encoding as it is performed in NetFlow v9 is possible.

In addition to flow data export, IPFIX also defines templates for exporting reliability statistics on the flow capturing process itself. Furthermore, the PSAMP WG defined information elements for reporting error statistics [RFC 5477] in terms of absolute and relative error as well as confidence limits for a certain confidence level. How such accuracy information related to IPFIX metering processes can be reported is subject to ongoing standardization activities [75].

3.2.6 Summary

Flow capturing has more than 20 years of history. First approaches followed a stateful metering with pull-based reporting mode. Such approaches, like RTFM, have been feasible for monitoring large links, however, they required separate probes. When NetFlow became available as a router feature with more and more hardware support, it became the de facto standard for flow capturing in enterprise and carrier networks. Currently, NetFlow v5 is the most commonly used format. However, due to the transition to IPv6 the more flexible NetFlow v9 is increasingly used. That NetFlow has been optimized for hardware support becomes evident in the timestamp encoding. Due to protocol design issues, however, the timestamp precision suffers from protocol design issues in NetFlow v9. Since NetFlow is often implemented in hardware, it is possible to perform unsampled flow capturing for all traffic in enterprise networks or small ISP networks. However, in large carrier networks sampling is used in order to keep the amount of collected flow data low and limit the processing effort in routers.

In contrast to NetFlow, packet sampling such as sFlow is a mechanism commonly used in switches operating in sub-IP layers, which typically do not have as much hardware and processing capacity as routers. Thus, it is mainly used for data center traffic monitoring and at Internet exchanges (AMS-IX, DECIX) , which provide switching facilities to which routers of ISPs and global carriers connect, but do not operate the routers. Since this thesis is targeted towards measuring One-way Delays in WANs, sFlow and packet sampling are not considered further.

IETF standardization efforts in PSAMP and IPFIX have lead to generic and well designed protocols for packet sampling and flow capturing. IPFIX is even suitable as general protocol for streaming monitoring information and will most likely replace SNMP or Syslog [RFC 5424] in some applications. Current standardization activities and the involvement of equipment manufacturers show that flow capturing and flow data processing stays a hot topic.

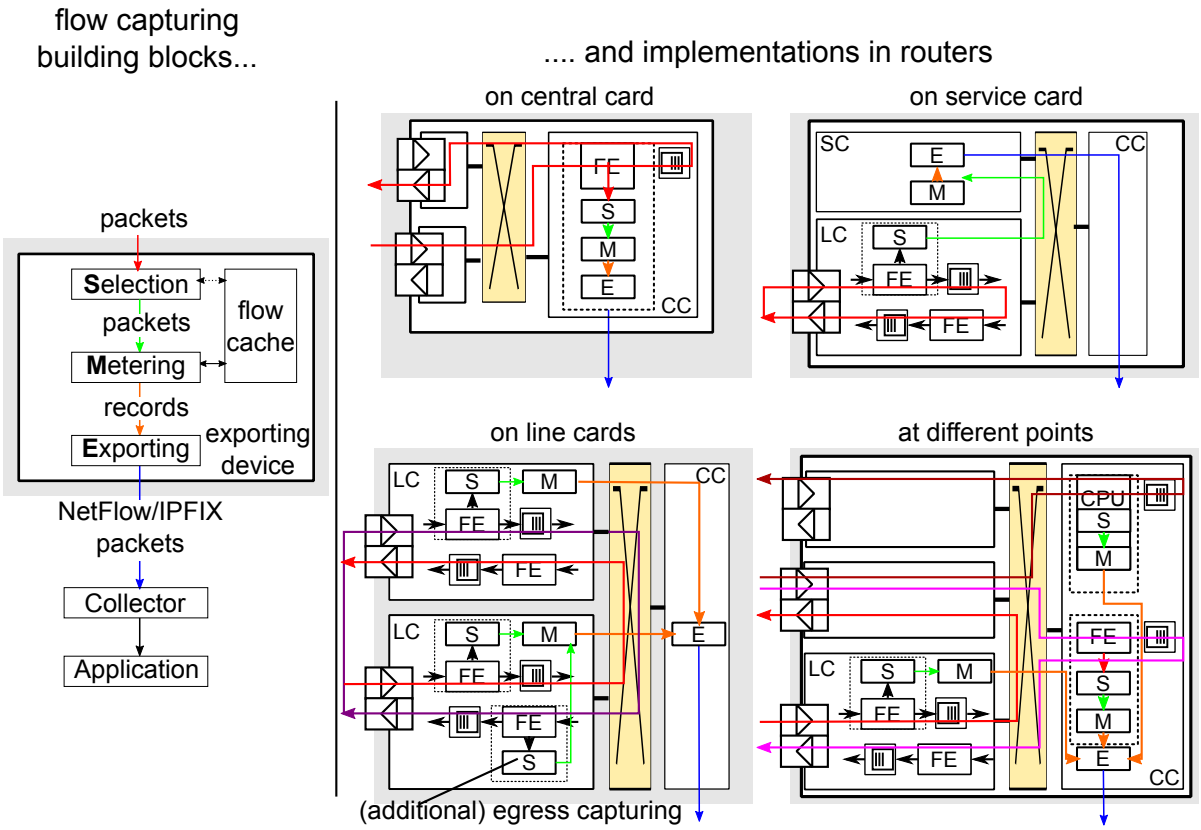


Figure 3.16: Flow capturing: building blocks and points for implementations in routers

3.3 Implementation in Routers and Probes

In order to understand the different characteristics of flow capturing implementations, it is important to know *how* and *where* flow capturing is implemented. This section gives an overview of different implementations in terms of placement of flow capturing functionalities in routers. The focus is on stateful flow capturing as it is implemented today with Cisco NetFlow or similar mechanisms.

Stateful flow capturing consists of the processes for packet selection, metering, and exporting (see Figure 3.16, left). These terms are aligned with IPFIX architecture terminology from [RFC 5470]. The selection process decides which packets are taken into account, i.e., it performs filtering or sampling [RFC 5475]. From the selected packets, the metering process creates flow records by means of the flow cache. The exporting process eventually encodes and sends records as NetFlow or IPFIX packets to the collector.

There is a large number of possibilities for implementing the selection, metering, and exporting processes in an exporting device. While flow capturing often is implemented as a router feature, there are also several implementations of standalone probes. Such probes receive packets that are mirrored from, e.g., a switch port and do selection, metering, and exporting, most often completely in software on general purpose hardware [76]. Some probes perform advanced features, such as defragmentation [77], or provide hardware support [78] for achieving higher performance.

Today, most flow capturing is performed in routers. Flow capturing as a router feature does not require additional equipment and can be implemented as part of the packet forwarding functionality. However, there are different implementation possibilities that differ in the location of the selection, metering, and exporting within the router. This affects which traffic and at which point in the router the traffic is metered. Figure 3.16 shows on the right four characteristic implementations. The first implementation (top left) shows a small router that performs all packet forwarding and also all flow capturing tasks close to the *Forwarding Engine* (FE) on a *Central Card* (CC). Flow records are selected and metered before queuing inside the forwarding process. Depending on the hardware capabilities of the *Line Card* (LC) or CC, some routers require additional hardware for performing flow capturing efficiently. Such hardware can be added as a dedicated *Service Card* (SC) (implementation top right), which performs metering and exporting. Selection or sampling can be performed on the LC before the packets are sent across the switch fabric to the SC. With such implementations, timestamps are created on the SC from packets mirrored before queuing. In large high performance routers, packet forwarding and also flow capturing is completely performed on the LCs (implementation bottom left). Flow records are sent to the CC, where they are put into packets and sent by the exporting process residing on the CC. Most commonly, flow capturing is performed on the ingress path of a LC. However, a router might also allow for enabling egress flow capturing (shown on the bottom LC). This is useful to monitor the traffic that finally leaves the router through a certain interface after address translations or tunneling features have been applied. In this case, timestamps are taken after major queuing stages (virtual output queues on ingress LCs). Whether records result from ingress or egress capturing can be indicated by the `flowDirection` fields in NetFlow v9 or IPFIX. Many router models allow flexible hardware configurations where FEs on LCs are optional and also the CC contains hardware and software based forwarding. Such an implementation is shown on the bottom right. Depending on the input interface, a packet might be captured on the LC with a hardware based FE, on the FE of the CC, or by software running on the central CPU. Since all flow capturing implementations in such a heterogeneous setup might behave differently, this has to be considered in terms of possible effects and errors. Typically, using the input port and flow direction fields of flow records as key is sufficient to distinguish different flow capturing implementations on the same router.

Flow capturing mechanisms in routers and probes are subject to continuous development. Research focuses on new flow metering concepts for software based probes [77] or metering that captures packet timing characteristics as part of the flow record. High effort has also been spent in the domain of flow sampling in order to increase the metering effort while still capturing valuable information. Estan et al. [79] propose automatic sampling rate control to handle *Denial of Service* (DoS) traffic. An interesting sampling method that tries to sample the same packets at different points in the network has been proposed by Duffield and Grossglauer [80]: *trajectory sampling*. It creates packet labels from invariant packet fields using hash functions, which allows correlating information associated with the same label obtained at different routers in order to get information about the path of a packet through the network. These research contributions show that flow capturing will extend towards new features that allow new monitoring applications. This thesis, however, is focused on data that is available from routers that are deployed today and thus does not consider such extensions in detail.

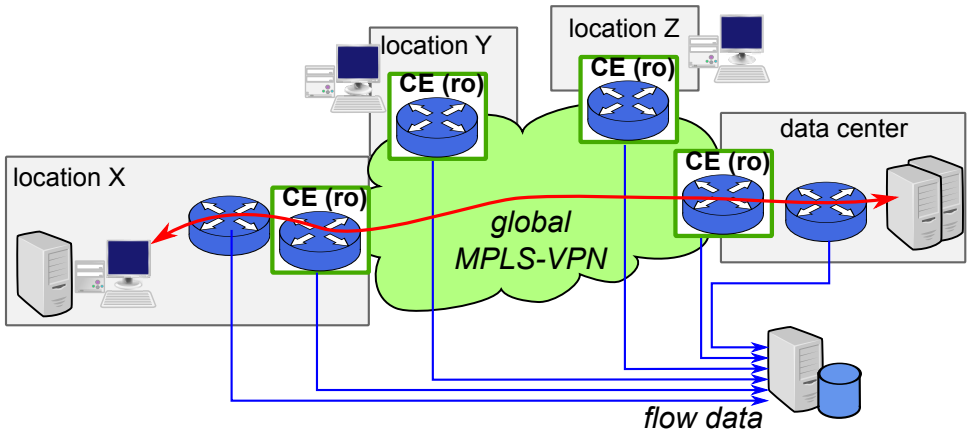


Figure 3.17: Typical flow capturing deployment in an enterprise network

3.4 Fields of Application and Deployment

The previous sections detailed flow capturing mechanisms as well as their implementation. As already introduced in Figure 3.3, flow capturing data is used for various applications, which are the focus of this section. Depending on the application, flow data from different points in the network is required. Thus, the application impacts flow capturing deployment, i.e, *where* and *how* flow capturing is enabled in routers and/or probes.

In the following, typical applications and deployment as well as current research topics for flow capturing in ISP networks and enterprise networks are presented. The flow capturing applications are classified into four categories, each handled in a separate subsection. The categories are Accounting and Reporting, Network Security, Traffic Behavior Analysis, and QoS Monitoring. This is a rough classification, since due to the different approaches and methods in this domain the classification borders are blurring.

Accounting and Reporting

One of the first and the still dominating application of flow capturing is traffic accounting and reporting. This covers coarse-grained network utilization statistics and traffic matrices for capacity planning, fine-grained utilization statistics such as traffic per application protocol, and per-host traffic analysis. To which extent flow data is captured and analyzed depends on the network size and effort spent: In large carrier and ISP networks flow capturing is often performed in core and border routers. Especially in the core, sampling is applied in order to limit the effort at high data rates. This is sufficient to obtain traffic core matrices and peering statistics about how traffic flows from and to other autonomous systems. In enterprise networks often a large number of edge routers capture unsampled flow data, which allows for detailed accounting on branch location or department level.

A typical flow capturing deployment for an enterprise scenario is depicted in Figure 3.17. The core network belongs to a carrier, which also operates the CE routers. Enterprise staff has read-

only (ro) access on CE routers, but can obtain flow data from them. Additionally the routers operated by the enterprise (e.g., in data centers) send flow data to a collector, where the data is processed in analysis applications.

One of the first documented systems that provides accounting and reporting functionality is Fluxoscope [81], developed for traffic accounting and reporting in the Swiss research and education network. Today, there are several commercial solutions available for accounting and reporting, such as Plixer Scrutinizer [82], Arbor Peakflow [83], Lancope Stealthwatch ([84]), NetQos [85] (acquired by CA in 2010), and IsarFlow [86]. For systems that provide accounting and reporting functionality, it is sufficient to operate at time scales of minutes, i.e., they can perform temporal aggregation of flow data.

Network Security

Since flow capturing does not provide a view on the packet content, it cannot provide as detailed information on possible attacks as an *Intrusion Detection System* (IDS) does, which scans the packet content for attack signatures. However, in contrast to an IDS, flow capturing can provide a network wide view on traffic and is therefore suitable for detecting suspicious traffic patterns of malicious activities. It is also useful for tracking down attacks or infected systems (possibly in combination with IDS or DPI information) [87].

ISPs often perform security analysis using flow data obtained from border routers. Obviously, such security analysis requires complete (unsampled) flow data. While several commercial systems already support security analysis, there is ongoing research in this area. In order to support the interactive analysis of security events, Mansmann [88] studied several visualization approaches and concluded that such approaches provide more insight than automated methods do. Classifier-based approaches for anomaly detection and cause identification have been studied by Münz [89]. Münz concluded that due to the unforeseeable nature of non-malicious traffic, there is a high probability for false alarms. However, certain frequent anomaly patterns (scans, password guessing) could be identified in order to automate alarm handling. While for security applications that analyze flow rates and communication relations of end systems a timestamp resolution in the minute range is sufficient, identifying anomaly patterns can demand for higher timestamp resolution, depending on the pattern definition and detection algorithm.

Traffic Behavior Analysis

While flow-based security analysis partly covers traffic behavior analysis, this field is much more general and has various applications. The goal is to identify traffic behavior based on flow data that is typical for certain applications, application usage, or network events. Results obtained in this way serve as input for traffic engineering, cause detection, and network design.

Wallerich et al. [90] developed a flow capturing based methodology for studying the persistency of flows, i.e., whether flows that are large in terms occupied bandwidth (“elephants”) will stay elephants for their lifetime. As a first result, they conclude that even with the high temporal aggregation that NetFlow typically provides due to active timers, NetFlow data is sufficient to

answer these questions. Second, they found that especially large flows are very likely to be persistent.

In addition to work that generally characterizes traffic behavior, there are several studies on how to extract certain effects from NetFlow data that indicate network or protocol problems. Limmer and Dressler focused on flow-based TCP connection analysis [91] and studied whether and how it is possible to infer whether TCP connections have been successful or whether there was a failure. They compared results obtained from packet traces and unsampled flow data, studied optimal timeout settings, and proposed additions to flow capturing mechanisms.

Dhamdere et al. [92] developed the "FlowRoute" concept for inferring router forwarding table updates. They used routing protocol messages and sampled flow data from backbone routers of a tier-1 ISP as input. The study shows that delayed forwarding table updates, as well as routing loops and other routing problems can be found using this approach.

Another approach that leverages traffic behavior analysis for identifying routing and other connectivity problems is the *Flow-based Approach for Connectivity Tracking* (FACT) [8]. The idea is to use unsampled NetFlow data obtained from an ISP's border routers for checking whether traffic is bidirectional to remote networks or whether there is unidirectional traffic only, identifying a connectivity issue. It was shown that this approach can detect issues that result from BGP problems. Furthermore, it was studied how the large amount of unidirectional traffic from network scans, for example, can be handled that is not caused by connectivity problems.

Besides extracting network effects from flow data, there is also work on identifying applications based on their flow-level characteristics. Schatzmann et al. showed in [93] that it is possible to identify webmail sessions in HTTPS traffic using unsampled NetFlow data. The classification approach is based on three types of features: the usage of other mail protocols in the server's vicinity, the typical timing patterns of such sessions, and periodic requests from clients.

Rossi and Valenti [94] focus on identifying P2P file sharing and P2P streaming sessions with flow data. They compare results from a packet trace by DPI with results obtained from unsampled NetFlow data generated from these traces. Depending on the P2P application and NetFlow timer settings, between 80% and 99% classification accuracy could be achieved.

QoS Monitoring

Besides analyzing traffic and network utilization, flow capturing can also be used for monitoring network characteristics that impact QoS. The work of this thesis lies in this domain, where only few studies have been conducted so far.

One important QoS metric in networks is packet loss, which has been studied by Gu et al. [95]. In their experiments, sampled flow data from two points in the network was used to estimate packet loss on a certain path. Packet traces from different networks served as input. From these packet traces, sampled flow data was created and different loss rates have been applied to the trace in order to evaluate the method. Results show that a packet loss of 0.5 % can be discerned at a 2 Gbit/s link with a sampling rate of 1/500. This approach requires that the same flows are observed at two points in the network, and that only flows that pass both points are taken

into consideration for calculating the packet loss on the path. Especially with sampling, where it is very improbable that flows of the same five tuple are observed at both points, this requires additional knowledge on topology, routing, or addressing, in order to fulfill this requirement.

In addition to packet loss, high network delays also impact QoS. An approach for OWD measurement based on special sampling techniques applied to flow capturing is presented by Lee et al. in [96]. In order to measure OWD between two observation points, timestamps of the same packets have to be available. Lee et al. propose a hash-based sampling technique that selects all packets of the same flows at two estimation points, such that the timestamps of the first and last packet can be used for OWD calculation. This approach is compared to hash-based packet sampling and trajectory sampling. Furthermore, different methods for OWD calculation are studied. The evaluations in these studies are not based on real NetFlow traces, but packet traces and a simulation environment are employed for obtaining the results. Thus, timestamp issues of the flow capturing devices themselves are not considered.

There is also a certain class of systems that use the IPFIX protocol as general data streaming protocol, but are not based on flow capturing mechanisms. [47] presents, for example, a monitoring system where IPFIX is used for reporting timestamps of packets for OWD measurement. They use a special packet sampling approach and no typical flow capturing. This packet sampling use case for OWD measurement using IPFIX is also mentioned in [RFC 5472].

Summary

The different classes of applications for flow capturing show that there are classical applications like accounting and reporting that do not require accurate timestamps and can also take sampled flow data as input. As soon as security analysis is considered, a complete (unsampled) view on traffic as well as timing characteristics become important, which holds even more for traffic behavior analysis.

In terms of QoS monitoring, precise flow record timestamps are vital, but have not been considered so far. Additionally, studies show that a high effort has to be spent if such QoS monitoring is performed with sampled flow capturing. Some approaches tune sampling mechanisms to fulfill QoS monitoring requirements.

The different studies show that today unsampled flow capturing is often available from several points in the network, at least in enterprise scenarios or networks of medium sized ISPs. This is a prerequisite for the flow-based OWD measurement approach presented in this thesis. Even if flow-based OWD measurement is not possible with *sampled* flow capturing today, current research on sophisticated sampling methods for QoS monitoring show that this might be possible in future.

3.5 Flow Data Processing

After having introduced the applications that use flow data, this subsection focuses on how flow data can be processed efficiently. There are two fundamental different approaches: *offline* and

online processing. Offline processing means that data is *stored* on a disk or in a database before it is analyzed and results are available. In contrast, online processing does not store the data, but processes data *in memory* right after it arrived at the collector. Depending on the application, it might be necessary to buffer data for some time in online processing. However, the overall processing speed has to keep up with the record arrival rate, thus buffering for coping with performance bottlenecks is only possible to a small extent. Online processing is also called *stream processing*, since the the data streams through the processing system.

In the following, flow data processing concepts used today are discussed. This covers selected commercial flow data collectors and analyzers, open source tools, frameworks currently in standardization, and current research in this area. Not every flow processing approach is addressed in detail, but important exemplary points are highlighted. A comprehensive overview on flow capturing and flow processing is provided on a website maintained by Simon Leinen [97].

Commercial flow processing systems, such as Arbor PeakFlow [83], CA NetQoS ReporterAnalyzer [85], or IsarFlow [86, 98] store flow data to a database or files before performing further analysis on it. This is an offline processing approach. In large deployments several collectors are necessary in order to cope with the high record rate. By deploying them close to the locations where high record rates occur, flow record traffic on the global network can be reduced. Furthermore, each collector can run analysis applications and hence the overall processing power of the monitoring system scales with the number of collectors deployed.

In addition to commercial software, also several open source tools have been developed in the last years. Flow-Tools [99] is a tool suite that provides a collector that stores flow records to files. It uses a Flow-Tools specific file format that keeps NetFlow timestamps as reported in the packet. Additionally, Flow-Tools contains several programs for performing analysis (filtering, tagging, statistics) on these files. SiLK [100] is similar to Flow-Tools in terms that it provides a collector that writes binary files and provides several tools for analyzing them. SiLK is explicitly targeted towards security analysis and provides more sophisticated analysis utilities than Flow-Tools. Another similar tool suite is nfDump [101], for which there is a database-based analysis frontend available, which is called nfSen [102]. nfSen provides a web-based GUI for creating reports, setting alarms and navigating through flow data. In summary, these open source tool suites also perform offline analysis similar to commercial software, since collectors always store flow data to files first.

An approach for online processing currently in standardization in the IETF is the *IPFIX Mediation Framework*. IPFIX Mediation is described in [RFC 5982] as “the manipulation and conversion of a record stream for subsequent export using the IPFIX protocol.”. The RFC highlights that there is a need for such general mediation functionality for three main reasons: growth of traffic volumes demands for distributed processing, multi-purpose traffic measurements extracts different metrics from the same original IPFIX data, and heterogeneous environments that demand processing different kinds of monitoring data while delivering results in a common format.

The mediation framework [103] defines mediators that consist of collecting process, intermediate process and exporting process. As shown in the example in Figure 3.18, a mediator receives data from IPFIX exporters or other data sources through collecting processes, performs differ-

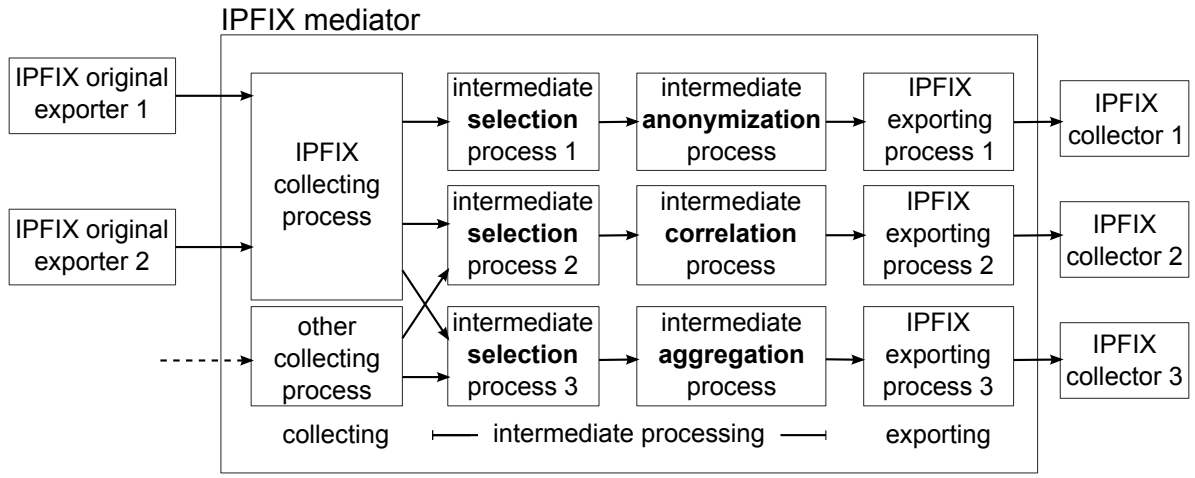


Figure 3.18: IPFIX Mediation Framework overview (based on [103])

ent intermediate processing steps on the data, and sends the data to IPFIX collectors through exporting processes. Currently, the following intermediate processes are described:

- **Intermediate Conversion Process**
  Converts non-IPFIX to IPFIX records

- **Intermediate Selection Process**
  Performs packet selection such as filtering or sampling, as defined by PSAMP [RFC 5475].

- **Intermediate Aggregation Process**
  Aggregates records by "Temporal Composition" or "Spatial Composition". Additionally, it might create statistics on aggregates, for which suitable IPFIX extensions [104] are in standardization.

- **Intermediate Anonymization Process**
  Deletes or anonymizes values of certain fields.

- **Intermediate Correlation Process**
  Correlates records for creating new metrics, such as OWD from packet reports, inter-arrival times or Jitter. Furthermore, records can be correlated to produce records describing bidirectional flows or records can be correlated with other metadata, e.g., from routing.

Inside a mediator the intermediate processing stage might chain several intermediate processes, as shown in the example in Figure 3.18. In this example, the mediator performs anonymization for selected records that are sent to collector 1. Furthermore, it performs some correlation on selected data from two sources and sends it to collector 2. On the third path it aggregates selected records from several sources and exports them to collector 3. More complex applications might even demand cascading of mediators. Additionally, it is possible to take IPFIX data from files instead of from the network as well as storing IPFIX data to files instead of exporting it. In

order to correctly deal with identifiers and timestamps in IPFIX mediation, protocol extensions are currently in standardization [105].

The IPFIX Mediation Framework is designed as an online processing approach that processes IPFIX records without storing them. By replacing IPFIX collecting processes with file readers and IPFIX exporting processes with file writers, it can also be used for offline processing. Some intermediate processes need to buffer flow records for a certain time, e.g., for aggregating records or correlating them. The framework document [103] describes several conditions for removing such buffered records from the cache: timeout based, if there are resource constraints, or based on an expiration policy. Furthermore, it is stated that for correlation processes such as correlating packet reports for OWD calculation, reports might be discarded if no matching report is found after a certain period. [103] describes a mechanism that defines a time-based sliding window of valid or active records that are in the cache and removed as soon as they leave the window. This is a similar approach as taken in this thesis for processing flow data.

In summary, the IPFIX Mediation Framework provides a flexible and powerful approach for online and offline processing of flow data. Concerning applications, Anderson et al. consider the IPFIX Mediation framework in their proposal for a distributed SIP monitoring scheme (SIPFIX) [106]. However, this is so far only a conceptual proposal and no implementation. Further related work on the framework or implementations is currently not known.

In addition to the current developments in IPFIX and commercial tools, approaches for flow data processing have also been in the focus of recent research work. Hofstede, Sperotto, Fioreze, and Pras compare in [107] the commonly applied approach of using databases with or without indexes to stream-based processing using nfDump. They conclude that unlike other results presented in earlier publications, using a *Data Base Management System* (DBMS) is not the best solution for flow data processing and that a DBMS is outperformed by nfDump for large data sets in terms of response time.

A framework for online processing of flow data for attack detection and backbone monitoring is presented in [108] by Dübendorfer, Wagner, and Plattner. The framework called UPFrame tackles the problem of online processing at the packet layer and efficiently distributes NetFlow-UDP packets to different plug-ins that perform processing within one computer or across the network. However, the framework does not provide further functionality, like chaining of plug-ins and handling non-NetFlow data, for example.

Another online processing approach, mainly for security applications is presented by Marinov and Schönwälder [109], who propose a “Stream-Based IP Flow Record Query Language”. In this language, queries are defined by filter, grouper, merger, and ungrouper statements that are mapped to a processing chain consisting of blocks that perform corresponding operations. The language also contains statements for specifying temporal constraints and uses a temporary storage for flow records during the processing. How such a buffering of records could be organized is not specified and an implementation of the system is not available so far. However, a buffering mechanism as described for IPFIX intermediate correlation processes or window-based mechanisms are applicable in this case.

In the database domain, online processing has been considered in a general way, not necessarily focused on processing of flow data or network monitoring information. There are two different,

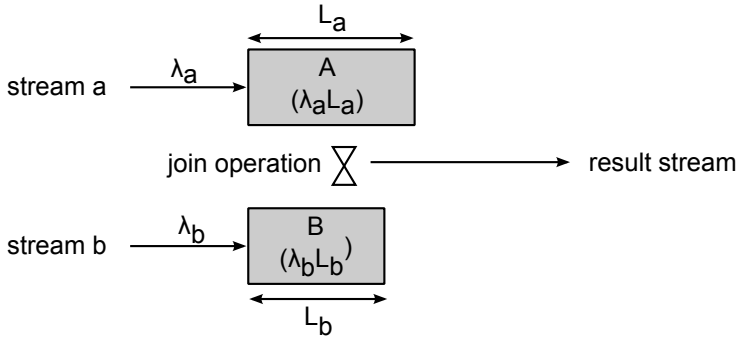


Figure 3.19: Sliding window join ([113])

however closely related, approaches: *data stream processing* [110] or *Data Stream Management Systems* (DSMS), and *Complex Event Processing* (CEP) [111]. In contrast to a DBMS that keeps a large set of predominantly static data on which changing queries are executed, a DSMS keeps the queries static and the data streams through it for creating the results. CEP, however, is not focused on streaming data, but on events where higher layer events are described by complex patterns of lower layer events. In CEP flowing data will thus trigger certain events that are reported to the interested parties according to the publish/subscribe paradigm. A comprehensive overview on these approaches and classification of existing implementations is given in a survey article of Margara and Cugola [112].

One of the problems considered in this thesis is matching flow data records from different data sources, in the simple case two record streams from different routers. Since the records are typically not available at the same time, online processing has to keep records in a *window* until both records are available and can be matched. In DSMS terminology, this corresponds to a window based operation on two data streams, more precisely a *window join* operation. Figure 3.19 shows this operation on two data streams and the resulting data stream. As shown in the figure, the amount of data tuples stored in memory depends on the window size L and the data rate λ. Choosing and adapting window sizes for dealing with memory constraints has therefore been considered in DSMS research.

Fundamental work on whether queries can be computed with bounded memory is presented in [114]. If this is not possible, the window size can either be explicitly specified or determined dynamically. The *Continuous Query Language* (CQL) [115], for example, specifies different window parameters in its stream-to-relation operations to explicitly define the amount of data to be buffered from the streams. In case of fixed window sizes, however, there have to be mechanisms that protect the system from problems due to memory limitations. Srivastava and Widom present an approach in [116] for memory-limited execution of windowed stream joins that tries to get good approximate results if only certain tuples can be taken into account due to memory limitations. Wu, Tan, and Zhou [117] present a method to dynamically adapt the window size based on characteristics of the input streams. Their approach considers intra- and inter-stream delay for calculating buffer sizes for worst-case scenarios. Flow data streams typically also exhibit unordered streams and certain delays resulting from timeout-mechanisms. However, flow records have several timestamps and traffic-dependent export characteristics, which complicate the problem and do not allow the application of the approach of Wu et al..

The concepts of the database domain have been taken into account for designing network monitoring systems, such as the Gigascope [118] system that can process high volumes of network monitoring data. It features its own query language and can work on different input data, such as packet traces, BGP updates, or NetFlow records. Join, merge, or aggregation operations specify a window that describes a certain amount of data on which the query is executed. Windows are defined on ordered attributes, not necessarily timestamps, and a heartbeat mechanism ensures that windows do not stall in the absence of new data. Gigascope has been developed by AT&T and is not publicly available. After the publication of the concepts in 2003 [118], no further work on it was published. Using DSMS for network monitoring has also been considered by Plagemann et al. [119], who studied how the general purpose DSMS TelegraphCQ [120] can be used for traffic analysis. They concluded that TelegaphCQ could be used. However, it is not suitable for being used as general traffic analysis tool due to certain restrictions. Apiletti et al. present the NetMine framework for traffic characterization in [121], which performs offline and online analysis. The online analysis is based on a DSMS and continuous queries for creating flow data from packet stream and filtering. Further refinement analysis, however, is performed offline.

In summary, the predominant approach used today by commercial as well as open source flow data processing tools is offline processing. Online processing is gaining interest, as the IPFIX Mediation framework that is currently standardized shows. Interestingly, concepts of the data base domain, such as DSMS or CEP have not been used for flow data applications to a large extent. This is most likely due to performance limitations and the need for more sophisticated processing steps that are better directly implemented in program code than specified using a query language. As shown even new query languages have been defined for flow processing independently from DSMS concepts mainly due to performance reasons and query language restrictions. While online processing is often used in research for flow data processing, no implementations or frameworks are available that support window-based processing, as it is required for, e.g., general correlation or aggregation functions. Furthermore, window handling and impact of window sizes on flow record correlation and aggregation have not been addressed so far.

3.6 Summary

This chapter detailed flow capturing terminology, mechanisms and applications, which is a prerequisite for flow-based delay measurement. While there have been several flow capturing approaches, NetFlow is a de facto standard and the activities related to its successor IPFIX show that flow capturing gains more and more importance. Since flow capturing is most often a router feature, flow data is often available from networks and serves as input for different applications. Despite the large amount of applications for flow capturing, flow-based OWD measurement has not been studied so far.

Due to distributed architectures, systems that collect and process flow data can analyze large amounts of data in an offline fashion. Online processing for analyzing flow data is hardly performed today despite the current efforts in the streaming database domain. This thesis proposes and studies an online processing approach for processing flow data for OWD measurement in

Chapter 5. However, the first step will be the development of the measurement approach itself in the next chapter.

4 Delay Extraction: Impact of Flow Capturing Properties and Errors

The previous chapters introduced the fundamentals of network measurement and flow capturing. These topics are input to this chapter that addresses how OWD can be extracted from flow data and which properties and errors have to be considered for obtaining proper measurement results.

The first section of this chapter motivates the approach of flow-based OWD measurement proposed in this thesis. Afterwards, the second section gives an overview of the basic OWD extraction approach using perfect flow data, i.e., with considering network effects only. Based on this, the third section presents the impact of flow capturing effects and measurement errors. Since timestamp errors have high impact on OWD calculation, these are handled separately in section four. The last section sums up the results.

An efficient processing approach for OWD extraction that can handle the effects and errors is presented in Chapter 5. From the effects detailed in Section 4.3 and Section 4.4 follows that an *exporter profile* is required to describe, compensate, and quantify effects and errors in flow capturing for obtaining reliable flow-based OWD measurements. Chapter 6 elaborates the concept of the exporter profile in more detail.

4.1 Rationale for Flow-based Delay Measurement

As introduced in Section 2.3 there are several delay measurement approaches currently deployed in networks. Flow-based OWD measurement exhibits special properties that make it attractive compared to other measurement approaches. This section first highlights properties of existing delay measurement approaches and then uses these properties for comparing current approaches to flow-based OWD measurement.

Continuous systematic delay measurements are predominantly performed in enterprise networks. While there are Internet-scale delay measurements across different network domains (e.g., the RIPE atlas project [122]), they serve only for observing phenomenons and do not have direct impact on network management. This has several reasons:

- Network-based services in an enterprise are business critical and hence network performance impacts application response time, which directly impacts the enterprise's produc-

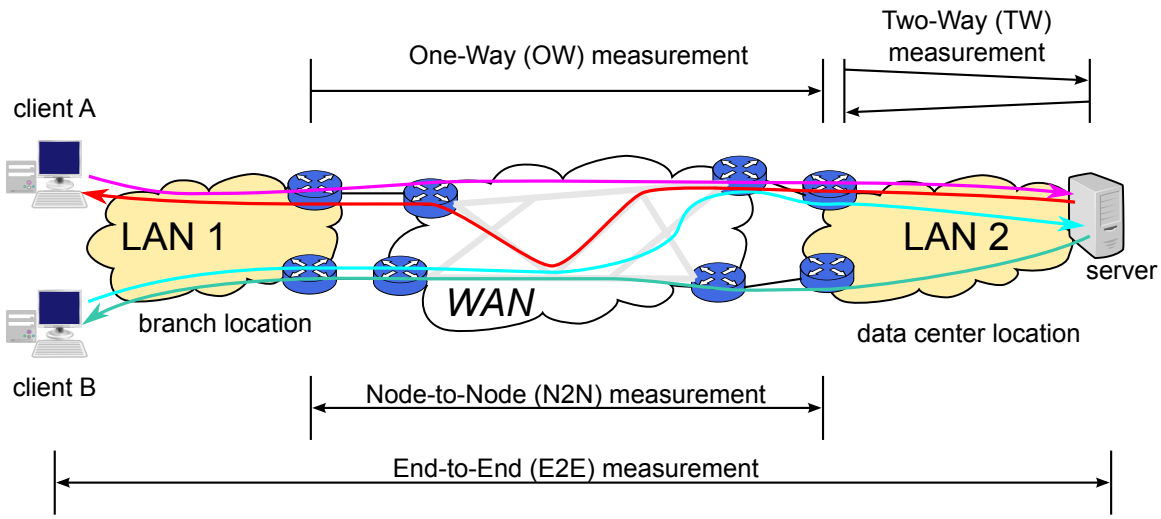


Figure 4.1: Traffic paths and measurement paths in an enterprise network

tivity. A good service quality can only be assured by continuous measurements since relying on user complaints is inappropriate. If users complain there is already severe service degradation and productivity is affected.

- The network and service management staff can react on bad measurement results immediately. Routing schemes or QoS parameters can be adjusted to address problems.
- All parts of the network are well managed and there are SLAs for the parts of the network that are not owned by the enterprise (e.g., MPLS VPN). Continuous measurements allow for checking whether SLAs are obeyed. This is a prerequisite to inform the provider about problems and potentially get financial compensation for service degradation.

In the Internet there are currently no inter-domain SLAs and as there are multiple transit providers involved (see Section 2.1.3), the facts that motivate delay measurement in enterprise networks are not valid in the Internet.

For comparing delay measurement approaches, there are three important criteria to consider that will be elaborated in the following.

- *One-Way* (OW) or *Two-Way* (TW) measurement
- *End-to-End* (E2E) or *Node-to-Node* (N2N) measurement
- Active or passive measurement

The criteria are elaborated in the following based on the enterprise network scenario depicted in Figure 4.1. In this figure the clients A and B at an enterprise branch location access a service at the data center across the WAN.

The figure shows on the top examples for OW and TW measurements. OW measurements require two observation points and therefore provide information on unidirectional path characteristics. In contrast, TW measurements use only one observation point and perform measurements by taking request-response patterns into account. Hence, TW measurements always provide combined information on the forward and reverse path. Furthermore, the characteristics of the responding entity influence TW measurements.

TW measurements can observe the same path characteristics as clients that use the same paths and are thus suitable for, e.g., monitor network impact on applications. However, TW measurements are definitely not suitable for tracking down network problems that occur on a single path. This is due to the fact that path characteristics are not symmetric, especially when problems arise. For example the forward and reverse path could take different routes as for the request and response of client A in Figure 4.1 and therefore experience different propagation delays. Or the load on the paths is different and therefore they are affected by different queuing delays. Hence, only OW measurements are suited for tracking down delay problems.

Figure 4.1 shows on the bottom that measurements can be performed between different points in the network. Each measurement point can either be an end system or a network node (e.g., router, switch, probe at a mirror port). E2E measurements are conducted between end systems and thus take the complete path into account. Measurements between nodes are called N2N measurements in this thesis and consider a certain network path only. Measurements between nodes and end systems (N2E or E2N) are also possible. As Figure 4.1 shows, N2N measurements between edge routers of an enterprise network are suitable in order to measure the WAN characteristics. Considering the overall delay, the WAN contributes the largest fraction of the propagation delay due to the long distance links. Since the WAN bandwidth is typically considerable lower than enterprise LAN bandwidth, the WAN also contributes the largest fraction of queuing delay. Hence, N2N measurements in enterprise networks provide information on the paths on which delay effects occur predominantly. Furthermore, N2N measurements are independent of end systems and are thus easier to deploy and manage from a network operation point of view.

The third criterion for comparison is whether a measurement approach performs active or passive measurements. While active measurements can be conducted in regular intervals and thus provide measurement results continuously, there is no direct relation to production traffic. As Figure 4.1 shows the traffic can take different paths and thus it is important to know, which application transactions actually experienced high delay on which path and to which traffic class they belong. Only with passive measurements such information is available from the measurement data itself. Additionally, active measurements produce artificial traffic and consume CPU resources in routers with N2N measurements. Both effects must not be neglected, especially on low bandwidth links and routers or switches with limited CPU resources.

Regarding the three criteria, it becomes obvious that a passive OW measurement approach suits best the requirements for enterprise delay measurements. Furthermore, passive OW measurement works on an N2N basis, which is also suitable in the considered scenario.

Table 4.1 shows different delay measurement approaches that have been introduced in Section 2.3. Ping probes are used for basic measurements only. Application probes, distributed network probes, or trace analysis are common approaches used in enterprise networks today.

Table 4.1: Comparison of delay measurement approaches

approach	**scope**	**OW/TW**	**active /passive**	**typical measurands**	**effort**	**comment**
ping probes	any	TW	active	delay, loss	low	basic tool
application probes	E2E	TW	active	delay, server response time	medium	requires dedicated measurement machines and application knowledge
distributed network probes	N2N	OW	active	delay, loss, jitter	medium	available as router feature. Per-path setup required. CPU and bandwidth impact.
trace analysis	N2E	TW	passive	delay, server response time	medium	medium effort to configure. Often dedicated equipment required. High packet processing effort.
coordinated packet sampling	N2N	OW	passive	delay	high	additional packet processing equipment required. No large scale deployment/product known.
flow-based	N2N	OW/TW	passive	delay, loss	low	flow capturing is a router feature, often already enabled. Flow data processing effort see Chapter 5

However, they are either active approaches, or passive TW approaches. Only coordinated packet sampling is a passive OW approach and thus provides the ideal combination of properties. However, this mechanism is hard to deploy since it requires a special packet sampling mechanism.

The last row in Table 4.1 shows the flow-based OWD measurement approach. It provides the ideal combination of properties. Furthermore, it requires almost no additional effort, since flow capturing is often already performed in routers and the feature can just be enabled if this is not the case. There is an effort for processing the flow data for extracting OWD samples, but Chapter 5 will introduce an efficient processing approach for this task. In summary, flow-based OWD measurement is very attractive.

4.2 Extracting Characteristics Assuming Perfect Flow Data

This section introduces the approach for extracting network characteristics from flow data, without considering properties and effects of the flow capturing mechanisms in detail. The main measurand of interest for extraction is OWD. As shown in the following also other quantities, such as packet loss, have to be taken into account for proper OWD measurement. At this stage, no measurement errors or flow capturing effects are considered. This means that perfect flow data from a stateful flow capturing mechanism is assumed, where each flow record describes a complete flow and has accurate timestamps. The following sections first introduce the terminology and general requirements associated with the OWD extraction. Then, the network effects are considered together with their impact on measurement.

4.2.1 Extraction Approach and Terminology

Extracting OWD from flow data works if there are at least two Observation Points (OP) (see Section 3.1.3) that create flow records for a flow. Figure 4.2 depicts such a scenario that also illustrates the terminology used throughout this thesis: A flow of packets traverses the network from source A to destination B and two devices on this path capture flow data. In typical scenarios flow capturing happens in routers (see Section 3.3). Therefore, a router symbol is used in figures.

The OPs where packets are metered to create flow data determine the points in the network between which OWD can be measured. Regarding routers it is important to consider that an OP is not the router itself, but a point inside the router. This is essential since router-internal delays (e.g., queuing delays) contribute to OWD (see OWD definition in Section 2.3.1). Additionally, there could even be several OPs for the same flow in one router (see flow capturing implementations in Section 3.3). Figure 4.2 shows the typical ingress flow capturing where the OP is in the ingress forwarding engine and hence in front of the queues.

Two OPs form an *Observation Point Pair* (OPP), as shown in Figure 4.2 for the two OPs OP_1 and OP_2. Packets of a flow first pass the first OP of an OPP (OP_1 in Figure 4.2) and the second OP (OP_2 in Figure 4.2) thereafter. The flow data generated from the flows passing these OPs is input for calculating characteristics of the network path in between, called *OPP path*. OWD

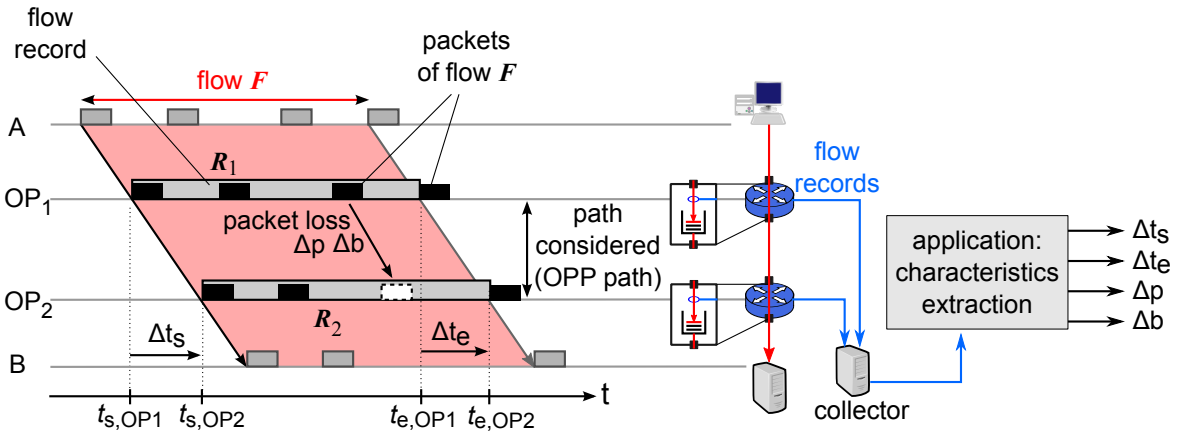


Figure 4.2: Extracting path characteristics from flow data: terminology

measured based on flow data will therefore always be an N2N measurement related to path segments and not an E2E metric as, e.g., defined in [RFC 2679].

Section 3.1.1 highlights that flow capturing obtains record timestamps from packet traversal times, i.e., there is only one timestamp associated with a packet. In the following calculations and illustrations, flow record timestamps are defined to correspond to the time of the first bit of the packet that a flow timestamp is related to.

OWD extraction requires the calculation of flow record attribute differences between the two observation points of an OPP. As the following sections show, calculating time stamp differences is not sufficient, but packet and byte count differences have to be calculated as well due to network effects. An attribute *difference* for an OPP is defined as the attribute value at the second OP minus the attribute value at the first OP. Figure 4.2 illustrates this for a packet loss in the flow that results in a negative *byte difference* $\Delta b = b_{OP_2} - b_{OP_1}$ and a negative *packet difference* Δp. The OWD on the OPP path results in the *start time difference* $\Delta t_s = t_{s,OP_2} - t_{s,OP_1}$ and *end time difference* $\Delta t_e = t_{e,OP_2} - t_{e,OP_1}$. In the basic example presented here, these *timestamp differences* equal the OWD samples. However, the following sections show that this does not hold in the general case.

The right hand side of Figure 4.2 depicts the application that extracts the characteristics from flow records taken from the collector. Figure 4.3 shows a general model of this application. This extraction model shows the basic processing steps and will be extended throughout the next sections with steps necessary for handling network effects, flow capturing effects, and flow data errors. Chapter 5 takes this abstract model as input and develops an efficient online processing approach.

As shown in Figure 4.3, records from several exporters arrive at the collector, among them the records $\mathbf{R}_1$ and $\mathbf{R}_2$, which we consider in the following. For simplification, the records are illustrated by three fields only: the Observation Point OP, the flow keys $\mathbf{K}$ and the attributes $\mathbf{V}$.

The processing for extracting path characteristics, such as OWD, can be divided into three fundamental steps:

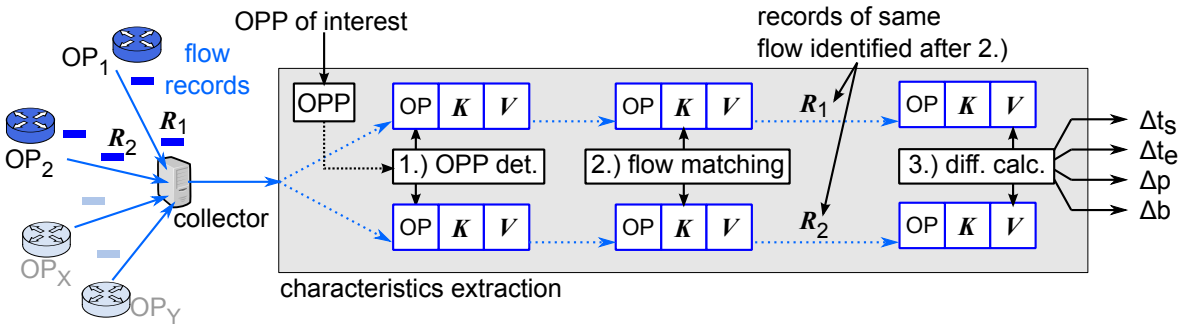


Figure 4.3: Characteristics extraction model

1. OPP determination based on OP fields
2. Flow matching based on **K**
3. Attribute difference calculation based on **V**

The first processing step filters on records of the OPP of interest. Additionally, the first step *orders* the records according the OPP definition. This is required in order to perform proper difference calculation. Relying on flow timestamps for OP ordering is not sufficient due to timestamp errors as Section 4.4 will show.

The second processing step checks if both flow records describe the same flow **F** across the network based on the flow key **K**. In the example of Figure 4.3, the records $\mathbf{R}_1$ and $\mathbf{R}_2$ are identified after this step. In the third step, the ordering of records in terms of OPP is known as well as that the records belong to the same flow. This allows for the correct calculation of the attribute differences Δt_s, Δt_e, Δp, and Δb.

While there is not much choice for the flow attributes taken from the attributes **V**, there are several degrees of freedom concerning the properties that identify the OP and the flow. The requirement for OP determination is that the values contained in the OP field must uniquely identify a point in the network, such as a router interface. An identifier consisting of the router address and an interface identifier fulfills this requirement.

Key values for flow identification must ensure that the mapping between packets and flows is consistent across all OPs, i.e. the values used must be *path invariant*. Thus, *path variant* fields, such as, e.g., IPv4 TTL, must not be used as key values. This is the same requirement as for OWD extraction based on coordinated packet sampling (see Section 2.3.3) with coordinated sampling [123]. The latter is obvious, since flow-based OWD measurement corresponds basically to coordinated packet sampling based OWD measurement without temporal and spatial aggregation of packets to flows.

The amount of OWD samples that can be obtained from flow data depends on the flow key definition that itself has not been considered so far. An upper bound for samples is obviously taking every *path invariant* fields into the flow key. However, the flow key chosen in reality is often a subset of *path invariant* fields, since, e.g., packet size and TCP sequence numbers are path invariant, but it does not make sense to include them in a flow key. The lower bound is

defining a flow as all packets passing an OP, i.e., a highly aggregated notion of a flow. This would allow for creating two OWD samples only: one at the activation of an interface/router and the other at shutdown. A sensible definition of a flow key, if OWD should be calculated from flow data, is therefore using endpoint defined flows (e.g., using IP addresses only). Using the five tuple (i.e., transport layer endpoints) is straightforward since it is common for flow capturing deployments today (e.g., NetFlow v5) and represents a good trade-off between effort of flow capturing and amount of OWD samples that can be calculated.

4.2.2 Impact of Network Effects

The previous section introduced the principle of network characteristic extraction and the related terminology. This section focuses on network effects that interfere with reliable determination of these characteristics, while still assuming that perfect flow data is available. As introduced above, flow-based OWD calculation corresponds to passive packet based OWD calculation with using timestamps of the first and last packet of a flow only. Consequently, OWD extraction from flow data is only possible if the flow start timestamps at both OPs correspond to packet traversal times of the *same packet* and the flow end timestamp at both OPs correspond to packet traversal times of the *same packet*.[1] This property is defined as *Start-End-Correspondence* (SEC). If SEC holds, the flow record timestamp differences (Δt_s, Δt_e) can be taken as OWD samples.

The following subsections elaborate three categories of network effects that impact SEC. Finally, the extraction model presented above will be extended to handle these cases. The three categories are *packet count change*, *identifier change*, and *partly disjoint paths*.

Packet count change

If the packet count in flow records changes on an OPP path, this is caused by effects on path segments or within devices. Three important cases are illustrated in Figure 4.4 and detailed in the following. For all cases, an OPP path is considered with routers and middleboxes between the two OPs that do not export flow data, thus details of the traffic there are not visible. Such scenarios are often found in enterprise networks (see Section 2.1.3).

The first case is *packet loss* on the OPP path, which may happen due to buffer overflow in routers or transmission errors on links. As illustrated in Figure 4.4 on the left, three different loss positions within a flow can be distinguished that have different impact on flow record timestamps. First, the first packet of the flow can get lost, which leads to the problem that Δt_s does not reflect the OWD, but the time difference between the first packet on OP1 and the second packet on OP2 (no SEC). Second, a packet loss can affect the last packet of the flow, leading to the same problem, however, with Δt_e. Third, a packet in the middle of the flow can get lost, in which case SEC holds and thus Δt_s and Δt_e are OWD samples.

[1] In general, it is sufficient to know the correspondence of the first packet's timestamps in order to take Δt_s as OWD sample and the correspondence of the last packet's timestamp in order to take Δt_e as OWD sample. However, if there is no correspondence on either start or end, it is practically impossible to judge whether it is on start or end. Thus, SEC is defined using the conjunction.

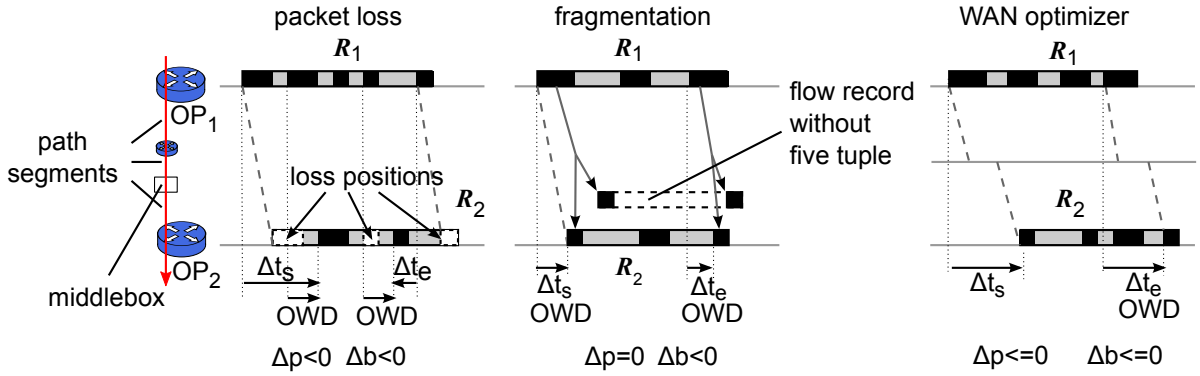


Figure 4.4: Packet count change scenarios

Packet loss in one of the three positions always leads to $\Delta p < 0$ and $\Delta b < 0$. However, without additional knowledge, it is impossible to distinguish the harmless third case from the cases that affect SEC. Consequently, OWD sample creation from all cases where $\Delta p < 0$ and $\Delta b < 0$ must be denied. A special case is a flow with one packet, which gets lost on the OPP path. This will lead to a flow record at OP_1, but none at OP_2. Thus, the extraction mechanism has to take care for such cases where it does not find a matching record.

The second scenario in Figure 4.4 illustrates *packet fragmentation*. This effect affects flow records if transport protocol fields (e.g., port numbers) are part of the flow key, since transport protocol headers are not present in fragments. Thus, fragments cannot be associated with the same flow as packets that contain transport protocol headers. This problem exists in most flow capturing scenarios where flow capturing devices do not reassemble fragments and transport protocol fields are in the flow key, such as with NetFlow.

With fragmentation, the same number of packets will be present in the flow records at OP1 and OP2 ($\Delta p = 0$). However, the record from OP2 contains less bytes than the records from OP1. Thus, such cases can be detected by $\Delta b < 0$. As a conclusion, even if start and end timestamp might not correspond to exactly the same packet in this case, OWD calculation will be correct and it is defined that SEC holds in case of fragmentation. Even if today fragmentation is often disabled in routers, OWD extraction has in principle to be aware for such cases, since fragments can often be found (see Section A.1).

There could be middleboxes on the path that cause packet count change. One important device class are WAN optimizers that improve application performance on long distance links by compression, caching, and shortcuts for protocol handshakes. In most cases, there is a pair of WAN optimizers at the edges of the long distance link, where the ingress WAN optimizer, e.g., performs compression and the egress optimizer handles decompression. This scenario for an ingress WAN optimizer is depicted on the right of Figure 4.4. While flows for which the device does not perform optimizations will pass the device unchanged, other flows will completely change in terms of byte and packet count as well as in timing. Due to optimization and compression, there will be $\Delta p \leq 0$ and $\Delta b \leq 0$, which is the same as for packet loss. On the egress WAN optimizer, there will be accordingly $\Delta p \geq 0$ and $\Delta b \geq 0$. Obviously, OWD samples must not be created from such flow records and both SEC-violating cases can be detected. From a deployment point of view, a solution to this problem could be to enable flow capturing on the

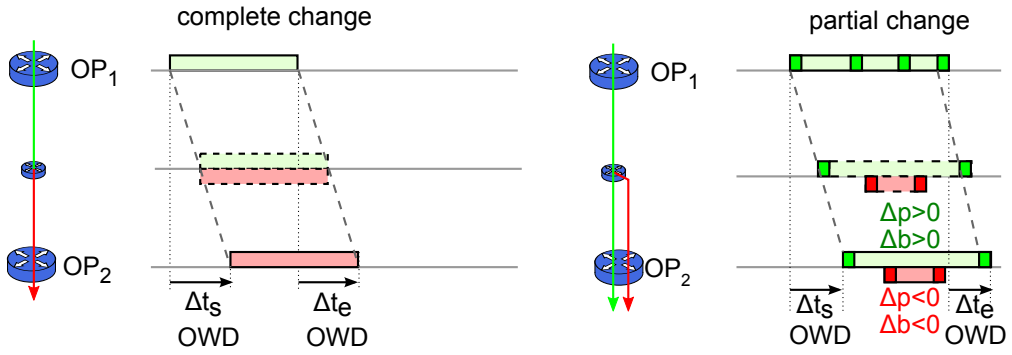


Figure 4.5: Key value change impact

interfaces of the middleboxes, therefore turning them into OPs. This would split the previously considered OPP path into two OPP paths, on which no WAN optimizer effects happen and OWD can be reliably calculated for these path segments. The delay introduced by the WAN optimizer itself, however, can still be measured only for cases where it leaves packets unchanged and SEC holds.

In addition to the previously mentioned scenarios, there might also be special routing schemes or (transient) routing loops, in which case packets traverse an OP more than once. In these cases the amount of packet traversals for the OPs of an OPP differs, thus SEC does not hold. Such cases lead to $\Delta p \neq 0$ and $\Delta b \neq 0$ and are hence detectable.

Key value change

One important step before calculating attribute differences is flow matching, which requires to find the same flow on both OPs. However, flows are identified by key values that might change on the OPP path in certain scenarios, i.e., network effects can lead to cases where the properties *are not fully path invariant*. Figure 4.5 depicts the two fundamental categories of *complete change* and *partial change*.

In the case of a complete change, key values change *for all packets* at a certain point in the network, for example at a network address translator (NAT) or with tunneling mechanisms. According to the flow definition, this splits the data transmission into two flows, as shown in Figure 4.5. There is still SEC in this case and OWD could be calculated. However, flow matching will not work unless the translation table of the device is known, which is typically not the case. Depending on the NAT type, only some key values change (e.g., source addresses change, destination addresses do not change). This would allow applying matching heuristics, which is not considered further in this thesis. As with WAN optimizers, turning on flow capturing at the interfaces of the key changing device would allow to extract characteristics of the path segments.

Partial change only affects key values of *a fraction of packets*. As Figure 4.5 shows this leads to a new flow from the point of key value change, while the original flow continues with fewer packets. Since for the original flow there is $\Delta p < 0$ and $\Delta b < 0$, this cannot be distinguished from

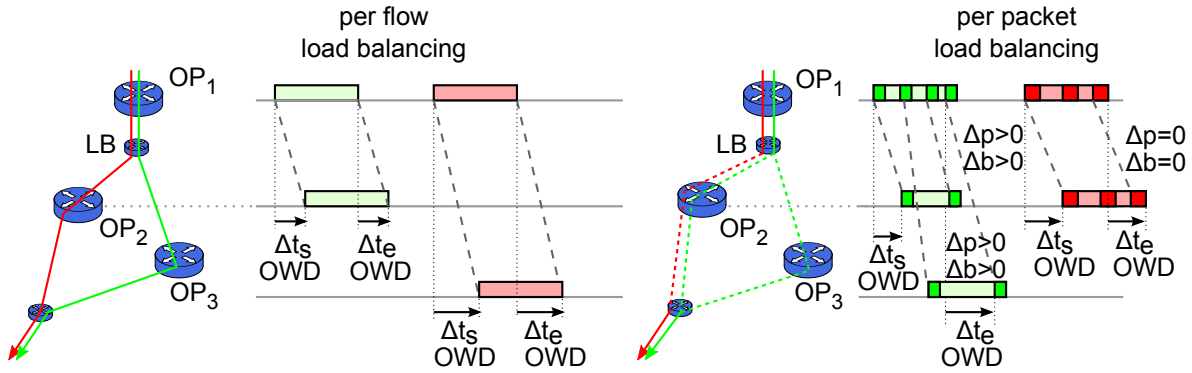


Figure 4.6: Effects from load balancing

packet loss and no OWD samples must be generated from the original flow alone, despite SEC could hold as for the illustrated case. Since typically not all key values change, e.g., the endpoint addresses will be untouched, there will just be an additional subflow. Based on the remaining unchanged keys the original and the subflow can be merged. Subflow merging demands for the consideration of flow capturing properties and is thus detailed in Section 4.3.1.2.

Partly disjoint paths

The primary goal of routers is to forward packets towards the destination. If there are several paths with similar characteristics leading to the destination, routers may perform load balancing (see Section 2.1.3), which leads to partly disjoint paths.

Figure 4.6 shows a scenario where a flow passes OP_1 after which load balancing (LB) is applied and packets are sent on the path to OP_2 as well as to the path to OP_3. On the left, *per flow load balancing* is applied, i.e., all packets of a flow are sent on the same path. In this case, flow records from the two OPPs can be matched and SEC holds, thus OWD samples can be calculated.

In the case of *per packet load balancing* on the right, the first flow is split up. Here, SEC does not hold for both OPPs and this case can be detected by $\Delta p > 0$ and $\Delta b > 0$. However, also in the case of per packet load balancing SEC can hold, if by chance all packets of a flow take the same path.

Load balancing is effectively equivalent to rerouting. Thus, typical rerouting events lead to similar effects, however, only sporadically. If a flow becomes rerouted, the effects are similar to the first flow on the right hand side of Figure 4.6, i.e., SEC does not hold.

Consequences

All network effects of the categories *packet count change*, *identifier change*, and *partly disjoint paths* need special attention in order to validate that SEC holds and correct OWD samples can be created. As shown, SEC cannot be evaluated by Δp alone, but Δb has to be taken into account

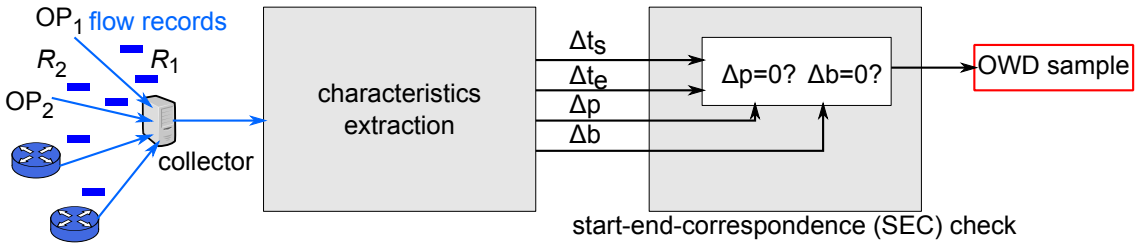


Figure 4.7: Extraction model extended by the SEC check block

as well. In addition to this, no additional knowledge on devices or effects on the path is required in order to detect whether SEC is potentially violated.

These considerations lead to an extension of the extraction model shown in Figure 4.7. After the characteristics extraction block, a SEC check block checks whether $\Delta p = 0$ and $\Delta b = 0$ hold before forwarding Δt_s and Δt_e as OWD samples.

4.3 Impact of Flow Capturing Effects and Errors

The previous section introduced how OWD can be extracted from flow data and how network effects impact the extraction principle. This was under the assumption of perfect flow data and independent of the capturing mechanism, i.e., flow records precisely described complete flows.

This section deals with effects from the flow capturing principle itself in the first subsection and with errors present in flow capturing mechanisms in the second subsection. The considerations in this section apply to stateful unsampled flow capturing mechanisms, such as NetFlow (see Section 3.2.4).

Timestamp errors are considered separately in Section 4.4.

4.3.1 Effects resulting from Flow Capturing

Due to stateful flow capturing, temporal aggregation of packets into flows impacts OWD extraction. Furthermore, flow keys containing path variant fields lead to certain problems. This subsection addresses both problems, which are flow capturing effects, not errors.

4.3.1.1 Temporal Aggregation

Stateful flow capturing performs temporal aggregation of packets into flow records. Flow capturing approaches like NetFlow create flow records based on timeouts, as introduced in Section 3.2.4. Timeout handling between two OPs is not synchronized and thus even with the same timeout settings configured, each OP might aggregate a different number of packets of a flow

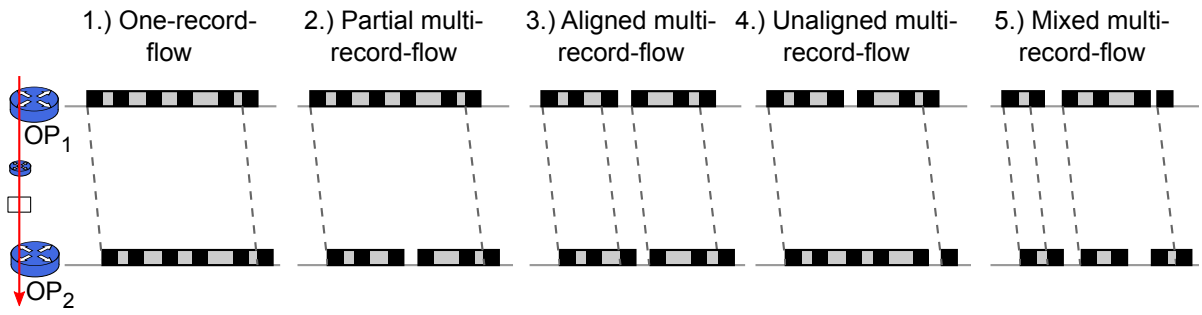


Figure 4.8: Flow record alignment

into a flow record. The created records hence differ in terms of the collected attributes and are not directly comparable. Even with absolutely precise timeout handling, network effects can lead to different packet interarrival times on each OP (jitter), potentially triggering a timeout on one OP and not on the other.

Different temporal aggregation on OPs impacts the number of records that each OP creates. Furthermore, different temporal aggregation leads to different alignment of records. Records are aligned if SEC holds. If more than two records are created for a flow at an OPP, there are different alignment scenarios. Figure 4.8 illustrates the following five exemplary cases.

1. **One-record-flow**
   Both OPs export exactly one record. This is the simplest case that has been considered for OWD calculation so far.

2. **Partial multi-record-flow**
   One OP exports several records for a flow while the other OP exports one record only. Here, the start and end times are aligned as depicted and OWD can be calculated accordingly.

3. **Aligned multi-record-flow**
   At each OP the same number of records is exported and the records are *aligned*. Thus, SEC holds for each record pair and an OWD sample can be created.

4. **Unaligned multi-record-flow**
   At each OP the same number of records is exported, but the records are *unaligned*. In terms of number of OWD samples, this can be compared to case 2.)

5. **Mixed multi-record-flow**
   This case is a mix of cases 2.), 3.) and 4.), where for each aligned record OWD samples can be created, but not for unaligned records.

Depending on traffic characteristics, there are more aligned or unaligned multi-record-flows. If, e.g., the traffic is bursty with long breaks in between, it is very likely that two OPs with the same timeout configurations run into an inactive timeout and produce aligned records. Contrary, if

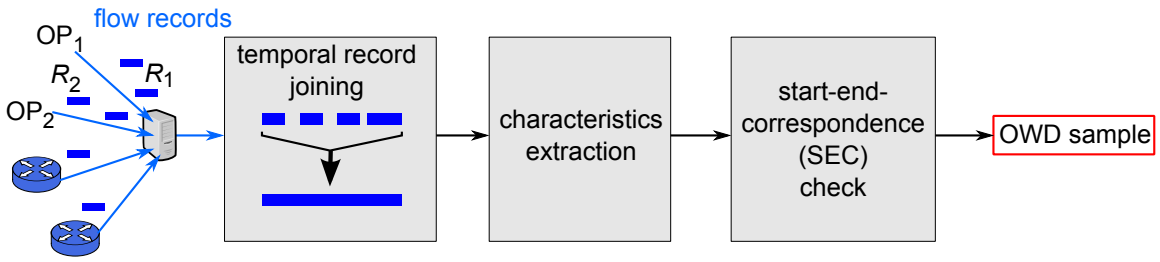


Figure 4.9: Extraction model extended for temporal joining of records

there are long flows, which are cut into records by active timeouts, it is very unlikely that the resulting records are aligned.

In addition to the alignment, of course, Δp and Δb have to be considered. The simplest method is joining the records of a flow on each OP for Δp and Δb calculation and if there is a difference, none of the records is used for OWD calculation. There are more sophisticated approaches discussed in Section 5.2.3 that check for alignment first and join records only in required cases. In any case, a block for temporal record joining has to be added to the extraction model. This block joins records before they enter the characteristics extraction block, as illustrated in Figure 4.9.

4.3.1.2 Path Variant Fields in the Flow Key

This subsection revisits the problem of key value change (introduced in Section 4.2.2). It considers this problem in more detail and in combination with temporal aggregation as well as other effects from flow capturing. This problem especially arises with NetFlow flow definitions that include the ToS byte as path variant field in the flow key (e.g., the static flow definition of NetFlow v5). The ToS byte contains the DSCP and ECN flags (see Section 2.1.1), which both might change on a path due to DiffServ policing, remapping of DiffServ classes, or congestion in ECN enabled routers. Hence there are several subflows (see Section 3.1.1) for an IP transport layer (IPTL) flow that can change on the path. With other capturing mechanisms than NetFlow v5 the ToS byte could be removed from the flow key by configuration. However, it is unlikely that such configurations will be used often since the ToS byte is of interest especially in enterprise networks where analyzing traffic based on traffic classes is a major use case for flow capturing. Therefore, it is very likely that path variant fields are part of the flow key and the problems highlighted in the following exist.

There are two corner cases concerning flows on an OPP path and a path variant field: either the field is changed in every packet (*complete change*) or the field is not changed in any packet (*no change*). If there is no change (Figure 4.10 on the left), all subflows will be unchanged. The extraction mechanism can detect this case by checking for $\Delta p = 0$ and $\Delta b = 0$. Then, the mechanism can directly match records based on the flow key and perform OWD calculation on a subflow basis. Contrary, with complete change the extraction mechanism can match flow records based on the path-invariant properties only (e.g., IP five tuple). Unambiguous subflow

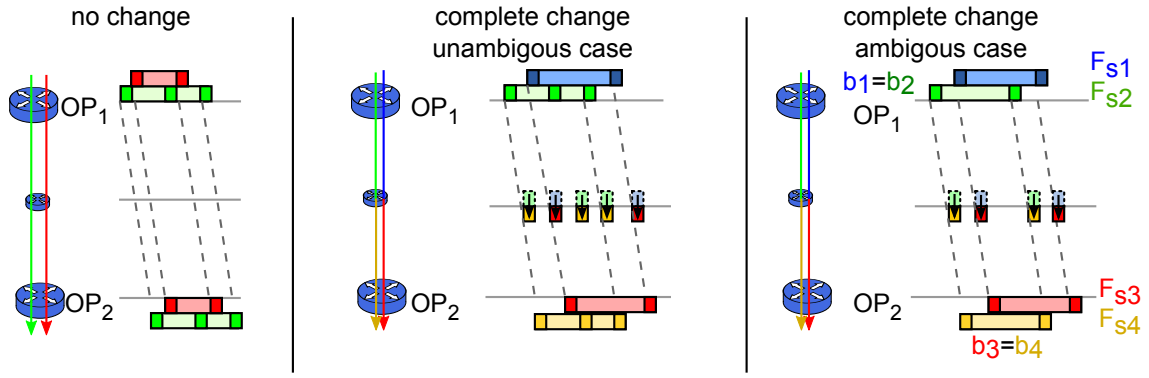


Figure 4.10: Effects of path variant fields in the flow key

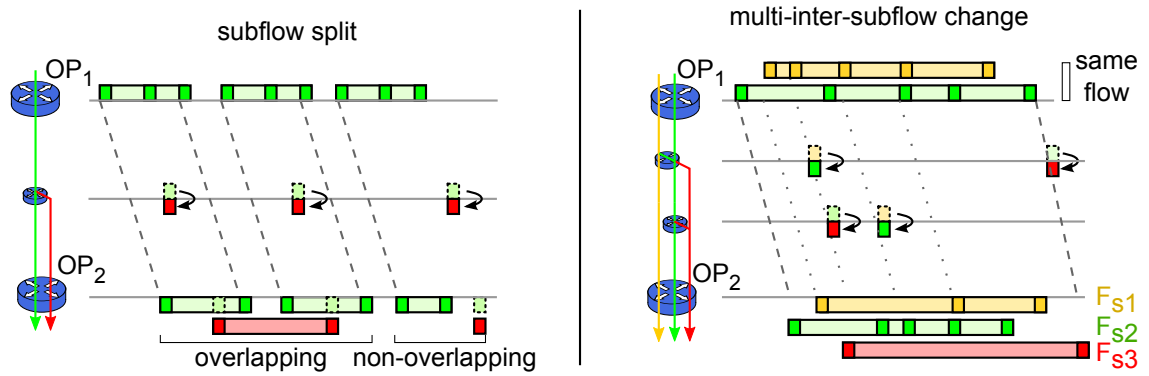


Figure 4.11: Effects of path variant fields in the flow key with partial subflow change

matching is possible as long as there is only one subflow at OP1 or the subflows differ in byte or packet count (Figure 4.10 middle).

As soon as there is more than one subflow for a flow and record byte count is the same[2] unambigous subflow matching is impossible. The extraction mechanism is unable to match a subflow record seen on OP1 to one of the subflow records of OP2 without taking additional knowledge into account (Figure 4.10 right). It cannot be inferred how the path variant fields changed on the path. E.g., in the presented example $\mathbf{F}_{s1}$ could map to $\mathbf{F}_{s3}$ or $\mathbf{F}_{s4}$.

Cases of subflow change that are between the two corner cases are cases with partial change, i.e., the path variant field is changed for a fraction of the packets of a flow only. Figure 4.11 shows on the left a simple case with one subflow where for some packets a path variant field changes (*subflow split*). In conjunction with temporal aggregation effects from flow capturing, this either leads to an overlapping case or an non-overlapping case. In the overlapping case, the original subflow is exported in two records that are overlapped in time by a record of the new subflow. In this example there are aligned subflow records, but this knowledge is not available for OWD calculation. Hence, the subflow records of OP2 have to be merged in terms of the path variant field as well as to be joined in time for OWD calculation with the records from OP1. In the non-overlapping case of subflow split, subflow records at OP2 have to be merged in any case.

[2] for example if each packet of Figure 4.10 has a length of 100 bytes

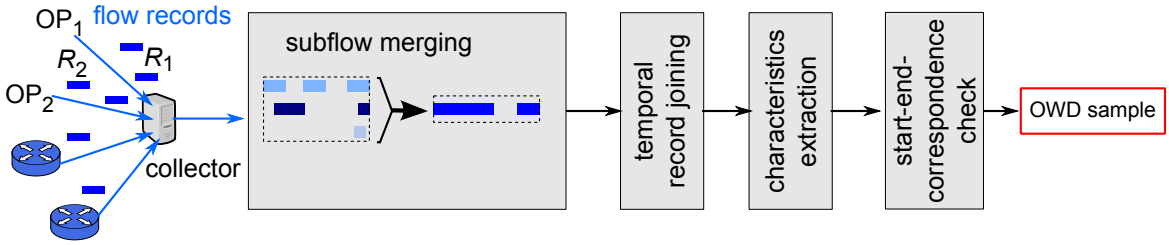


Figure 4.12: Extraction model extended with block for subflow merging

In the example on the right hand side of Figure 4.11, a flow consists of two subflows from the beginning. On the OPP path, there are two positions where the path variant field changes for a fraction of packets. This example illustrates that even if the packet and byte count for a single subflow is consistent across the OPP, SEC can be violated if the record and byte count for the other subflow records is inconsistent. The green subflow record in the right example of Figure 4.11 can have the same attribute properties on both OPs, however, SEC does not hold. This is due to the fact that it is impossible to decide whether $\mathbf{F}_{s3}$ consists out of the two packets that are now missing in $\mathbf{F}_{s1}$ or whether some packets moved from $\mathbf{F}_{s1}$ to $\mathbf{F}_{s2}$ and other packets from $\mathbf{F}_{s2}$ to $\mathbf{F}_{s3}$. Consequently, a check for $\Delta p = 0$ and $\Delta b = 0$ for each subflow is required, before calculating OWD on a per-subflow basis. This holds only under the assumption that path variant fields change monotonically, e.g., that policers change traffic classes only in the way that they get less priority. It can be assumed that this is typically the case.

As illustrated by the examples above, including path variant fields demands for attention with flow-based OWD calculation. Extraction of OWD on a subflow-basis is only reliable if there is no complete change or if packet and byte count consistency holds for every subflow. If this does not hold, subflow records have to be merged. As a conclusion, the extraction model has to be extended by a subflow merging block (see Figure 4.12) that considers the various cases and performs subflow merging if necessary . This can range from simple algorithms that always merge subflows and check for one-record-flows to more sophisticated algorithms that perform OWD extraction on a per-subflow basis.

4.3.2 Errors in Flow Capturing

The previous subsections dealt with effects from flow capturing while assuming correct flow data, i.e., no errors in the flow capturing process itself. However, there are several effects that lead to measurement errors and data loss, which have to be considered in record matching and OWD calculation. The following sections detail errors in the flow data itself (record fields) and errors in the data export (record loss).

4.3.2.1 Errors in Record Fields

This section considers errors in record fields, excluding timestamp fields. Such errors impact matching of records, especially if the OPs of an OPP have different error characteristics. In general, every field of a flow record could be subject to errors during metering and exporting. However, only certain error cases arise in reality and are presented here. These errors are packet count errors and byte count errors. Nevertheless, other errors and implausibilities might arise depending on other/new flow capturing equipment. Flow processing and OWD extraction mechanisms have to be robust against them and report such cases.

Packet count errors are errors where a packet of a flow passes an OP, but the counters in the flow record do not reflect this packet. This can happen, e.g., if the flow cache (see Section 3.1.3) is full and no flow entry can be created for the first packets of a flow, but for every other packet thereafter. Such an *accounting loss* is also reflected in other record values, i.e., the start time and packet count will also not reflect these packets.

In terms of byte count, some exporters report the Ethernet payload size, while other exporters report the IP packet size[3]. Which size to report as byte count is not exactly specified, even the NetFlow v9 RFC [RFC 3954] specifies the field `IN_BYTES` only as "Incoming counter with number of bytes associated with an IP Flow". This ambiguity results in errors if two exporters of an OPP handle the byte count differently. This is especially a problem for all packets smaller than 46 bytes, for which one exporter reports the IP packet size and the other the minimum Ethernet payload size (46 bytes). In the following we will use the term *byte count sensitivity threshold* for such a minimum reportable payload size, which will be denoted by k. Exporters reporting the Ethernet payload size have a byte count sensitivity threshold of $k = 46$ and will always report byte counts ≥ 46. This problem is called *byte count limitation* and the resulting error on an affected OPP *byte count error*. In the following the bounds of this error for the case where one OP has a byte count sensitivity threshold of $k > 0$ and the other OP has $k = 0$ are derived. $k = 0$ means that an OP reports the IP packet size. Depending on the packet size, the maximum byte count error *per record* is

$$e_{\mathrm{r,b}} = p \cdot (k - b_{\mathrm{p,min}}), \tag{4.1}$$

with $b_{\mathrm{p,min}}$ being the smallest IP packet size. According to its definition, the error is always positive and adds to or subtracts from $|\Delta b|$ depending on whether OP1 or OP2 has the byte count limitation. The smallest theoretically possible IP packet size is 20 bytes (no payload), thus the highest possible error per packet is 26 bytes. Evaluations of packet sizes in real networks (Section A.1) show that there are no such packets, but packets with 21 bytes can be found. Considering cases relevant for OWD calculation (UDP and TCP packets) there is still a considerable amount of packets with size below 46 bytes. Thus, the byte count error is severe and has to be considered in OWD calculation. This is especially important since Δb is an important indicator for effects that compromise SEC, but cannot be detected by evaluation of Δp alone.

[3]This error was discovered by the author and a master student [12]. Several NetFlow experts have not been aware of this problem.

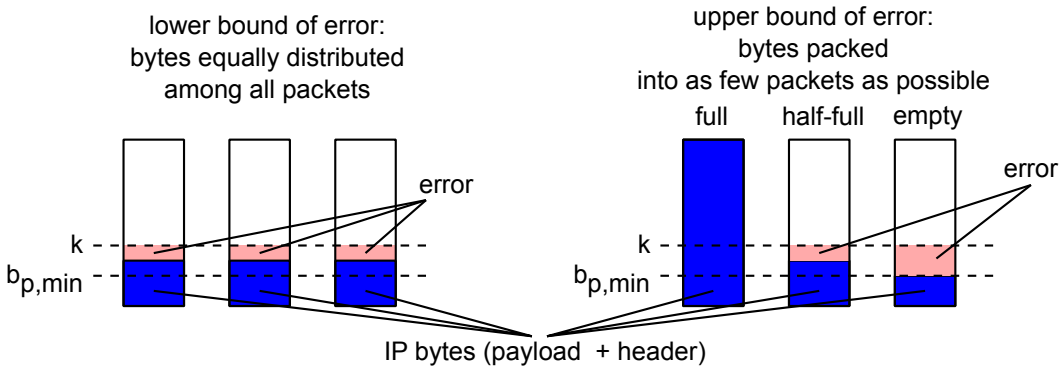


Figure 4.13: Lower and upper bound of byte count error

Whether there can be an error for a certain record at an OPP and the value of this error depend on distribution of the bytes among packets contained in a record. Since this distribution cannot be known from evaluating a flow record containing more than one packet, the real error cannot be calculated. Only lower bound and upper bounds can be given for two corner cases (see Figure 4.13). The lowest error in Figure 4.13 left results if the bytes are equally distributed among packets, i.e., the difference between the byte count of a packet and k is as small as possible and approaches zero if $p \cdot k$ bytes are present:

$$e_{\mathrm{r,b}_{min}} = max(p \cdot k - b, 0) \tag{4.2}$$

for b being the byte count from the IP packet sizes (of the OP that reports the lowest value).

The upper error bound results from the case where all bytes are packed in as few packets as possible (Figure 4.13 right). Then, every remaining packet will contribute the largest possible error. Obviously, this means that the maximum and minimum packet size ($b_{\mathrm{p,max}}$(MTU) and $b_{\mathrm{p,min}}$) have to be taken into consideration.

A first step for calculating the upper bound is considering how many completely filled packets can be present. This is the amount of MTU-size packets that can be filled with the amount of bytes not taken by the headers ($b - p \cdot b_{\mathrm{p,min}}$). The number of these packets is

$$n_{\mathrm{full}} = \left\lfloor \frac{b - p \cdot b_{\mathrm{p,min}}}{b_{\mathrm{p,max}} - b_{\mathrm{p,min}}} \right\rfloor \tag{4.3}$$

If $n_{\mathrm{full}} = p$ all packets are full and there will be no error. Furthermore, the highest error occurs if the maximum amount of packets contains the minimum number of bytes only. However, packets may exist that have more than the minimum bytes but less than the MTU (packet in the middle of Figure 4.13, right). The error contribution of this middle packet is

$$e_{\mathrm{r,b_{halffull}}} = max(k - b_{\mathrm{p,min}} - [(b - p \cdot b_{\mathrm{p,min}}) - n_{\mathrm{full}} \cdot (b_{\mathrm{p,max}} - b_{\mathrm{p,min}})], 0). \tag{4.4}$$

The remaining packets contain only headers and contribute with the maximum amount of error, i.e., the error of these empty packets is

$$e_{\mathrm{r,b_{empty}}} = (p - n_{\mathrm{full}} - 1) \cdot (k - b_{\mathrm{p,min}}) \tag{4.5}$$

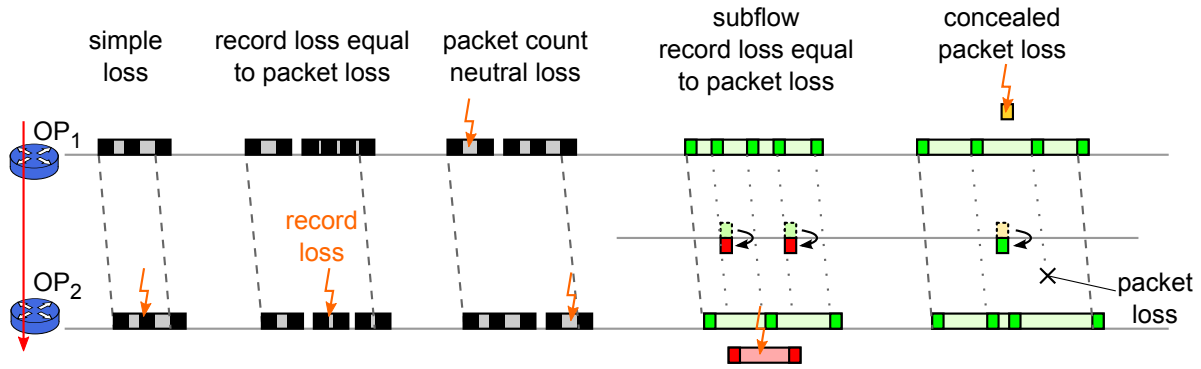


Figure 4.14: Record loss scenarios

Thus, the upper bound for the error is

$$e_{\mathrm{r,b}_{max}} = \begin{cases} 0 & n_{\mathrm{full}} = p \\ e_{\mathrm{r,b}_{\mathrm{halffull}}} + e_{\mathrm{r,b}_{\mathrm{empty}}} & otherwise \end{cases} \tag{4.6}$$

Using these calculations for upper and lower bounds, it is possible to determine whether Δb values for two records from an OPP path with the 46-bytes-error are reasonable or whether there was an effect that potentially breaks SEC and an OWD sample must not be created. As input parameter $b_{\mathrm{p,min}}$ can be chosen according to the transport protocol and the MTU ($b_{\mathrm{p,max}}$) has to be provided as input. In the extraction model (Figure 4.12), this has to be considered in the SEC-check stage.

As a side note, it is worth to mention that 46-byte-problem's impact vanishes for IPv6, where the IP header has already a size of 40 bytes and packets of UDP over IPv6 have at leas a length of 48 bytes.

4.3.2.2 Record Loss

For several reasons flow capturing mechanisms cannot guarantee that flow records for every flow are in the end available to the analysis application. First, the exporter itself might be under high load, such that it cannot send every flow record created in a packet due to overflow of internal queues. Second, flow records are sent using unreliable transport protocols (NetFlow uses UDP) and there is no application layer mechanism for compensating packet loss. Third, the collector itself might lose records due to overflow of packet queues in the receiving stage or due to other overload effects.

Collectors can quantify loss due to sequence numbers in flow data packets. For NetFlow v5, loss can be quantified on a per-record basis while NetFlow v9 only allows the quantification of lost NetFlow packets (see Section 3.2.4). Flow record loss rates can approach 2 % or even more in reality, thus this effect has high impact and must be considered in every flow data processing application.

While it is possible to quantify the loss per exporter, it is not possible to know which records have been actually lost. For extraction of network characteristics, especially packet loss and OWD, record loss has several impacts, as illustrated in Figure 4.14. In the case of *simple loss*, there is one record on OP1, but the only record from OP2 is lost. This means that no characteristics can be extracted at all. If there are several records per flow and one is lost, this cannot be distinguished from packet loss, especially if those records have short durations (second case in Figure 4.14). A similar case where only records from OP1 are lost, however, would correspond to negative packet loss by evaluating Δb, which is obviously an error (not shown in the figure). If records with the same number of packets from both OPs are lost, this is a *packet count neutral loss*. This case highlights again that for OWD extraction it is essential to take Δb into account in order to limit the possibility that SEC does not hold. However, SEC still cannot be guaranteed even if $\Delta b = 0$, since the byte count of the lost record and the lost packet can be the same.

In terms of subflows, a *subflow record loss equal to packet loss* can occur, as the fourth case of Figure 4.14 illustrates. Again, there is an ambiguity between packet loss and record loss. It is even possible, if not very probable, that packet loss is concealed by a lost subflow in conjunction with a subflow merge (e.g., DSCP reclassification) on the path (fifth case in Figure 4.14).

As a consequence from record loss, completely exact packet loss determination from NetFlow data is not possible. Also record loss makes checking Δb for OWD extraction even more important, even if there is some possibility that some cases where SEC does not hold cannot be detected. This will lead to a small number of wrong OWD samples. In terms of the extraction model (Figure 4.12), no further stages are necessary to detect record loss, since this is handled by the SEC-check stage. However, all mechanisms have to be designed in a way that they are robust against record loss and do not wait for flow records that might never arrive.

4.3.3 Summary of Effects and their Impact

The previous sections presented several effects and errors in flow capturing. For each of the effects, the impact on OWD calculation, i.e., whether Start-End-Correspondence (SEC) can be assured, has been considered and actions for detecting and handling effects have been given. This section gives a summary on these effects and draws conclusions for OWD extraction mechanisms that have to deal with these effects and errors.

Table 4.2 shows a summary of effects in the order they have been presented. For each effect, the impact on Δp and Δb for an OPP path where this error occurs is given. Additionally, the table shows which effect has impact on SEC or on the amount of OWD samples that can be extracted. The last column gives the action to take for detecting and/or handling or compensating the effect.

For each single effect listed in the table there is an appropriate action, such that it can be detected and SEC can be assured. However, this is not the case if effects are combined, which leads to ambiguities and a certain probability that the effects compensate for each other. In such cases the effects cannot be detected, however, SEC might be violated nevertheless and an OWD sample created from Δt_s or Δt_e is possibly wrong.

Table 4.2: Network effects and flow capturing effects summary

effect	**impact on**				**action**
	Δp	Δb	SEC	number of OWD samples	
packet loss	< 0	< 0	x	x	discard if $\Delta p \neq 0$
fragmentation	0	< 0	-	-	none
WAN opt., ingress	< 0	< 0	x	x	discard if $\Delta p \neq 0 \vee \Delta b \neq 0$
WAN opt., egress	> 0	> 0	x	x	discard if $\Delta p \neq 0 \vee \Delta b \neq 0$
load balancing before OP1, paths combined on OP2	> 0	> 0	x	x	discard affected records ($\Delta p \neq 0 \vee \Delta b \neq 0$)
load balancing on OPP path	< 0	< 0	x	x	discard affected records ($\Delta p \neq 0 \vee \Delta b \neq 0$)
temporal aggregation	0	0	-	x	temporal joining of records
path variant fields in flow key	0	0	x	x	subflow merging, per-subflow OWD calculation possible if $\Delta p \neq 0 \vee \Delta b \neq 0$ holds for the flow
packet not captured on OP1	> 0	> 0	x	x	discard if $\Delta p \neq 0$
packet not captured on OP2	< 0	< 0	x	x	discard if $\Delta p \neq 0$
byte count error	0	$\neq 0$	-	-	check for $\Delta b = 0$ affected
record lost from OP1	> 0	> 0	x	x	discard if $\Delta p \neq 0$
record lost from OP2	< 0	< 0	x	x	discard if $\Delta p \neq 0$

There are several effects that depend on the network or its configuration. Taking such knowledge into account could give advantage to OWD extraction algorithms. However, additional knowledge can never remove all ambiguities, as there will always be, e.g., packet loss, record loss, and packet capturing loss, even with a very low probability.

Since there always can be a combination of effects, this leads to two conclusions

1. Not all cases where SEC is violated can be detected. There is always a certain probability of OWD measurement errors from these effects.

2. All attributes present for improving the matching and SEC detection should be taken into account in order to reduce the possibility of errors.

The second point means that especially byte count should be taken into account. However, byte count itself is subject to errors and thus the extraction mechanism has to be adapted to each OPP's characteristic, i.e., it has to check for Δb within the error bounds of this OPP.

Flow capturing mechanisms that perform packet sampling (e.g., sFlow, sampled NetFlow, IPFIX PSAMP) have not been considered. In these cases there is a very low probability that records that describe the same packets at both OPs can be found (i.e., SEC holds), which is a requirement for OWD extraction. Coordinated sampling approaches for stateful flow capturing, such as trajectory sampling (see Section 3.3) approaches can solve this problem.

4.4 Timestamp Errors

After effects from flow capturing mechanisms that impact OWD calculation from flow records have been considered in the previous section, this section focuses on an essential fact: the accuracy of flow record timestamps. Based on a general model that covers all effects observed in flow data, these effects are systematically addressed in this section and consequences for system design are drawn in the last subsection.

As a general principle the highest priority is on quantification and compensation of systematic errors. The second highest priority is on quantifying the impact of random errors on accuracy. If both is not possible for certain errors (i.e., the error behavior cannot be captured by statistical means), flow record data is considered as blunder and must be removed (third highest priority), i.e., respective samples have to be detected and discarded.

4.4.1 Model of Timestamp Creation

The basic model of timestamp creation with NetFlow was introduced in Section 3.2.4. In the following, this basic model initially introduced in Figure 3.12 will be extended by properties of flow capturing devices that affect the accuracy of timestamps. Figure 4.15 shows this extended model highlighting the points where effects can affect the timestamp accuracy.

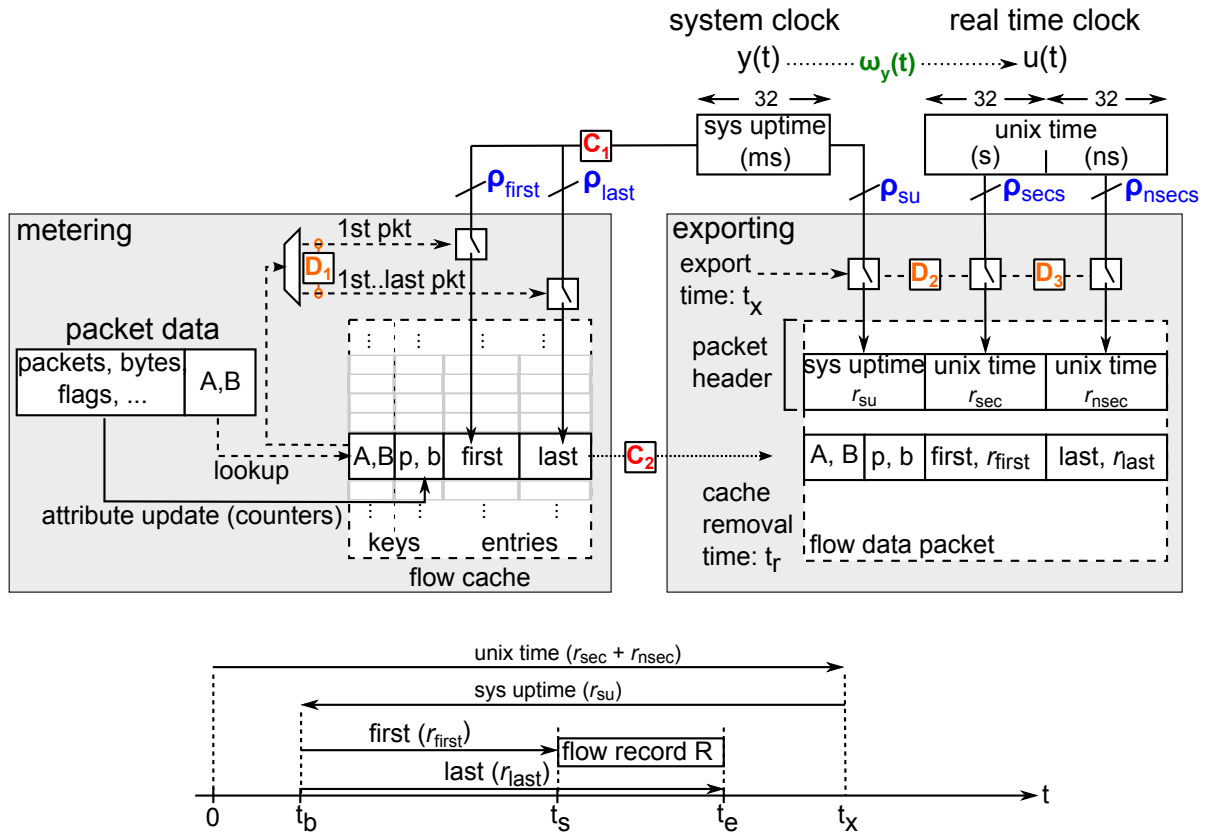


Figure 4.15: Timestamp creation model highlighting accuracy impacting points

This model is based on observations and measurements of NetFlow v5 data from a very large number of exporters. Thus, it reflects properties of lots of different flow capturing devices. Additionally, this model is kept general and covers all paths of timestamp values within a flow capturing device. Although it is unlikely that there are flow capturing devices for which this model does not fit, extensions and modifications for such flow capturing devices would be straightforward.

Figure 4.15 introduces symbols for clocks and timestamps that will be used in the following sections. There are two internal clocks, the system clock $y(t)$ and the RTC $u(t)$, which are used for creating timestamp values. The RTC is typically synchronized to a global reference time t, but in general always differs slightly from t. $u(t)$ models the difference from t. The system clock $y(t)$ is often a free running hardware clock based on a quartz oscillator. The values of these clocks are taken for creating the *raw timestamps* of the packet header for system uptime r_{su}, UNIX time seconds r_{sec}, and UNIX time nanoseconds r_{nsec}. Raw timestamps for flow start and end time are r_{first} and r_{last}. These raw timestamps are input for calculating the *record timestamps* export time (t_{x}), boot time (t_{b}), flow start time (t_{s}), and flow end time (t_{e}), as introduced in Section 3.2.4 and shown at the bottom of Figure 4.15.

The symbols highlighting accuracy impacting points are classified into four categories, each represented by a different symbol and different color in Figure 4.15:

- **The Skew $\omega_y(t)$ between exporter-internal clocks** describes the effect that internal clocks are unsynchronized among each other. Section 4.4.2.2 discusses these effects.

- **Limited resolutions ρ of raw timestamps** lower the resolution of record timestamp values reported although the protocol specifies 32 bit width for all timestamp fields. The limited resolution can result from low resolution of employed clocks as well as from cutting off bits from clock values. The latter could be motivated by memory requirements for saving memory in the flow cache. Errors resulting from limited resolutions are discussed in Section 4.4.3.

- **Internal conversions C_x of timestamp values** lead to effects that trailing unused bits of timestamp values are not cut off (zero), but changed to other values. This might not affect accuracy itself, but lead to effects that have to be considered in error distributions and is therefore also covered in Section 4.4.3.

- **Delays D_x in timestamp processing** cause inconsistencies in timestamp values like different timestamps for the same event. Section 4.4.4 will address these effects.

Whether and to which degree errors and effects are present depends on the implementation of the flow capturing device and its hardware components. If several metering processes or several exporting processes are present in a device, each may have its own characteristics. Consequently, the errors present in flow data depend not only on the device, but also on the observation point where packets are metered. Additionally, every timestamp value is obviously subject to overflows. This is not an error, but a property of the flow capturing mechanism itself.

4.4.2 Clock Synchronization

As introduced previously, there are two clocks for creating flow data possibly not synchronized to each other or to a global reference time. The first part of this section discusses issues of RTC synchronization and possibilities for removing systematic errors introduced by unsynchronized clocks. The second part considers the system clock and effects resulting from the skew between the system clock and the RTC as well as possibilities for correction of errors.

4.4.2.1 Real-time Clock Synchronization

The timestamps provided by the RTC $u(t)$ in flow data packets are input for calculating the absolute flow record timestamps. These timestamps are created from the value of the RTC at export time, i.e., $u(t_x)$. Thus, an offset of $\omega_u(t) = u(t) - t$ impacts all absolute timestamps calculated from t_x with the offset $\omega_u(t_x)$. As for OWD measurement flow timestamps from different flow capturing devices are considered, taking into account the accuracy of these timestamps and their synchronization is essential. The total impact of RTC offsets on OWD calculated from timestamps of an OPP is $\omega_{u,\mathrm{OP2}}(t_{x,OP2}) - \omega_{u,\mathrm{OP1}}(t_{x,OP1})$.

With hundreds of routers in large networks, keeping track of every router's clock and its configuration with respect to time synchronization is a challenging task. Thus, it is always likely that some clocks run unsynchronized for certain time and that this is not noticed by administrators

or network management tools. Multi-domain networks complicate the problem of synchronized clocks, as every domain might rely on its own synchronization mechanism. Especially in enterprise networks, flow data is also exported from routers that do not belong to the enterprise, but to the a carrier, as it is, e.g., the case for CE routers of an MPLS VPN (see Section 2.1.3). Administrators of the enterprise networks often do not have permissions to change configurations on CE routers, thus keeping the clocks of these routers synchronized is up to the carrier. Flow data collected from a global enterprise network showed that unsynchronized clocks of CE routers are a severe problem. Several router clocks' offset was in the range of several minutes. Evaluations in [3] also show that RTC skew differs between exporters and that there is almost no drift .

Due to the possibility of unsynchronized clocks, it is important to check whether clocks are synchronized before starting OWD extraction. This means that it has to be checked whether $u(t)$ reported by each exporter is consistent with reference time or whether it shows offset, skew, or drift. Possible actions if such effects are detected are either dropping all data exported from a respective exporter or compensating for such effects. In Section 2.3.4 related work for skew and offset estimation has been presented. Based on these, offset and skew can be estimated as $\omega_u(t) = \omega_{0,u} + \varphi_{0,u} \cdot t$. After checking that drift is negligible, clock synchronization errors can be compensated before extracting the OWD.

4.4.2.2 Skew of Internal Clocks

The model of Figure 4.15 shows that there are two independent clocks in an exporter: the RTC that is possibly synchronized to UTC and the system clock. Evaluations of NetFlow data show that these two clocks are often not synchronized to each other and that the system clock is most likely a free running quartz oscillator. This leads to a time-dependent offset, i.e., a skew called *system clock skew* denoted by $\omega_y(t)$. This effect was found by the author and published in [6]. Trammell et al. also reported this effect in [74]. This system clock skew adds an *export delay dependent error* to flow record timestamps.

The system clock skew $\omega_y(t)$ is a characteristic value, modeled as the skew between the system clock y(t) and the RTC u(t). It is not modeled as skew to t, and can therefore be calculated directly from NetFlow raw timestamp fields that report y(t) and u(t). Evaluations from NetFlow data show that the system clock skew mean is constant, i.e., there is no drift, but only little noise. The system clock skew is therefore denoted by the constant $\varphi_{0,y}$. The overall system clock offset is hence

$$\omega_y(t) = y(t) - u(t) = \omega_{0,y} + \varphi_{0,y} \cdot u(t). \tag{4.7}$$

If there are several flow capturing processes in an exporter (see Section 3.3), each might have its own system clock and hence different values for the system clock skew. Thus, the system clock skew has to be considered as a characteristic *per Observation Point*. The system clock skew puts an error on timestamps, especially it impacts the boot time t_b calculated from the RTC and system time. This leads to continuous *boot time shift*, i.e., the reported t_b changes over time:

$$t_\mathrm{b}(t) = u(t) - y(t) = -\omega_y(t) \tag{4.8}$$

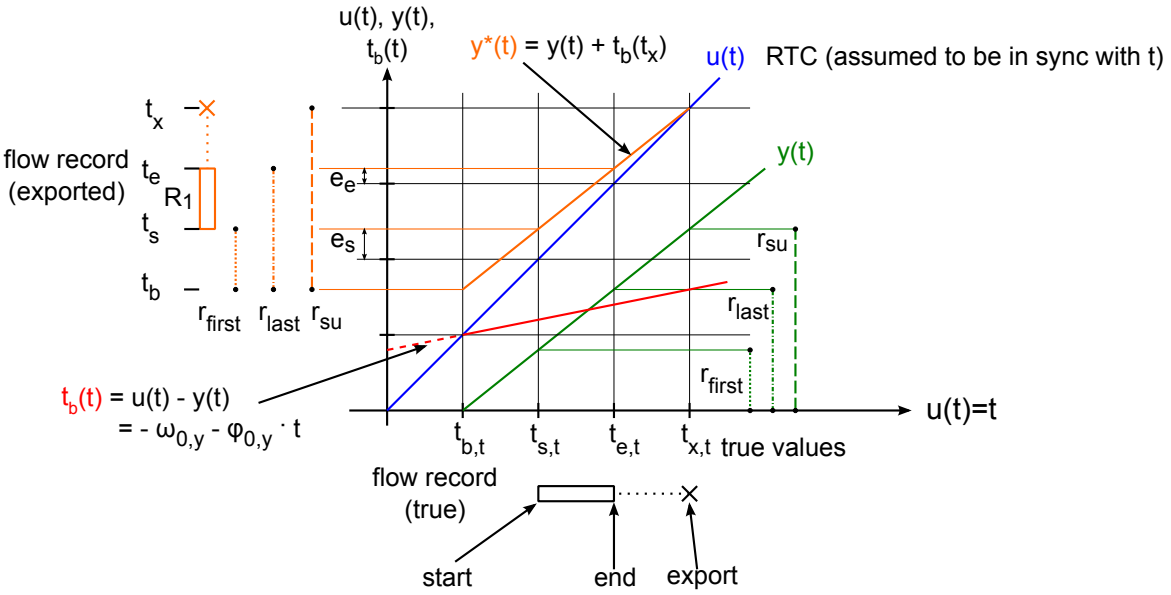


Figure 4.16: Impact of system time skew on record start and end timestamps

As t_s and t_e are calculated from t_b (see Figure 4.15), $\omega_y(t)$ also impacts these timestamps. The impact on these timestamps depends on the export delay, i.e. time interval between t_s, t_e and t_x, which have been defined in Section 3.1.3 as δ_s and δ_e.

Figure 4.16 shows the relations between the true timestamps ($t_{b,t}$, $t_{s,t}$, $t_{e,t}$) and the timestamps reported in flow records due to system clock skew (t_b, t_s, t_e). t_x is not affected by this effect, thus $t_x = t_{x,t}$ holds. In order to simplify illustrations, $u(t)$ is assumed to be synchronized to the true time t and the following considerations are based on $t_b(t)$. The x-axis in Figure 4.16 indicates the true timestamps for the exemplary flow record depicted below. From the depicted $y(t)$ (green line) the raw timestamps are created (orange symbols on the lefthand side). In this example, there is a negative system clock skew, i.e., $\varphi_{0,y} < 0$ and thus a positive boot time shift $t_b(t)$. The orange line indicates $y^*(t)$, which is $y(t)$ shifted up in a way that it intersects with $u(t)$ at t_x:

$$y^*(t) = y(t) + t_b(t_x) \tag{4.9}$$

The line of $y^*(t)$ in Figure 4.16 allows to read the reported erroneous timestamps directly: they are the y-values of $y^*(t)$ at the points where the x-values are the real timestamps.

The corresponding raw timestamps are illustrated as well on the left in Figure 4.16. The reported timestamps result in a reported flow record (left hand side) that is distorted compared to the true flow record. This is indicated by the error of the start timestamp $e_s = t_s - t_{s,t}$ and end timestamp $e_e = t_e - t_{e,t}$.

The error of the start timestamp can be calculated by taking the export delay δ_s into account and the impact of the offset accumulated in this interval:

$$e_s = (t_{x,t} - t_{s,t}) \cdot (-\varphi_{0,y}) = \delta_s \cdot (-\varphi_{0,y}) \tag{4.10}$$

Considering this error as difference between $y^*(t)$ and $u(t)$, we can derive another equation:

$$e_{\mathrm{s}} = y^*(t_{\mathrm{s,t}}) - u(t_{\mathrm{s,t}}) = y^*(t_{\mathrm{s,t}}) - t_{\mathrm{s,t}} \tag{4.11}$$

Using equations (4.8), (4.9), (4.10), and (4.11), we can derive the equation for export delay dependency error correction:

$$t_{\mathrm{s,t}} = t_{\mathrm{x}} + \frac{r_{\mathrm{first}} - r_{\mathrm{su}}}{\varphi_{0,y} + 1}. \tag{4.12}$$

Accordingly for the end timestamp, we get

$$t_{\mathrm{e,t}} = t_{\mathrm{x}} + \frac{r_{\mathrm{last}} - r_{\mathrm{su}}}{\varphi_{0,y} + 1}. \tag{4.13}$$

Considering OWD measurements, the error depends on the system clock skew of the observation points. More specifically, it depends on the system clock skew *difference* between OP1 and OP2 of the OPP: $\Delta\varphi_{0,y_{\mathrm{OPP}}} = \varphi_{0,y_{\mathrm{OP1}}} - \varphi_{0,y_{\mathrm{OP2}}}$. The error introduced by $\Delta\varphi_{0,y_{\mathrm{OPP}}}$ depends on the export delay, and thus eventually on the export timers. If an exemplary inactive timeout of 10 s is considered, the end timestamp of a record will experience an error of 1 ms for $\Delta\varphi_{0,y_{\mathrm{OPP}}} = 1{\cdot}10^{-4}$. As the evaluations in Section 7.2.4 show, such $\Delta\varphi_{0,y}$ values are common in NetFlow data. The accuracy of standard quartz oscillators is in this range of +/- 100 ppm (+/- $1{\cdot}10^{-4}$). For long records that have, e.g., been exported due to active timeout of 60 s, an error of 1 ms will already exist for $\Delta\varphi_{0,y_{\mathrm{OPP}}} = 16.7{\cdot}10^{-6}$.

These considerations show that system clock skew can cause errors in OWD measurement of several milliseconds. However, this error can be compensated if $\varphi_{0,y}$ is known. This error is independent of other synchronization errors. If $u(t)$ is not synchronized to t, this is simply an additive error as discussed previously.

4.4.3 Timestamp Resolution and Effects

The timestamp creation model presented in Section 4.4.1 contains parameters describing limited resolutions of raw timestamps. These resolution parameters, as well as how resolution limitation causes systematic and random errors is evaluated in this section. First, the typical resolution characteristics, i.e., typical parameters for flow data exporters, are considered. In the second part, systematic errors caused by resolution characteristics of single OPs as well as their impact on OPPs with exporters that have different resolution characteristics are detailed. The third part deals with random errors and their distributions.

4.4.3.1 Typical Resolution Characteristics

There are several effects concerning timestamp resolution in flow data, i.e., the real resolution of timestamps is lower than the resolution that could be reported by the flow capturing protocol.

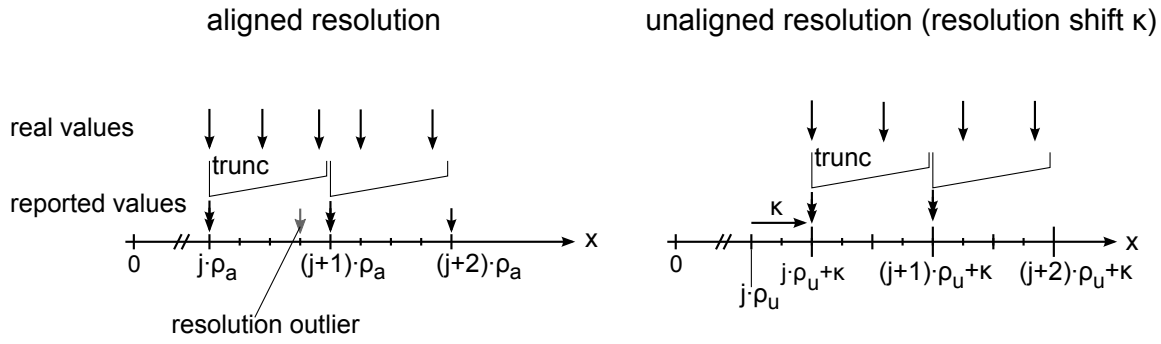


Figure 4.17: Resolution terminology

E.g., the raw timestamp r_{first} field can report a millisecond value in NetFlow v5. However, it has a resolution of 4 ms on some exporters only. These effects are represented by the ρ-parameters associated with each raw timestamp in the timestamp creation model. In the following, this model is detailed based on resolution effects that can be found in flow data. After the definition of basic terms, the impact of the resolution on record timestamps is derived. First, the basic case without taking internal timestamp conversion into account is considered. Second, unaligned resolutions resulting from conversion are presented. The parameters of the model and the examples presented apply to NetFlow v5. However, it can also be adapted and applied to timestamp values of other flow capturing protocols and mechanisms.

The ρ-parameters describe the resolution of timestamp values and are denoted using the same index as the timestamp value as well as the same units. The ρ-notation is used for raw timestamps as well as for record timestamps (e.g., ρ_{first} denotes the resolution in milliseconds of the timestamp r_{first}, ρ_{s} denotes the resolution of t_{s} accordingly). Two resolution classes are distinguished: *aligned* resolution ρ_{a} and *unaligned* resolution ρ_{u}. If neither the index a nor u is given, alignment is not specified. The *aligned resolution* of a timestamp x is defined to be the largest value for which after division by ρ_{a} the remainder is zero for the *majority* of timestamps. This means that timestamp values are truncated due to limited resolution by the *trunc* function, defined as:

$$\text{trunc}(x, \rho_{\text{a}}) = x - x \bmod \rho_{\text{a}} \tag{4.14}$$

From this definition it results that timestamps are *aligned* to the resolutions ρ, thus called *aligned resolution*, illustrated on the left in Figure 4.17. Especially for resolution parameters to be expected from limitations in digital devices, this means that the last bits are zero (cut off). Effects resulting from limited resolution are modeled as truncation of the real timestamp value (*truncation assumption*). This results straightforwardly from typical implementations in digital devices, where cutting off bits is plausible, but rounding up to the next aligned number is not. Resolution is defined based on the majority of timestamps, since there are sometimes effects or errors that lead to situations where a small number of samples has apparently a higher resolution than the resolution of the device (*resolution outliers*, see Figure 4.17 on the left).

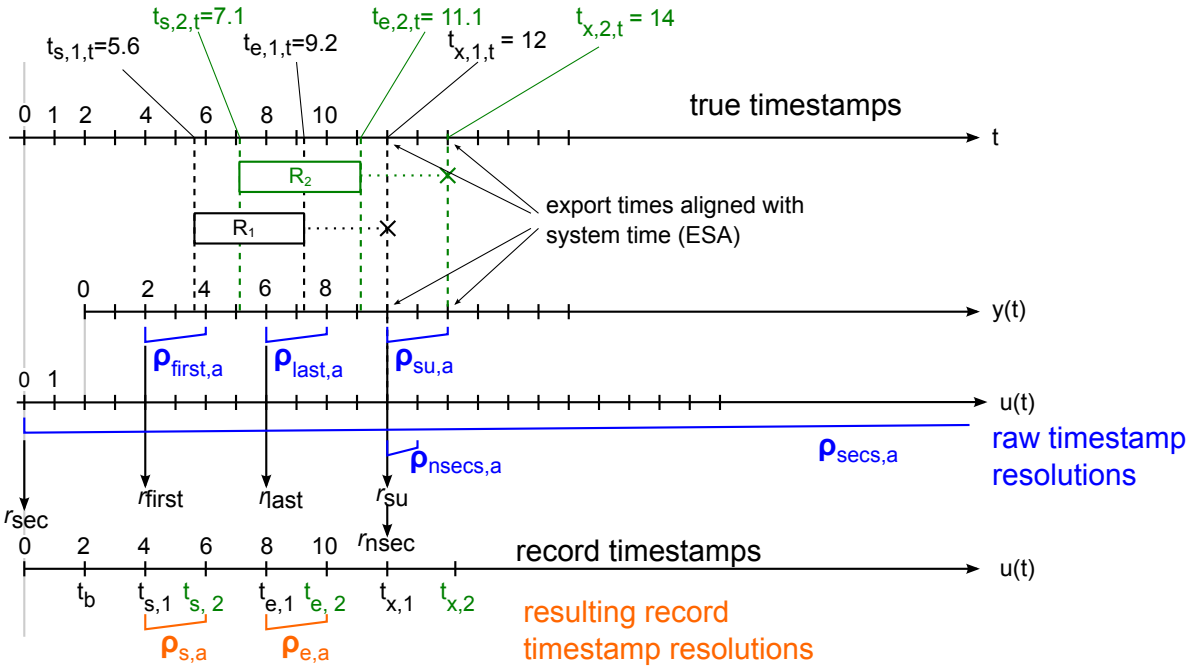


Figure 4.18: Illustration of timestamp resolutions (aligned timestamps)

In flow capturing devices timestamps are not always aligned due to different internal clocks and conversion of measurement values. For modeling such effects, it is important to consider *unaligned resolution*, where the resolution is shifted by a resolution alignment offset κ, as depicted on the right of Figure 4.17. In this case, the remainder of $\frac{x}{\rho_u}$ is not zero, but the remainder of $\frac{x-\kappa}{\rho_u}$ is zero, i.e., after applying the trunc-function, κ is added to the value by the metering process. If there is such an offset, *unaligned resolution* is present.

In general, all ρ-parameters impact the record timestamps reported by an exporter. This is illustrated in the example of Figure 4.18 that shows the true time t as well as the exporter internal clocks. For better understanding, we consider the effects based on exemplary values. All values are given in milliseconds and it is assumed that y(t) and u(t) are not subject to clock skew, neither against each other, nor to t. Furthermore, in this section, we assume that record export times are aligned with the system clock, i.e., ρ_{su} does not have impact on the reported export time, but system uptime reported equals the system uptime at export time. This property is called *Export-Systime-Alignment* (ESA) and its impact is detailed in the next section. In the example the offset between y(t) and u(t) is $t_b = 2$ ms. We consider two flow records: $\mathbf{R}_1$ describes a flow from $t_{s,1,t} = 5.6$ ms to $t_{e,1,t} = 9.2$ ms, and $\mathbf{R}_2$ from $t_{s,2,t} = 7.1$ ms to $t_{e,2,t} = 11.1$ ms (true timestamps). Furthermore, we assume for this example $\rho_{first} = 2$ ms, $\rho_{last} = 2$ ms, and $\rho_{su} = 2$ ms as well as $\rho_{sec} = 1$ s and $\rho_{nsec} = 1{\cdot}10^6$ ns.

The resolution values and resulting timestamps for $\mathbf{R}_1$ are depicted in Figure 4.18. Due to the limited resolution of start and end timestamp, $r_{first} = 2$ ms and $r_{last} = 6$ ms result. Concerning the export timestamp, $r_{su} = 10$ ms results due to ESA. Furthermore, $u(t_x)$ is reported as $r_{sec} = 0$ s and $r_{nsec} = 12{\cdot}10^6$ ns. This results in the record timestamps reported to be $t_{s,1} = 4$ ms and $t_{e,1} = 8$ ms. For $\mathbf{R}_2$ we get $t_{s,2} = 6$ ms and $t_{e,2} = 10$ ms (raw timestamps not shown in detail).

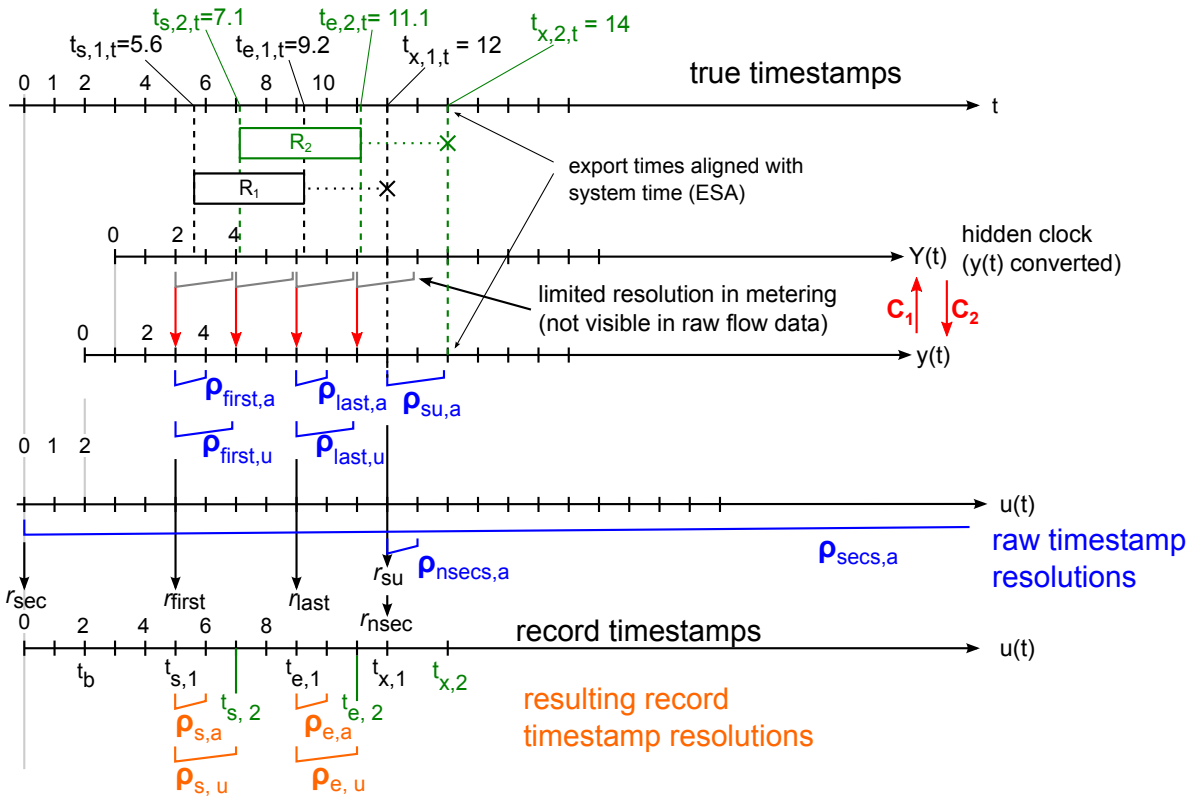


Figure 4.19: Illustration of timestamp resolutions, including conversion effects that lead to unaligned timestamps

From the start and end timestamps of the two records, we can see that the record timestamp resolution is $\rho_s = 2$ ms and $\rho_e = 2$ ms. In this case the timestamps are aligned.

Note that in this simple case record timestamp resolutions could have been directly calculated from ρ_{first} and ρ_{last}. However, this is a result from the values of ρ_{su}, ρ_{sec}, and ρ_{sec} that have been chosen in a way to keep the example simple.

The next example (Figure 4.19) shows how internal timestamp conversion leads to unaligned timestamp resolution. This example uses the same exemplary records and resolution parameters as the previous example. In contrast to the previous example, there is now an internal clock in the metering block, which has an offset against $y(t)$. Its timestamps are converted to $y(t)$ values before being written into flow records. Thus, timestamp values of this internal clock are not visible to the outside, but hidden. In the example, the internal clock has an offset of 1 ms to $y(t)$. The timestamp values are therefore not aligned to $y(t)$ according to the resolution ρ_s, but there is an unaligned resolution of start and end timestamps. Record timestamps reported of $\mathbf{R}_1$ are $t_{s,1} = 5$ ms and $t_{e,1} = 9$ ms, as well as, $t_{s,2} = 7$ ms and $t_{e,2} = 11$ ms for $\mathbf{R}_2$. As shown by these values, record timestamps exhibit resolution shift, i.e., $\rho_{s,a} = 1$ ms, but $\rho_{s,u} = 2$ ms.

In NetFlow data of production networks, several exporters that perform such conversion as previously mentioned can be found. Furthermore, there is even a small skew of internal clocks, which leads to the effect that the resolution alignment offset κ is time dependent, i.e., $\kappa(t)$. As a result, $x \bmod \rho$ is not constant but describes a sawtooth.

Table 4.3 lists exemplary resolution values extracted from flow data for different OP types. The resolution values are given per OP type, since flow capturing characteristics can differ for different OPs of the same exporter, i.e., a single exporter can have OPs of different OP type. Since OWD calculation in this work considers ingress flow capturing only (see Section 3.3), it is sufficient to distinguish the flow capturing characteristics per OP (using the definition of OP that contains the input interface identifier). For all OP types, ρ_{sec} has a resolution of one second, and ρ_{nsec} is 15,258 ns on all exporters. It can be assumed that the latter is the raw resolution of the RTC, since clock components often divide one second into 2^{16} ticks ($\frac{1\text{ S}}{2^{16}} = 15,258$ ns).

On OPs of type 1, ρ_{su}, ρ_{s}, and ρ_{e} are four (aligned), while for type 2 only ρ_{su} has a limited aligned resolution. OPs of type 3 do not have a limited resolutions, except in ρ_{nsec}. For type 2 OPs, $\rho_{\text{s,a}}$ and $\rho_{\text{e,a}}$ differ from $\rho_{\text{s,u}}$ and $\rho_{\text{e,u}}$. This is an OP that shows the time-dependent resolution shift behavior (saw tooth in lower six bits of r_{first} and r_{last}). The last column of Table 4.3 shows the resolution ρ_{d} of the record duration d, which is $\rho_{\text{d}} = 4$ ms for type 1 and $\rho_{\text{d}} = 64$ ms for type 2, both unaligned. The latter shows that the hidden clock has exactly a resolution of 64 ms and its start and end timestamps are converted in the model at one time instant (C_1) into r_{first}, and r_{last}. OPs of type 1 are, e.g., Cisco 7200 series and type 2 are, e.g., Cisco 6500 series performing hardware-based (MLS) NetFlow.

4.4.3.2 Systematic Errors from Limited Timestamp Resolution

OWD is calculated from record timestamps of different OPs, and the record timestamps itself are calculated from raw timestamp values. Hence, the limited resolution in raw timestamps leads to errors in record timestamps, which lead to errors in OWD calculation. Every raw timestamp represents an input quantity that is used to calculate the output quantity (OWD) according to the measurement function. This means that the errors caused by resolution limitations propagate according to the laws of error propagation, as introduced in Section 2.2.2. In this section, the systematic errors and their propagation is considered. The next section deals with random errors.

Figure 4.20 shows the measurement functions for obtaining the record timestamps and finally the OWD samples from raw timestamps. As introduced in Section 4.4.1, there are five raw timestamps for each flow record, which are the input quantities. The first step is calculating t_{b}, which is then an input quantity for calculating t_{s} and t_{e}. Finally, Δt_{s} and Δt_{e} are calculated as potential OWD samples. It is important to note in terms of systematic errors that at two points of the model values are subtracted from each other, while at other points, values are added.

Table 4.3: Typical timestamp resolutions

	Raw timestamp resolutions					Record timestamp resolutions				
OP type	$\rho_{\text{sec,a}}$	$\rho_{\text{nsec,a}}$	$\rho_{\text{su,a}}$	$\rho_{\text{first,a}}$	$\rho_{\text{last,a}}$	$\rho_{\text{s,a}}$	$\rho_{\text{s,u}}$	$\rho_{\text{e,a}}$	$\rho_{\text{e,u}}$	ρ_{d}
1	1 s	15,258 ns	4 ms	4 ms	4 ms	4 ms	4 ms	4 ms	4 ms	4 ms
2	1 s	15,258 ns	4 ms	1 ms	1 ms	1 ms	64 ms	1 ms	64 ms	64 ms
3	1 s	15,258 ns	1 ms	1 ms	1 ms	1 ms	1 ms	1 ms	1 ms	1 ms

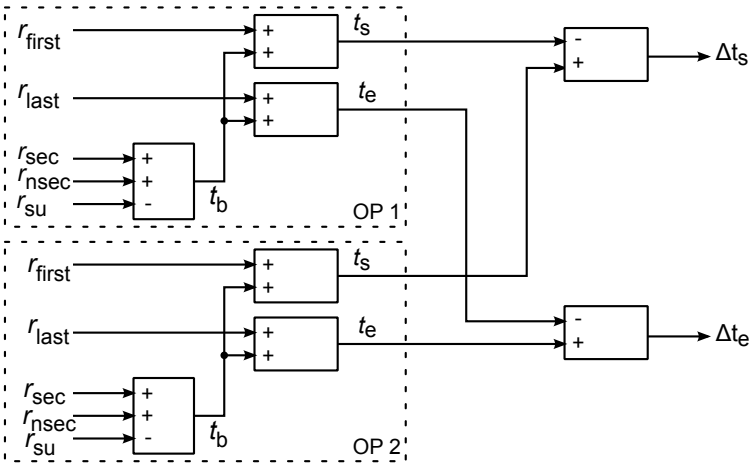


Figure 4.20: Measurement functions for OWD calculation

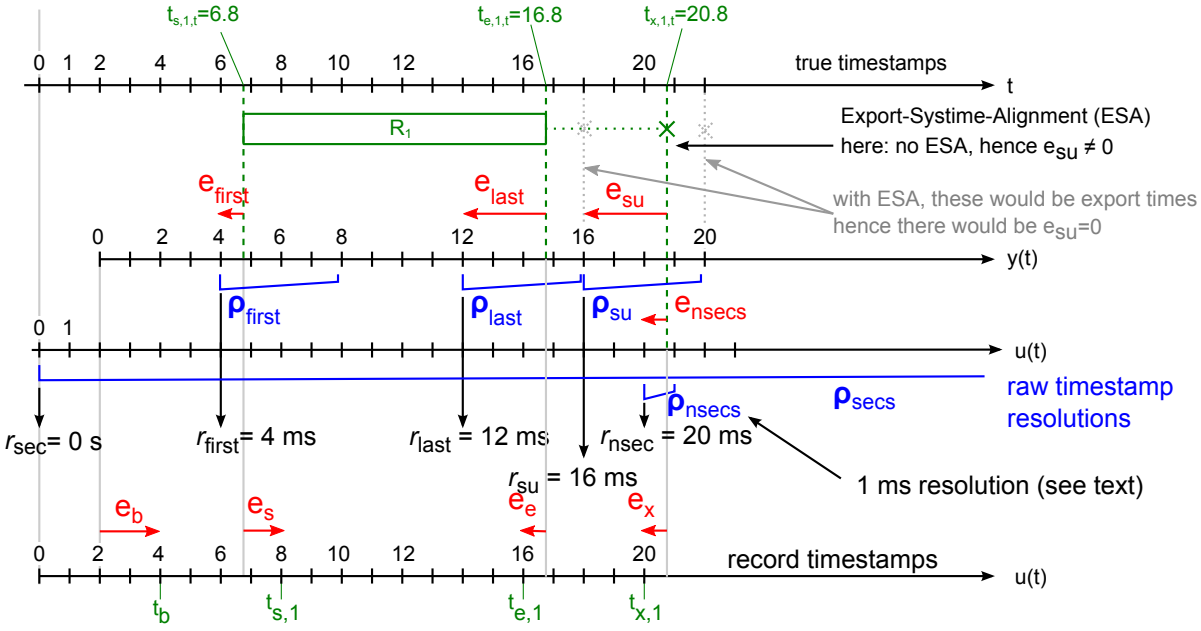


Figure 4.21: Errors resulting from resolution effects. Exemplary values of OP type 1

Figure 4.21 illustrates how limited resolution leads to errors in raw and record timestamps. For this example, resolution values for OPs of type 1 (see Table 4.3) have been chosen. As in the previous examples $y(t)$ and $u(t)$ are aligned (no skew) and an exporter without internal conversion is considered. We will consider one record with $t_{s,t} = 6.8$ ms, $t_{e,t} = 17.8$ ms that is exported at $t_{x,t} = 20.8$ ms.

Due to the limited resolutions ρ_{first} and ρ_{last}, r_{first} and r_{last} report smaller values than the true values and errors of $e_{first} = -0.8$ ms and $e_{last} = -2.8$ ms result from this effect. The system uptime at export time can also be subject to resolution effects. Concerning the system uptime, it depends on whether record export times are aligned with system uptime. If this property

(*Export-Systime-Alignment* (ESA), introduced in the previous section) holds, records are always exported at times $y(t_x)$ that are whole multiples of ρ_{su}. Thus, here will be no error caused by ρ_{su} ($e_{su} = 0$). If ESA does not hold, ρ_{su} causes a systematic error, as shown in the example of Figure 4.21. Due to the absence of ESA, $e_{su} = -2.8$ ms results for the exemplary values.

The raw export timestamp t_x are represented by two values, r_{sec}, which is 0 s in our example and r_{nsec}. The r_{nsec} value has special characteristics since it is reported in nanoseconds and therefore in much higher granularity than the other values. However, its resolution is typically $15,258$ ns, which is still higher than the resolution of the other timestamp, but 1 ms is no whole multiple of this value (least common multiple is 7.629 s). Thus, it is rare that its value is aligned with r_{su}. Software that processes NetFlow data (e.g., nfDump [101] or flow-tools [99]) always truncates the r_{nsec} values to milliseconds during processing (see Note 2 in Figure 4.21), i.e., this corresponds to $\rho_{nsec} = 1$ms.

From the measurement function, it results that $e_b = e_{nsec} - e_{su}$. In the example this is a positive value: $e_b = 2$ ms. The errors for start and end time eventually contain the errors for the boot timestamp as well as the errors of r_{first} and r_{last}: $e_s = e_{nsec} - e_{su} + e_{first}$ (adds up to 1.2 ms in the example) and $e_e = e_{nsec} - e_{su} + e_{last}$ (-0.8 ms in the example).

As shown in the example, the raw timestamps always underestimate the real value due to the truncation assumption. We consider any of the raw timestamp values r_a and use the index a for this purpose. If we switch from errors e_a of single value r_a to the general estimated systematic error (bias) β_a, we can define $\beta_a = -\frac{\rho_a}{2}$.

The bias of the export timestamp value is determined by the resolution of the nanoseconds-part of the unix timestamp reported r_{nsec}. This is due to the fact that $r_{sec} = 1\,s$ in all observed cases, which has to be double-checked for each exporter, of course. Taking the calculation of record timestamps into account, the overall bias values of start and end timestamps are

$$\beta_s = \beta_{nsec} - \beta_{su} + \beta_{first} \tag{4.15}$$

and

$$\beta_e = \beta_{nsec} - \beta_{su} + \beta_{last}. \tag{4.16}$$

If ESA holds, ρ_{su} does not cause a systematic error, hence, $\beta_{su} = 0$ in this case. The bias impacts timestamp differences between two exporters according to the measurement functions:

$$\beta_{\Delta t_s} = \beta_{s,OP2} - \beta_{s,OP1} \tag{4.17}$$

$$\beta_{\Delta t_e} = \beta_{e,OP2} - \beta_{e,OP1} \tag{4.18}$$

Since this systematic error adds up according to the measurement functions, it can also happen that resolution effects compensate each other in terms of systematic errors. This is for example the case if $\rho_{first} = \rho_{su}$ and ESA does not hold, as it is the case for the example in Figure 4.21. If

the bias at both OPs is identical, the systematic errors of timestamp differences will also compensate each other. Details on effects from systematic errors and a study on ESA is presented in Section B.2 based on lab measurements.

For all considerations on systematic errors it does not matter whether resolutions are aligned or unaligned, since each resolution limitation truncates timestamps and thus leads to a bias. If knowledge on timestamp resolutions of OPs is available, the impact of systematic errors on OWD samples can be quantified and compensated.

4.4.3.3 Random Errors from Limited Resolution

In addition to systematic errors, limited timestamp resolution also causes random errors, which propagate according to the measurement functions, as introduced in Section 2.2.2. The random error caused by limited resolution ρ of an input quantity results in a uniform PDF of width ρ. We consider the random error only, i.e., assume that the systematic error has been removed. The distribution is uniform

$$f_x(t) = \begin{cases} \frac{1}{\rho_\mathrm{x}} & -\frac{\rho_\mathrm{x}}{2} < x < \frac{\rho_\mathrm{x}}{2} \\ 0 & otherwise \end{cases} \tag{4.19}$$

with zero mean and the standard deviation

$$\sigma_\mathrm{x} = \sqrt{\frac{\rho_\mathrm{x}^2}{12}}. \tag{4.20}$$

Due to error propagation, the output quantities are affected accordingly. If the random errors of the input quantities are independent of each other, the resulting PDF of the output quantity (OWD) random error is the convolution of the input quantities' random error PDF, as introduced in Section 2.2.2.

In case of convolution of PDFs, the ranges of the distributions are added as well as the variances sum up and therefore standard deviation of the OWD samples will be

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} \rho_i^2}{12}}. \tag{4.21}$$

Given the resolution parameters introduced before, the standard deviation of the random errors are

$$\sigma_{\Delta \mathrm{t_s}} = \sqrt{\frac{{\rho_\mathrm{nsec}}^2 + {\rho_\mathrm{su}}^2 + {\rho_\mathrm{first}}^2}{12}} \tag{4.22}$$

and

$$\sigma_{\Delta \mathrm{t_e}} = \sqrt{\frac{{\rho_\mathrm{nsec}}^2 + {\rho_\mathrm{su}}^2 + {\rho_\mathrm{last}}^2}{12}}. \tag{4.23}$$

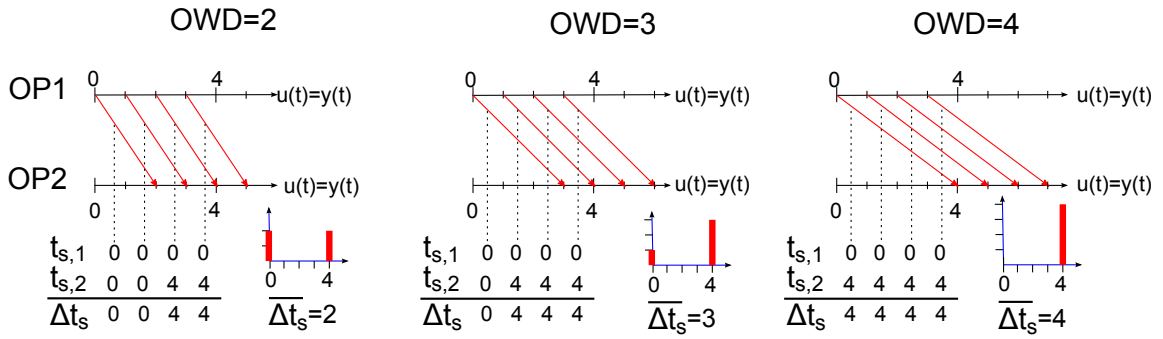


Figure 4.22: Impact of OWD on error distribution

If ESA holds, ρ_{su} will not cause any random error and has to be omitted.

As mentioned above, convolution of PDFs for achieving the overall random error is only possible, if the input quantities are independent. While the packet arrival times of packet multiplex streams are in general independent of each other, flow-based OWD calculation input quantities might be, however, *not independent*. If the clocks of the two OPs are well synchronized and OWD is constant, effects occur that lead to a discretized PDF for random error. This effect also occurs if the OWD varies in a small range only. In the extreme case where two OPs of an OPP with the same resolution reside in the same device, this effect leads always to a bimodal PDF, as the laboratory measurements in Section B.2 show. The effect that leads to discretized PDFs is illustrated in the following.

First, the case of constant delay and completely synchronized clocks of metering and exporting processes is considered. The resulting distributions and how they result from timestamp values for three different OWD values are shown in Figure 4.22. This example assumes a scenario with $\rho_{\mathrm{first}} = 4$ ms, $\omega_{0,y} = 0$, $\varphi_{0,y} = 0$. We consider record start times only and show only the RTC, not the system internal clock for sake of simplicity. In order to average out random errors, we consider four samples distributed equally across an interval corresponding to the timestamp resolution. The record timestamps obtained and Δt_{s} values from each OP are illustrated in a table below and illustrated in a histogram. In this example it can be clearly seen how the bimodal distribution results from the limited timestamp resolution and that for OWD values being a whole multiple of the resolution, there is an *exact alignment* and a single peak results. This effect has also been validated using a laboratory measurement setup where a flow passes a router twice (Section B.2).

In general there is an offset $\Delta\omega_{0,y}$ between system clocks of two routers, since the system clock was started at different router boot times. Thus, the resolutions are not aligned, as illustrated in Figure 4.23 for different $\Delta\omega_{0,y}$ values. The three examples illustrated show how the distributions depend on this offset (same OWD in each case). From this and the previous examples we see that the error distribution cannot be known in advance, since it depends on the measurand itself as well as on the offset of system clocks. Moreover, the offset of system clock offset is never constant due to system clock skew. Therefore, even with constant delay, error distributions will continuously change and after a certain time the system clock skew leads to a complete cycle of possible distributions. For the case illustrated in Figure 4.23 this is the

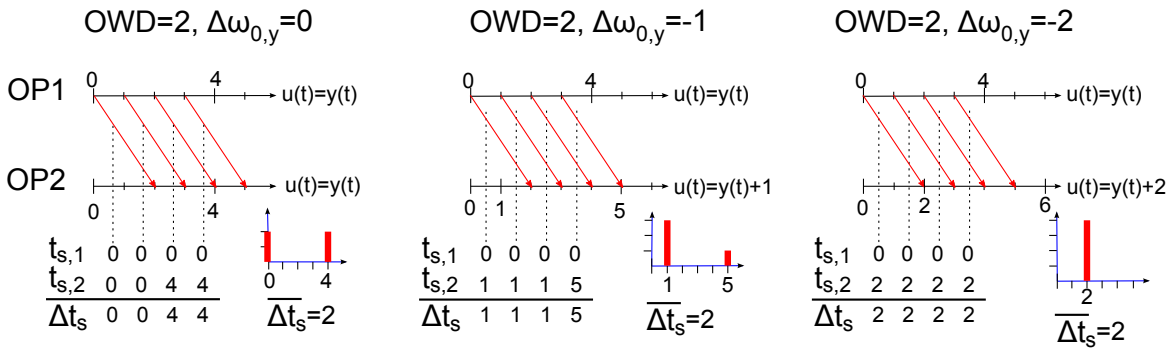


Figure 4.23: Impact of resolution offset due to system clock skew on error distribution

case when the combined system clock skew lead to an offset change of 4 ms. Depending on the system clock skew values, this can already happen within one minute. The resulting error distribution of such an interval will be the combination of all bimodal distributions and thus be a discretized triangular distribution.

The examples considered so far addressed the case when both OPs have the same limited resolution, i.e., the error distribution was a discretized triangular distribution. If the OPs have different resolutions, a discretized trapezoidal PDF with a higher number of peaks (see Section B.1) results. These effects are not detailed further in this thesis, since it is sufficient to be aware of the problematic cases with a very low number of peaks.

As indicated in Section 2.2.2, general confidence interval formulas are based on the assumption that random errors follow a normal distribution. Nevertheless, the confidence interval formulas based on the Student-T distribution (Equation (2.6)) and the formula based on known standard deviation (Equation (2.7)) can be used as rough approximation if there is no normal distribution. For distributions that result from OPPs where OPs have different timestamp resolutions, the PDF has a higher number of peaks and a good approximation can be achieved using the Student-T based formula. However, Student-T based confidence interval calculation with distributions that have few peaks only are highly problematic: If there are two OPs with the same timestamp resolution values, we will see distributions as indicated in Figure 4.23. If only few samples are observed in such cases, there might be a single peak only (Figure 4.23 right). Thus, the standard deviation is zero and hence a Student-T based confidence interval according to Equation (2.6) has zero width as well. There is a high probability that such a confidence interval is highly inaccurate: the next sample could be on the other peak. Hence, the mean value lies in between two peaks. The coarser the resolution of the two OPs, the more severe this problem becomes. Section 7.2.3 will show this problem based on OWD samples obtained from an OPP where each OP has a resolution of 64 ms.

The problem indicated using the Student-T based confidence interval calculation can be avoided by using the normal distribution based confidence interval formula with known standard deviation. Here, the standard deviation of the distribution resulting from convolution of the two resolution based continuous uniform distribution is taken as input (Equation (4.21)). Due to the discretized distributions, the standard deviations might differ from this value depending on the

position of the peaks. Nevertheless, the evaluations in Section 7.2.3 show that a fair approximation can be achieved.

This section has shown which random error distributions can be expected from resolution effects. In general, those distributions adhere to the shape that follows from convolution of rectangular distributions caused by resolution effects. However, the input quantities are not independent due to resolution effects and thus discretized versions of these distributions result out of it. This has an impact on the calculation of confidence intervals.

4.4.4 Exporter-internal Delays

After dealing with limited timestamp resolution and resulting effects, we now have a look at the exporter-internal delays on timestamp values. The timestamp creation model introduced at the beginning of this section (Figure 4.15) shows three points where delay in setting timestamp values can occur. At each of the points indicated, *more than one raw timestamp value* is created at *one defined time instant*. These defined time instants and the created raw values are

- **Record start time** t_s , i.e., the time at which the first packet of a flow arrives. At this time instant the raw timestamps r_{first} and r_{last} are written into the flow cache.
- **Record export time** t_x, i.e., the time when records are exported in a packet. At this time instant r_{su}, r_{sec}, and r_{nsec} are written into the packet header.

Depending on the exporter model and implementation, the raw timestamp values are not all written at the same time instant, but there can be a (sporadic) delay leading to timestamp errors. These potential delays are modeled by delay parameters D_x, which can be positive or negative. There are three such parameters in the timestamp creation model (Figure 4.15) and the errors caused by them will be called D_1-Error, D_2-Error, and D_3-Error.

The D_1-Error is caused by the delay between setting the raw record timestamp values r_{first} and r_{last} for the first packets of a flow (D_1). It leads to the effect that for flow records of one-packet-flows sporadically a duration larger than 0 ms is reported, i.e., a duration equal to the timestamp resolution (e.g., 4 ms). Such duration values are obviously wrong, but they indicate that the two raw timestamp values are not written at the same time instant. Most likely there are two consecutive operations that read the system clock's value and write it to memory. If the system clock just advances by one tick between the two operations, this results in the error described. In order to get rid of this blunder, record timestamps have always to be checked for the same value in case of one-packet-flows. If the two timestamps differ, they should be set to the same value, while it does not matter whether both values are set to t_s or t_e, as long as it is done in the same way for all records. The systematic error introduced by taking the one or the other is very small: it is the time between the two memory write operations to the flow cache. However, if the difference between the two timestamps is larger than the timestamp resolution, the record is subject to a large error (blunder) and must be dropped.

The D_2-Error results from the effect that at export time t_x, three timestamp values are written: the system clock's current value (r_{su}) and the RTC value in r_{sec} and r_{nsec}. On some exporter

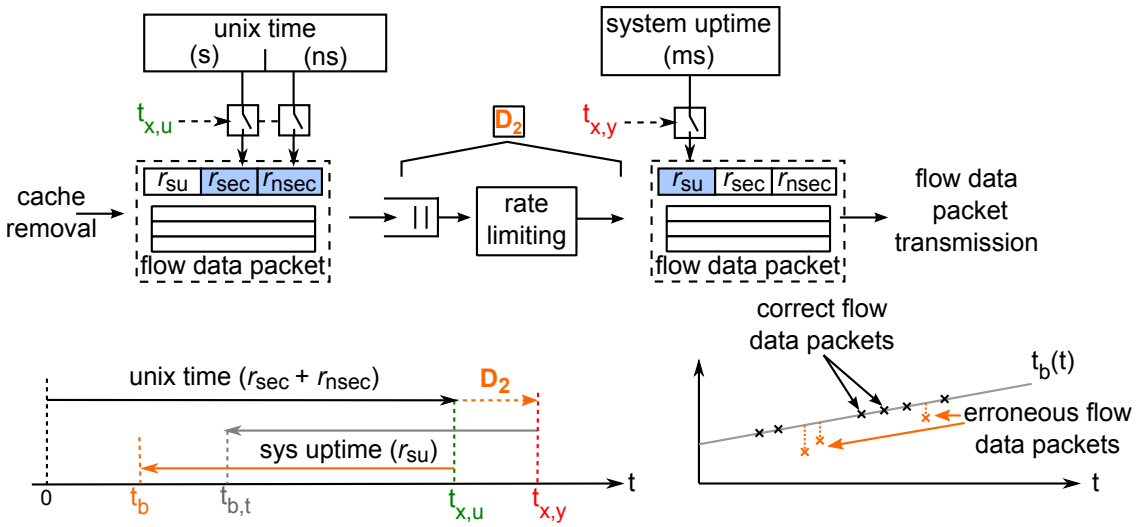


Figure 4.24: Error introduced by exporter-internal delay in the exporting process

models there is a delay D_2 between writing the system clock's and the RTC value, which leads to effects that record timestamps are sporadically considerably too low[4]. This problem is illustrated in Figure 4.24. As pointed out by Trammell et al. [74], RTC and system clock timestamps might not be applied at the same time to the flow data packet, but there might be a delay in high load situations, e.g., due to rate limiting. This leads to the effect that there is no single t_x from which both values result, but due to the delay the boot time t_b calculated from raw timestamps is too early, i.e., D_2 is negative in this case. The diagram on the bottom left of Figure 4.24 illustrates this error between the true boot time value $t_{b,t}$ and the reported boot time t_b. Since record start and end times are calculated based on t_b, these values of all records sent with the packets are affected by the error.

The error caused by D_2 happens sporadically only. Hence, it can neither be quantified as random or systematic error nor can it be compensated or averaged out. As a consequence, flow data processing mechanisms should try to detect records where this effect occurs and handle those records as blunder, i.e., drop them. Detection is possible: as shown in Figure 4.24 there is a discontinuity in t_b for these NetFlow packets compared to unaffected packets. As presented earlier in Section 4.4.2.2, t_b is not constant due to system time skew. Therefore, it is necessary to know whether $t_b(t)$ is ascending or descending when comparing t_b for blunder detection. As soon as t_b values are not monotonically increasing or decreasing, the D_2-Error is present. It is therefore named *boot time discontinuity blunder*. In principle knowledge of $t_b(t)$ could also help to compensate this error, since the r_{su} value of a certain RTC timestamp can be estimated based on this. A method that could be used for such compensation is proposed by Trammell et al. [74] for compensating the missing millisecond value in NetFlow v9 packets. This method is not considered further in this work.

Finally, the D_3-Error describes the effect that the nanoseconds-part and the seconds-part are not written into the packet header at the same time instant. It is likely that such problems sometimes

[4]The effect of timestamps being considerably too low was discovered by the author. The effect has independently also been reported by Trammell et al. [74], who also point out that the reason for this effect is exporter-internal delay.

occur, as this is the same problem of several write operations to memory as it was observed for the raw timestamp on flow start (D_1). However, it is impossible to quantify this delay value from flow data, since there is no reference to the time instant when the packet was really exported. Additionally, the resolution of r_{nsec} is typically very small. Hence, the impact of this error on flow start and end timestamps by comparison to reference start and end timestamp cannot be studied.

In contrast to other parameters of the timestamp creation model, the exporter-internal delay causes sporadic errors that cannot be modeled as systematic or random errors. However, the problems arising from D_2 and D_1 can always be detected. Thus, affected samples can be compensated (D_1-Error) or dropped (D_2-Error). Hence, the D parameters need not to be quantified, but are only symbols indicating the presence of a possible error at this point of the model. In principle, a quantification of how often the errors occur is possible in order to obtain knowledge on how many records and how many OWD samples are affected. However, this is no requirement for the OWD calculation itself.

4.4.5 Summary and Consequences for System Design

All timestamp errors and effects presented in the previous sections require appropriate handling in order to reduce the impact of these errors on OWD measurement. Depending on the error, it is possible to compensate the effects including knowledge on the effect's presence and its parameters or quantify the error in terms of uncertainty. If both is not possible and errors are detected, samples must be discarded.

A summary of the effects and errors is given in Table 4.4. For each effect, the impact on OWD measurement is given as well as whether the effect is sporadic or always present and whether the extent of the effect depends on the OP or Exporter. Additionally, the fifth column of the table indicates how an error can be detected and possibly compensated. The last two columns show whether the raw timestamps of flow records and/or which parameters quantifying the effect are required.

From Table 4.4 it can be concluded that almost all effects demand for knowledge on exporter or OP parameters for proper detection/compensation. These parameters describe the RTC offset $\omega_{0,u}$ and skew $\varphi_{0,u}$, the system time skew $\varphi_{0,y}$, and the raw timestamp resolutions ρ_{x}. Additionally, there are certain effects that do not cause an error themselves, but which impact the characteristic of other errors. One such effect is ESA, which impacts systematic and random error. Another effect is resolution shift, which impacts the distribution of random error. As a conclusion, a system for OWD measurement has to know about these effects and the parameters in order to quantify and compensate the errors for enhancing OWD samples accuracy. Failing to take these effects into account can lead to large errors, e.g., 30 ms systematic error from different OP resolution in the case of 4 ms and 64 ms resolutions on an OPP. Chapter 6 presents the exporter profile that specifies the required parameters and also methods for obtaining them.

Table 4.4: Timestamp errors summary

Error/effect	Impact on OWD measurement	sporadic	OP/Exp. specific	detection/ compensation	raw ts req.	param. required
RTC offset and skew	large and variable systematic error	-	x	comp., if linear	(x)[1]	$\omega_{0,u}$, $\varphi_{0,u}$
skew of internal clocks	export delay dependent error	-	x	comp based on $\varphi_{0,y}$ and export delay	x	$\varphi_{0,y}$
	resolution shift	-	x	-	x	$(\varphi_{0,y})$
limited timestamp resolutions	systematic error	-	x	comp. of bias	-	ρ_s, ρ_e, ρ_{su}, ρ_{sec}, ρ_{nsec}, ESA
	random error	-	x	quantification of random error, calculation of confidence intervals	-	ρ_s, ρ_e, ρ_{su}, ρ_{sec}, ρ_{nsec}, ESA
Export-Systime-Alignment (ESA)	no error from ρ_{su}	-	x	in conjunction with random/systematic errors	-	-
Exporter-internal delays	D_2-Error: t_s values too low	x	(x)[2]	detect and drop	x	$\varphi_{0,y}$
	D_1-Error: $d > 0$ and $p = 1$	x	(x)[3]	comp.: consistent setting if $d \leq \rho_d$	-	-

[1] Without raw timestamps, RTC compensation accuracy depends on internal clock skew since the export delay dependent error cannot be compensated and adds to RTC error.
[2] On some exporters in high load situations
[3] Exporter specific, but general compensation method available

Another point to consider in system design is that there is a dependency between timestamp compensation steps and that the compensation steps therefore have to be performed in the correct order. This is especially critical for steps that compensate RTC or system clock skew, since the system clock skew is defined as skew between system clock and RTC (and not UTC). Thus, performing RTC compensation before performing compensation steps that are based on system clock skew parameters would lead to wrong results.

The various steps as well as how they are performed in sequence and which parameters are taken into account in each step are depicted in Figure 4.25. The steps can be grouped into three stages: blunder filtering, timestamp correction, and uncertainty quantification. The two blocks of the blunder filtering stage check for errors resulting from exporter-internal delay and discard records with implausible timestamps as blunder. In the timestamp correction stage, the fist step compensates the export delay dependent error caused by system clock skew. Then, the

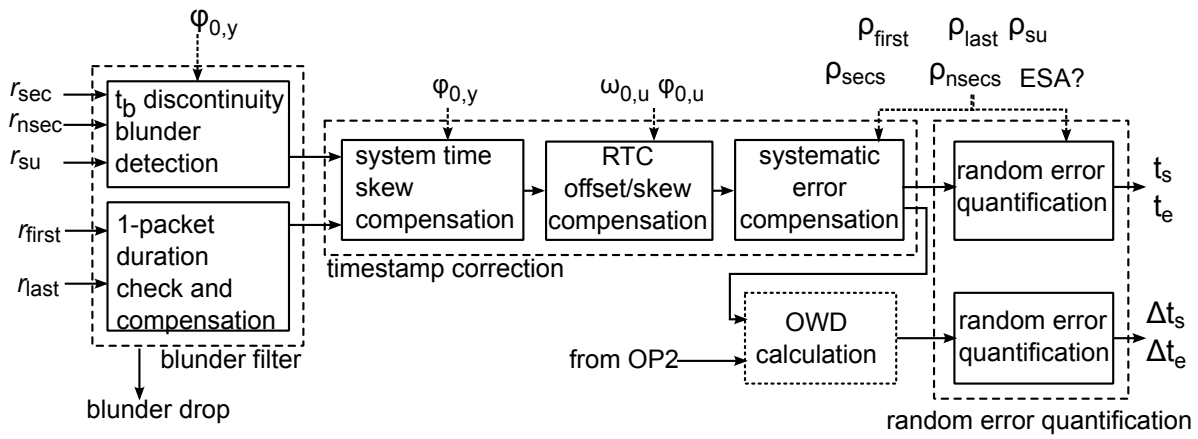


Figure 4.25: Steps for timestamp error detection and compensation

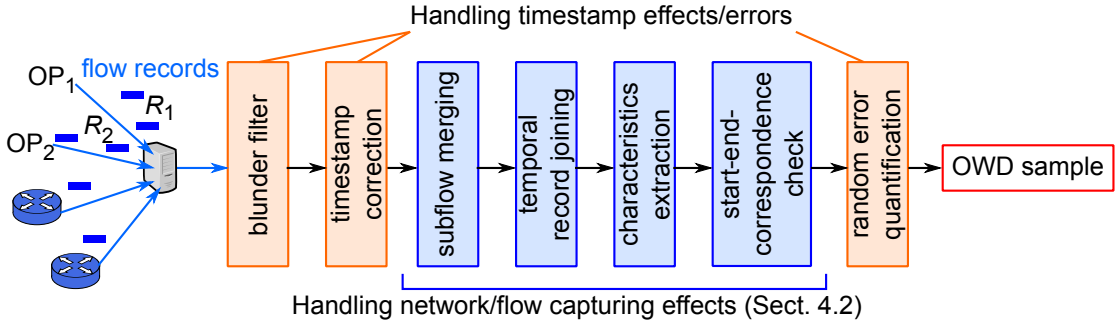


Figure 4.26: Extraction model extended for taking timestamp errors into account

RTC offset/skew is compensated, for which it is mandatory to happen after t_b error detection only, not before. The last step of the timestamp correction stage compensates systematic errors depending on resolution parameters and whether there is ESA or not. The same parameters are also input to the last stage, which offers two possibilities for random error quantification. While it is possible to quantify the random error or give confidence bounds per timestamp of an OPP, a preferred way for OWD calculation is calculating the confidence intervals per OWD sample. Thus, OWD is calculated first then confidence intervals are obtained.

The timestamp compensation steps have to be added to the extraction model that has been developed in the previous section for considering different network and flow capturing effects and errors (Figure 4.12). Figure 4.26 shows the model extended by the three blocks for blunder filtering, timestamp correction, and random error quantification. It is essential to perform filtering and correction steps before any other processing steps like merging or joining, since the latter also do processing based on timestamps.

In summary, this section introduced a model for timestamp creation and highlighted typical effects and their parameters as they can be found in flow data. The model was created based on characteristics found in a data set from a global enterprise network containing flow data from several hundred routers. Additionally, some properties of the model have been validated by experimental setups. The model is general and by the use of the parameters it can be adapted to

different exporter characteristics as well as to other protocols (IPFIX, NetFlow v9). Despite the general approach of the model there might still be existing or upcoming exporters with characteristics that cannot be mapped on this model and certain timestamp effects might not have been quantified. Therefore, it is important to check whether results of timestamp compensations or error quantifications are reasonable. In addition to that, strict rules for plausibility checks should be applied.

4.5 Summary

This section dealt with effects and errors from flow capturing that impact OWD extraction from flow data. Step by step an extraction model has been developed that deals with these effects. It is important to note that even with perfect flow data OWD calculation requires evaluation of packet and byte counts due to network effects. Otherwise, it is not possible to judge whether the first and last packet of a flow correspond to flow record start and end times, a property that is called *Start-End-Correspondence* (SEC). Taking flow capturing effects into account, a flow can lead to a different amount of flow records on OPs with different alignment. This makes temporal record joining and subflow merging necessary for checking SEC. Furthermore, it has been shown that in certain network configurations (e.g., per-packet-load balancing, WAN-optimizers, sampled NetFlow), OWD extraction is possible for very few samples only or nearly impossible.

Errors in flow capturing are the well-known problem of record loss, as well as other errors like byte count errors and errors in timestamp values. Record loss leads to the effect that there are ambiguous cases in flow record matching and not all ambiguities can be resolved by record joining or subflow merging. Byte count errors complicate the problem of flow matching, but it is possible to calculate byte count error bounds. Limitations in timestamp resolutions and other errors impact record timestamp and therefore OWD samples. Resulting systematic timestamp errors can have large impact, but can be compensated if the resolution characteristics are known. Furthermore, limited resolutions lead to random errors with discretized distributions, which impact confidence interval calculation. Effects from exporter-internal clock skew can lead to errors of several milliseconds, but generally can be corrected. Exporter-internal processing delays lead to blunder, which cannot be corrected, but must be detected and samples have to be discarded. In summary, with knowledge on the export device, the errors can be modeled and addressed in a way that leads to considerable improvement in OWD measurement.

As shown, OWD extraction from NetFlow data is challenging in terms of data processing due to the different joining and merging actions to perform at high records rates. Furthermore, knowledge on the flow capturing devices is required in order to compensate and quantify errors that impact OWD as good as possible. The next chapter deals with processing of flow data for OWD extraction while Chapter 6 will introduce the exporter profile concept for gaining knowledge on flow capturing device properties.

5 Online Extraction of Delay Samples

Based on the delay extraction model described in the previous chapter, this chapter presents design considerations for an efficient online processing implementation of this model as well as dimensioning guidelines.

The first section introduces flow export effects and resulting problems for flow data processing. Furthermore, the first section states four fundamental requirements on flow processing for OWD extraction. Based on the problem statement and the requirements an online processing approach is developed that is based on a sliding window mechanism. In the second section design questions on the sliding window approach are considered and an integrated window based processing mechanism that implements all joining, merging, and extraction steps using one window is presented. The third section introduces a window dimensioning method that allows estimating the amount of OWD samples created as well as the memory consumption based on profile parameters that describe flow record arrival.

The online processing approach presented in this chapter has been implemented based on a processing framework that was developed in the context of this thesis [5]. This processing approach was used to obtain the results from flow data of an enterprise network that are presented in Chapter 7.

5.1 Rationale for Window-based Online Processing

This section highlights the export effects that lead to the problem that an OWD extraction block has to wait a certain amount of time for records. Furthermore, it presents requirements for OWD extraction and an approach of a multi-stage window-based extraction mechanism that addresses the problem under the given requirements.

5.1.1 Export Effects and Problem Statement

Due to the timeout-based flow record creation mechanisms implemented in flow capturing approaches like NetFlow, there are not only different temporal aggregations of packets into flow records (Section 4.3.1.1), but also the time at which flow data is available for processing and OWD extraction is affected. This means that in the simple case of a one-record-flow **F** OWD extraction can only start if a record from each OP is available at the collector, i.e., OWD ex-

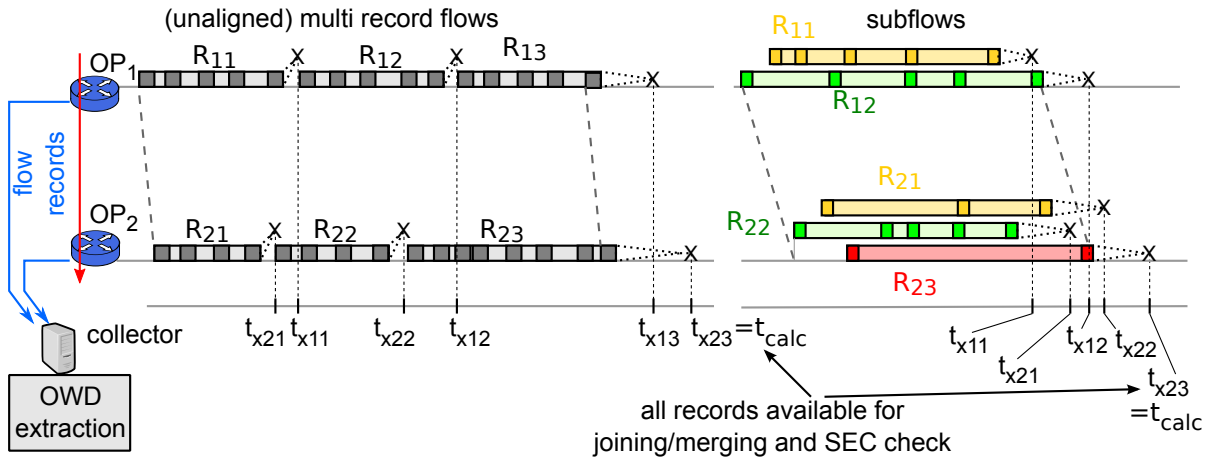


Figure 5.1: Impact of multi-record-flows and subflows on delay extraction time

traction can only start after $max(t_x\mathbf{F}_{,OP1}, t_x\mathbf{F}_{,OP2})$. As before, we assume that the delay between record export time t_x and receiving time at the collector is negligible.

If a flow is exported by means of several records, OWD extraction has to wait a longer time until all records are available as shown in the two examples of Figure 5.1. In the left example, flow information is exported by three records per OP as an unaligned multi-record-flow. In the right example different records result from different subflows (effects introduced in Section 4.3.1). In both cases, OWD values can only be extracted after all records are available since record joining (left case), record merging (right case), or both, are necessary for performing the check for Start-End-Correspondence (SEC) (Section 4.3.1).

In the case of a multi-record-flow (Figure 5.1 left) at some point the decision has to be taken whether further records are to be received for a flow or whether the last record has been received already. In general, this cannot be known from flow data alone. The decision could be based on heuristics that take the exporter characteristics into account. E. g., if the timeout settings and their accuracy is known, the extraction stages can assume after a certain time that all records for a part or the complete flow have been received. If the pauses where no packets are sent in the flow are long, this will lead to aligned multi-record-flows and SEC check could be performed for each part of the flow independently.

Waiting until all records of a flow are available can take a very long time in cases where flows with durations of hours or days are present. In such cases where packets are sent continuously, flow records are triggered by active timeouts and are available for the complete flow duration. This will typically result in an unaligned multi-record-flow and waiting for all records is necessary to check SEC while only two OWD samples (flow start, flow end) can be obtained. While Δt_s is available after the first records, it cannot be used as OWD sample because Δb and Δp can be calculated only at the end in order to validate whether SEC holds.

In terms of a flow where records are created for several subflows (Figure 5.1 right), the arrival time of the last subflow record determines the time at which characteristics can be calculated. Even if at any time before complete data of subflows is available, OWD calculation on a per-subflow basis could lead to wrong results, due to effects like multi-inter-subflow change detailed in Section 4.3.1.2. Joining multi-record-flows in time and merging of subflows demands for

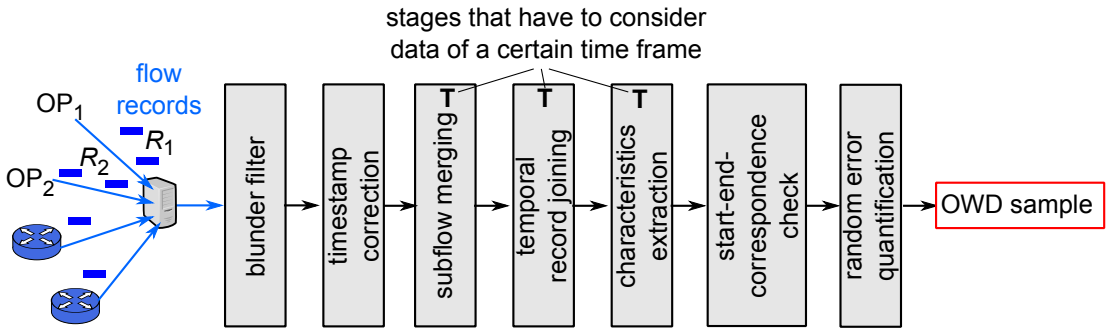


Figure 5.2: Extended extraction model indicating stages that have to consider data of a certain time interval

waiting until the last record of this flow arrived, or if the pause without receiving further records has been long enough, as highlighted above.

Both previously described problems illustrate that several stages of the OWD extraction model introduced in Section 4.4.5 require to take records of a certain time interval into account, i.e., they have to wait until the required data is available. Figure 5.2 shows the extended extraction model where the stages that consider data of a certain time frame are marked by a **T**. These stages are subflow merging, temporal record joining, and characteristics extraction. If the stages cannot wait until the required records of a flow have arrived, SEC check will fail and no OWD samples can be created for these flows. Section 5.1.3 addresses this problem based on the requirements that are introduced in the next section.

5.1.2 Requirements on Processing

This subsection states four fundamental requirements on flow data processing for OWD measurement that have to be considered in designing the online extraction mechanism. For these requirements the properties of the input data (flow data collected from several exporters) as well as the requirements of downstream applications that make further use of OWD calculations are taken into account.

Requirement 1: Timeliness

The measurement results should be delivered as early as possible to the application. OWD samples are typically used to detect network performance problems like throughput limitations or routing problems. Especially if business critical services are provided over the considered network, fast alarming and response is required else performance degradation can lead to financial loss. Systems that monitor higher layer services take OWD measurements into account in order to find problems for overall performance degradation by correlating monitoring data from several sources. For such systems it is essential that data is available timely in order to deliver analysis results early, such that helpdesk and support teams can fight problems by taking appropriate actions and respond to user complaints appropriately.

Requirement 2: Ordering

The measurement samples should be provided ordered by the timestamps they report the OWD value for. This means that if a measurement sample with a certain timestamp has already been delivered to the application, no measurement sample with an earlier timestamp must be delivered to the application thereafter. Such measurement samples are typically of no use, since an alarm has already been risen or OWD samples of that time interval have already been processed.

Requirement 3: Sample count and correctness

Flow data based OWD measurement should provide as many OWD samples as possible, for which it can, after checking with all available information, be assumed that the samples are correct. The high amount of samples is important in order to obtain OWD in the highest granularity possible, i.e., not only per subnet or host, but also per transaction or service class. Additionally, a high amount of samples allows to create mean values of a certain time interval taking a higher number of samples into account, thus reducing the impact of random errors and increasing confidence.

Requirement 4: Reliability

Processing of flow data for obtaining OWD samples has to take into account that resources like processing power, memory and I/O are limited. Flow processing must be operable and correct even under the highest load. This means that designs must consider ressource consumption in peak load scenarios, i.e., the highest record rate that is expected, which depends on traffic and flow capturing devices.

As the next section will show, the requirements contradict each other to a certain extent, e.g., if the fastest possible delivery of measurement results is chosen, the highest sample count cannot be achieved. The following sections will provide a suitable processing approach with configurable window parameters that allow to adjust the system behavior for finding a suitable trade-off between the requirements.

5.1.3 Approach of a Multi-stage Window-based Extraction Mechanism

There are several degrees of freedom for implementing the OWD extraction model. This section discusses five implementation considerations based on the requirements introduced previously.

Implementation Consideration 1: Offline vs. Online Processing

Section 3.5 introduced offline and online processing approaches. With offline OWD calculation, flow data collected from the network is written into files where each file contains flow data of a certain time interval or a certain amount of flow records. While such a solution typically fulfills Requirement 4, it does not fulfill Requirement 1, since it takes at least the time to write the file before the measurement results are available. Consequently, such an offline processing approach is not suitable, but the flow records have to be processed right after they are collected in an online fashion. Thus, all stages of the extraction model have to be implemented in an online processing approach.

Implementation Consideration 2: Exploiting Multiprocessor Architectures

In order to fulfill Requirement 4, CPU resources have to be considered in order to cope with the flow record rate that is expected in typical network scenarios. Thus, efficient processing mechanisms have to be used, as well as the processing itself should be parallelized as far as possible by splitting the online processing in several threads that can exploit modern multicore architectures. Since the extraction model consists of several independent stages, each stage can be implemented as separate processing block running as a different thread. The record receiving stage as well as all filtering and correction stages have been implemented this way based on the flow processing framework designed in the context of this work [5]. Some processing blocks even internally make use of several threads.

Implementation Consideration 3: Local vs. Distributed Processing

If processing power or memory on a single computer is not sufficient, processing could also be distributed across several computers. This includes several computers within a LAN (compute cluster) or computers that also collect flow records in a distributed way and are hence have to communicate via a WAN among each other (see Section 3.5). Distributed processing could be achieved by splitting record streams in a way that each computer can compute OWD samples independently from other computers, i.e., OWD samples for each OPP could be extracted separately. Splitting record streams accordingly can only be performed if it is known which OPPs are traversed by flows from certain source addresses and to certain destination addresses. This requires *knowledge on routing*, which could be gained by corresponding profile parameters. Using such extensions, parallel flow data processing for OWD extraction is feasible.

Distributed processing, however, has not been considered further in this thesis since the processing framework was already able to handle more than 100,000 flow records per second in an online processing scheme for scenarios with high memory requirements [5]. This performance is enough to handle the flow record rate of large enterprise networks. Additionally, this implementation is still prototypical and there is room for performance optimizations.

Implementation Consideration 4: Record Expiration

In addition to CPU resources, memory consumption has also to be considered for Requirement 4. From online processing, it results that flow records are buffered in memory until they can be further processed with respect to subflow merging, record joining, or characteristics extraction. In the characteristics extraction block at least two matching records from an OPP have to be available for creating an OWD sample. However, due to record loss and other effects it can happen that no matching records are found and unmatched records stay in memory. In order to solve this problem an expiration strategy is necessary that removes unmatched records from memory.

In general there are two possible expiration strategies: using a periodic cleanup task or a sliding window. A periodic cleanup task removes all records of a certain age at periodic intervals, while a sliding window is updated for every arriving record and removes expired records immediately.

Since every record removal requires changes in index data structures, it is more efficient to perform such operations in regular intervals. The expiration mechanisms implemented for OWD extraction is a sliding window consisting of several buckets. The window has a preconfigured length and moves bucket by bucket on record arrival, if required. Cleanups are performed at bucket boundaries, i.e., if the window moves by one bucket. The cleanup frequency can be adjusted: A higher number of buckets leads to a higher cleanup frequency. A bucket-based window allow for finding records with a certain timestamp more easily. They can be found by addressing a certain bucket and no linear search is necessary.

The sliding window for record removal is called *record window* and shown in Figure 5.3 on top. The record window with the size L defines how long records stay in memory. Its upper edge corresponds to the current time t_{curr}, in online processing typically taken from input data, i.e., the timestamp when the last record arrived at the processing stage. The lower edge of the window t_{expire} defines when records are removed. Flow records can arrive unsorted wrt. their timestamps and they are inserted into the window as shown in Figure 5.3.

The window size L is an important dimensioning parameter, since the number of records kept in memory determines the maximum memory consumption. Furthermore, a larger window will lead to a larger number of OWD samples that can be obtained: the more records in memory, the higher the probability that matching records can be found, especially for longer flows where joining of records is necessary. This clearly shows the contradiction of Requirement 3 and Requirement 4, which will be addressed in the window dimensioning method that will be presented in Section 5.3.

Implementation Consideration 5: Window for Result Sorting

Due to different flow and record durations, OWD samples calculated from records are typically unsorted wrt. the timestamp that they are related to. E.g., two records of a flow that has a duration similar to the record window will produce two OWD samples: one close to t_{curr} from flow end and the other close to t_{expire} from flow start. For a short flow, however, both OWD sample timestamp are close to t_{curr}. Depending on which record timestamp is taken to move the record window, OWD sample timestamps can even be outside of the record window.

In order to fulfill the Requirement 2, the OWD samples have to be sorted before they are forwarded. In order to achieve this, another sliding window for result sorting is used that is called *result window*. This window is depicted in Figure 5.3 below the record window and in general has to be larger than the record window. The result window is moved according to record timestamps aligned with the record window and reports results at t_{result}. In terms of memory requirements (Requirement 4) the result window is less critical than the record window since results are much smaller in size than flow records. A smaller result window leads to earlier delivery of measurement results (Requirement 1). If it is chosen too small, however, the probability that results arrive after t_{result} and are discarded becomes higher. The latter affects the number of OWD samples (Requirement 3), which shows that there is a contradiction of Requirement 1 and Requirement 3 in terms of result window settings that demands for finding a suitable trade-off.

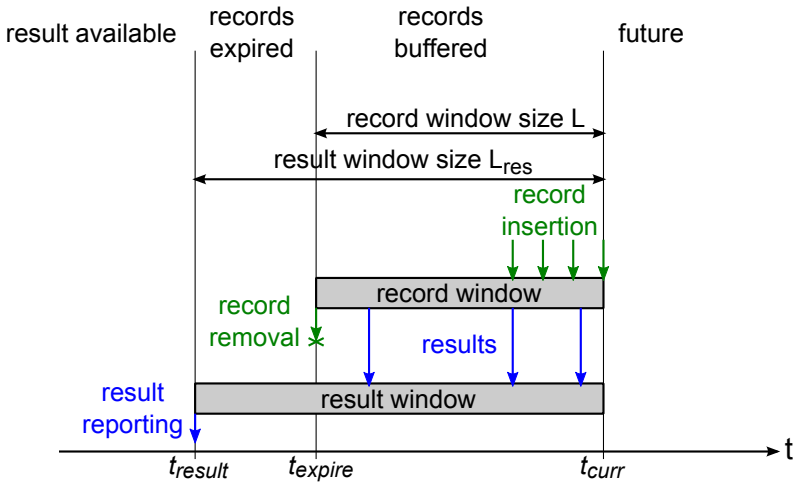


Figure 5.3: Window-based OWD extraction principle

Summary

The implementation considerations show that a multi-window-based online processing scheme realized using multiple processing blocks that can run as different threads is most suitable for OWD extraction. Furthermore, distributed processing is possible, but not a hard requirement at the current stage.

As shown the requirements contradict each other to a certain extent and suitable trade-offs between timeliness (Requirement 1), sample count (Requirement 3), and memory consumption (Requirement 4) are required. The ordering requirement (Requirement 2) demands for window-based result sorting. More details on the window mechanism itself will be considered in the next section while methods for addressing the most important window dimensioning trade-off will be presented in Section 5.3.

5.2 Window-based Flow Data Processing

The previous section derived a multi-window-based online processing scheme for the implementation of OWD extraction based on general requirements. In the following, the OWD extraction block and its window handling mechanisms are detailed. The main focus is on record window handling based on an implementation that uses *one record window* for joining, merging, and OWD extraction. In the first section, record export characteristics are studied and the term *export jitter* is defined. The second section highlights the impact of these characteristics on window handling and window dimensioning. The last section finally presents the design of an integrated window-based delay extraction processing block.

5.2.1 Export Characteristics in Detail

As introduced in Section 3.2.4, NetFlow removes records from the flow cache based on five different criteria: fast timeout (ϑ_{fast}), inactive timeout (ϑ_{inact}), active timeout (ϑ_{act}), cache full events, or closing TCP connections. Other stateful flow capturing approaches are similar. This leads to different minimal export delays ($\delta_{\text{e,min}}$) and minimal/maximal record durations (d_{min}, d_{max}) that depend on which timeout mechanism triggered the expiration or whether it was triggered by another mechanism (cache full events, TCP connections closed). Corresponding values for ideal timer handling and immediate record export are given in Table 5.1 for the assumption $\vartheta_{\text{fast}} < \vartheta_{\text{inact}} < \vartheta_{\text{act}}$, which holds for typical implementations and configurations.

The active timeout triggers expiration of records for active flows. Thus, the maximum duration of such records is as large as the timeout value itself. The minimum value results from flows where no more packet is seen after $\vartheta_{\text{act}} - \vartheta_{\text{inact}}$, and the active timeout is triggered right before the inactive timeout would occur. The minimal export delay occurs if a record times out and is sent just after the last packet was captured (record of length ϑ_{act}).

If the record is removed after expiration of the inactive timeout, a record might describe few packets only and can thus be of minimal duration zero. The maximum duration is limited by the case where the inactive timeout triggers just before the active timeout would ($\vartheta_{\text{act}} - \vartheta_{\text{inact}}$). With inactive timeout, the minimal export delay is ϑ_{inact}. Fast timeouts create records of duration zero to ϑ_{fast} that are in the latter case exported right after the last packet is seen (δ_{e}=0). All other expiration triggers create records with durations from 0 to the maximum (limited by ϑ_{act}) that are exported immediately. In summary, the record duration varies from 0 to ϑ_{act} and the minimal export delay varies between 0 and ϑ_{inact}.

Table 5.1: Impact of triggers for record expiration on record duration and export delay

trigger	d_{min}	d_{max}	$\delta_{\text{e,min}}$
Active timeout	$\vartheta_{\text{act}} - \vartheta_{\text{inact}}$	ϑ_{act}	0
Inactive timeout	0	$\vartheta_{\text{act}} - \vartheta_{\text{inact}}$	ϑ_{inact}
Fast timeout	0	ϑ_{fast}	0
TCP closed/cache full	0	ϑ_{act}	0

The values given in the table can serve as input for basic understanding and window considerations. Real values differ from these values due to several effects that lead to *export jitter*, which is defined as the difference between the minimum possible export delay $\delta_{\text{e,min}}$ and the actual export delay δ_{e} for a record:

$$j = \delta_{\text{e}} - \delta_{\text{e,min}} \tag{5.1}$$

There are several effects that lead to export jitter: first, record timestamps result from packet timestamps, which might be different from the time at which a timer expires. Second, exact

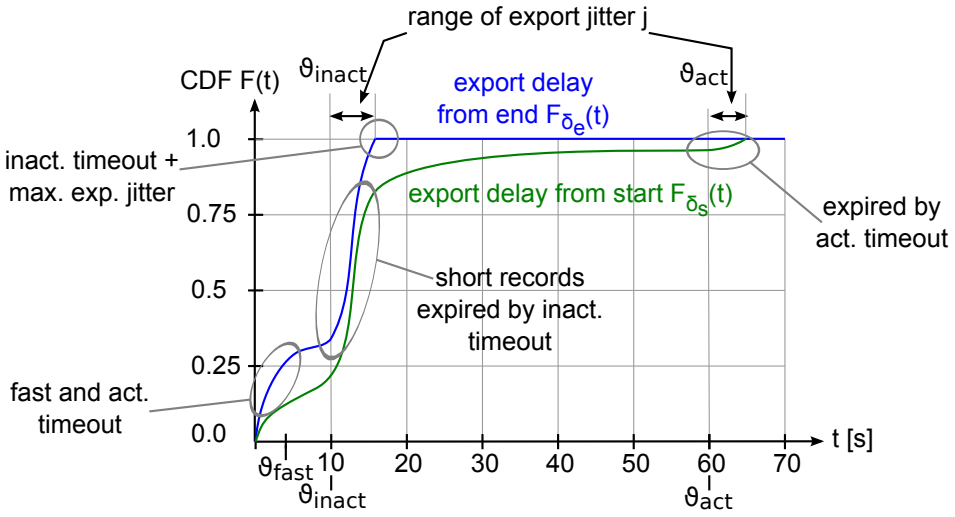


Figure 5.4: Typical CDF of export delays

timer handling is processing intensive, thus timers are handled at coarse grained intervals typically. Third, the export delay is not only caused by record expiration, but also by compiling several records into a packet. Consequently, real export delay values are often higher than values given in the table, while lower values are typically not found. In most cases there is a fixed upper bound for the export jitter.

If we consider distributions of export delay values, we get a *Cumulative Distribution Function* (CDF) as depicted in Figure 5.4, which is in detail traffic dependent. Section C.1 presents two examples of such a CDF taken from exporters of an enterprise network. This figure shows the typical distribution of the export delays δ_e ($t_x - t_e$, blue line) and δ_s ($t_x - t_s$, green line). These two values differ per record by the record duration d.

The CDF shows that in this example approximately 30 % of records have a an export delay δ_e below 5 seconds. Expiration of these records is triggered by fast timeouts, active timeouts, or non-timeout effects (as listed in Table 5.1). All other records are expired by the inactive timeout and thus show an export delay from end δ_e that is between ϑ_{inact} and ϑ_{inact} plus the maximum export jitter.

The distribution of δ_s shows that there are few short records exported by fast timeouts and many short records expired by the inactive timeout. Furthermore, the maximum value for δ_s is larger than the active timeout, which indicates that there are records longer than the active timeout due to export jitter of the active timeout.

5.2.2 Sliding Window Handling and Dimensioning

Online processing for OWD calculation requires a window that defines which records should be kept in memory for processing and which records can be removed. While the abstract window-based processing approach has been discussed in Section 5.1.3, this section covers important window handling details. We start by basic considerations taking only one-record-flows into

account, develop design decisions based on the previously considered export characteristics, and finally consider multi-record-flows. The following four basic design questions will be addressed:

1. Window movement: timestamp to use
2. Record matching: on insertion or removal
3. Size of record window
4. Size of result window

Regarding record window movement, there are three possible timestamps of arriving records on which the window can base: t_s, t_e and t_x. In the following, we consider window management properties and required window sizes for each of these possibilities. In contrast to previous considerations, we no longer consider export characteristic of a single OP, but both OPs of the OPP for which OWD is calculated have to be considered.

Using record start time t_s for window management it has to be considered that start time stamps of arriving records differ by $\vartheta_{act} + max(j)$, since the smallest export delay δ_s is 0 and the largest is $\vartheta_{act} + max(j)$. Hence, a window size of $\vartheta_{act} + max(j)$ is necessary. With smaller values records expired by active timeout could not be matched, since they are always dropped from memory when short records exported by fast timeout arrive ($\delta_s = 0$) and move the window. $\vartheta_{act} + max(j)$ has to be taken from the OP of the OPPs where this sum has the larger value.

In contrast to record start time, the variation of record end times is much smaller for a record stream, since it is only up to $\vartheta_{inact} + max(j)$. Thus, a much smaller window would suffice compared to using record start time. $\vartheta_{inact} + max(j)$ has to be taken from the OP of the OPPs where this sum has the larger value.

In the two cases discussed previously, the upper window edge (t_{uwe}) will be often equal to the export time t_x, since there are some records arriving where $\delta_s = 0$ or $\delta_e = 0$. If the export time is taken directly for window handling, this will always be the case and the window will not "jump" depending on export delays of arriving records as it will be the case when taking t_s or t_e for window handling. Furthermore, the window handling is based almost on the current time and thus also result creation and reporting are aligned. Thus, the system will provide results at known timestamps and not in a bursty way depending on export characteristics. If there are situations where no flow records arrive due missing production traffic, heartbeat messages that move the window can be send through the processing blocks. Consequently, using the export timestamp is the best solution and has been chosen for this work.

After considering the timestamp for the window movement, we now address the second design question: the record matching strategies. Figure 5.5 shows an export-timestamp-based record window and indicates which OWD samples are calculated from the records in the window. For creating an OWD sample for an OPP, the two flow records from each OP have to be matched. This can happen *on insertion* into the window, i.e., each time a record arrives, it is checked whether a matching record is already contained in the window. If the latter holds, both records are removed from the window and result samples are created. Another possibility is to match

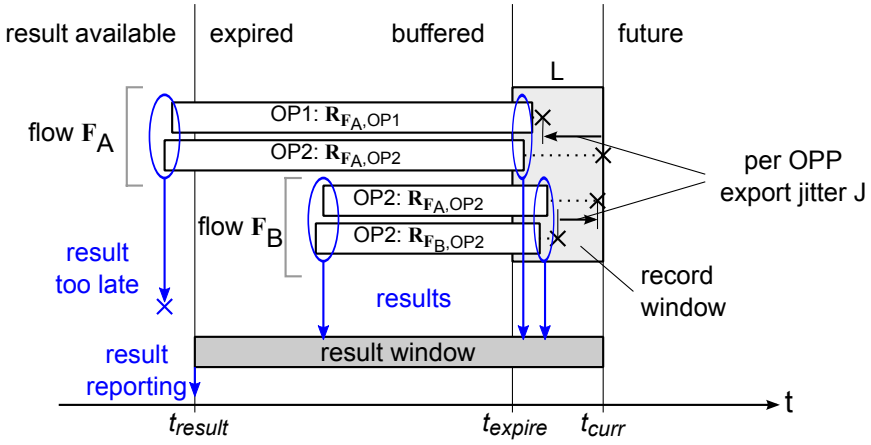


Figure 5.5: Window handling for one-record-flows based on export timestamp

records *on removal*, i.e., all records are simply added to the window when they arrive. Only when records expire, matching records are searched and result samples are created.

Matching on insertion saves memory, since records can be removed as soon as matching records arrive. However, if multiple matches are possible, matching and removal on insertion means a higher effort in terms of processing since always the possibility of further matches have to be evaluated. Multiple matches are possible as soon as multiple non-disjoint OPPs are considered: E.g., let there be three OPs OP_A, OP_B, and OP_C. If we assume that for the two OPPs OP_A, OP_B and OP_B, OP_C OWD samples should be extracted, a record from OP_B must not be removed as soon as a matching record from OP_A arrives, since there might still arrive a matching record from OP_C. Such additional information and checks are not necessary in case of matching on removal, since either all three records are in the window or not. Furthermore, if we consider multi-record-flows, we can only decide with high confidence whether SEC holds if all records have been arrived (i.e., are still in memory when matching). This also advocates matching on removal since a high accuracy is more important than memory savings. Hence, matching on removal has been chosen since it reduces processing complexity and also reduces the risk of matching errors.

The third design question is the size of the record window. For basic window dimensioning with export timestamps, the *per-OPP export jitter* has to be considered:

$$J = t_{\mathrm{x},\mathrm{OP2}} - t_{\mathrm{x},\mathrm{OP1}} \tag{5.2}$$

The per-OPP export jitter depends on the timeout settings and export jitters of either OP. J can have large positive or negative values for the worst case scenario where a record is expired by the active timeout on one OP and by the inactive timeout on the other OP. By using the definition

of δ_e, the t_x symbols in Equation (5.2) can be replaced and the minimum per-OPP export jitter can be derived as

$$J_{min} = t_{e,OP2} - t_{e,OP1} + min(\delta_{e,OP2}) - max(\delta_{e,OP1}). \tag{5.3}$$

We neglect the OWD $t_{e,OP2} - t_{e,OP1}$ in this formula since it is small compared to the export delays. Furthermore, we replace the minimum and maximum export delay by using values from Table 5.1 plus the per-OP export jitter j for the maximum export delay of OP1:

$$J_{min} = 0 - (\vartheta_{inact,OP1} + j_{OP1,max}). \tag{5.4}$$

J_{max} is derived accordingly as a positive value containing OP2 parameters.

In order to be able to match all one-record-flows in the window, the record window size must be

$$L_{max} = max(-J_{min}, J_{max}). \tag{5.5}$$

Using equations from above we can derive the maximum window size required for matching one-record-flows as

$$L_{max} = max(\vartheta_{inact,OP1} + j_{OP1,max}, \vartheta_{inact,OP2} + j_{OP2,max}). \tag{5.6}$$

This shows that export-time-based window management requires only a window size according to the per-OPP export jitter. Furthermore, decreasing inactive timeout settings also leads to a smaller window and thus less memory requirements. More advanced window dimensioning and asymmetric windows will be discussed in Section 5.3.

In terms of the last design question, the result window size, it has to be considered which result timestamps are to be expected from flow records inside the record window. If the result window is too short, this leads to sample drops, as illustrated in Figure 5.5. The latest result timestamp to expect is calculated from a record with the export time equal to the lower window edge of the record window that has the maximum difference between export timestamp and record start timestamp. This maximum time interval is $max(\delta_s) = \vartheta_{act} + j_{max}$, hence the result window size that assures that no results are dropped is $L + \vartheta_{act} + j_{max}$, with $\vartheta_{act} + j_{max}$ taken from the OP where this sum has the highest value.

The previous considerations hold only for one-record-flows. As soon as there are several subflows that must be compared or merged and temporal record joining is necessary, a window size of L_{max} is not sufficient. This is illustrated in Figure 5.6 for an example with two subflows $\mathbf{F}_{s1}$ and $\mathbf{F}_{s2}$ for which records at OP1 and OP2 are created. OP1 creates two records for the subflow $\mathbf{F}_{s1}$. In this case, a shorter window will not allow calculating any samples. With a shorter window, the record for $\mathbf{F}_{s1}$ from OP2 is not available for matching and due to subflow changes

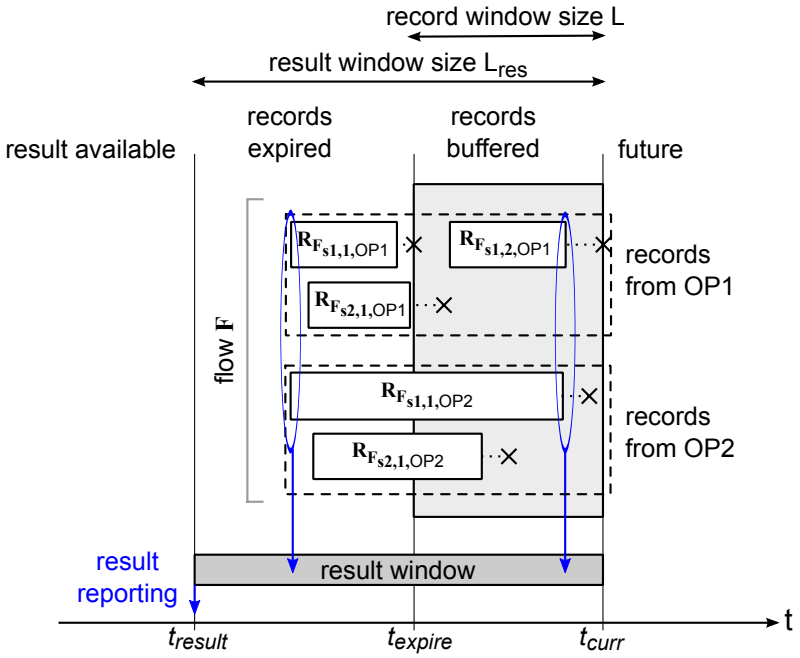


Figure 5.6: Window containing subflows of a multi-record-flow that require joining and merging

on the path no samples for $\mathbf{F}_{s2}$ without subflow merging can be generated. Consequently, only an infinite window size can always guarantee that all records of a flow are available for OWD calculation. Thus, the size of the record window will always be a trade-off and the amount of samples lost depends on traffic characteristic. If the number and characteristics of multi-record-flows are known, such a trade-off could be quantified. This can be achieved by extending the exporter profile towards such parameters, but is not considered in this thesis.

5.2.3 Integrated Window-based Delay Extraction

Based on the previously discussed considerations a delay extraction block that integrates record merging and joining has been implemented as proof of concept. It has been realized in Java based on the flow processing framework developed in the context of this thesis [5]. The framework allows using this processing block in a processing chain where other processing blocks perform flow data receiving, systematic error compensation, random error quantification, and other tasks. This section highlights the architecture of this delay extraction block and introduces its most important functional blocks.

The window-based delay extraction block is illustrated in Figure 5.7. It receives NetFlow record objects and sends OWD sample objects to downstream processing blocks. This figure shows several data structures (blue color), three functional blocks (red color), and an example with two flows that will be used to illustrate the overall block functionality at the end of the section.

The record window and the result window are realized as export time based window consisting of several buckets that are moved bucket by bucket. Window and bucket sizes are configurable.

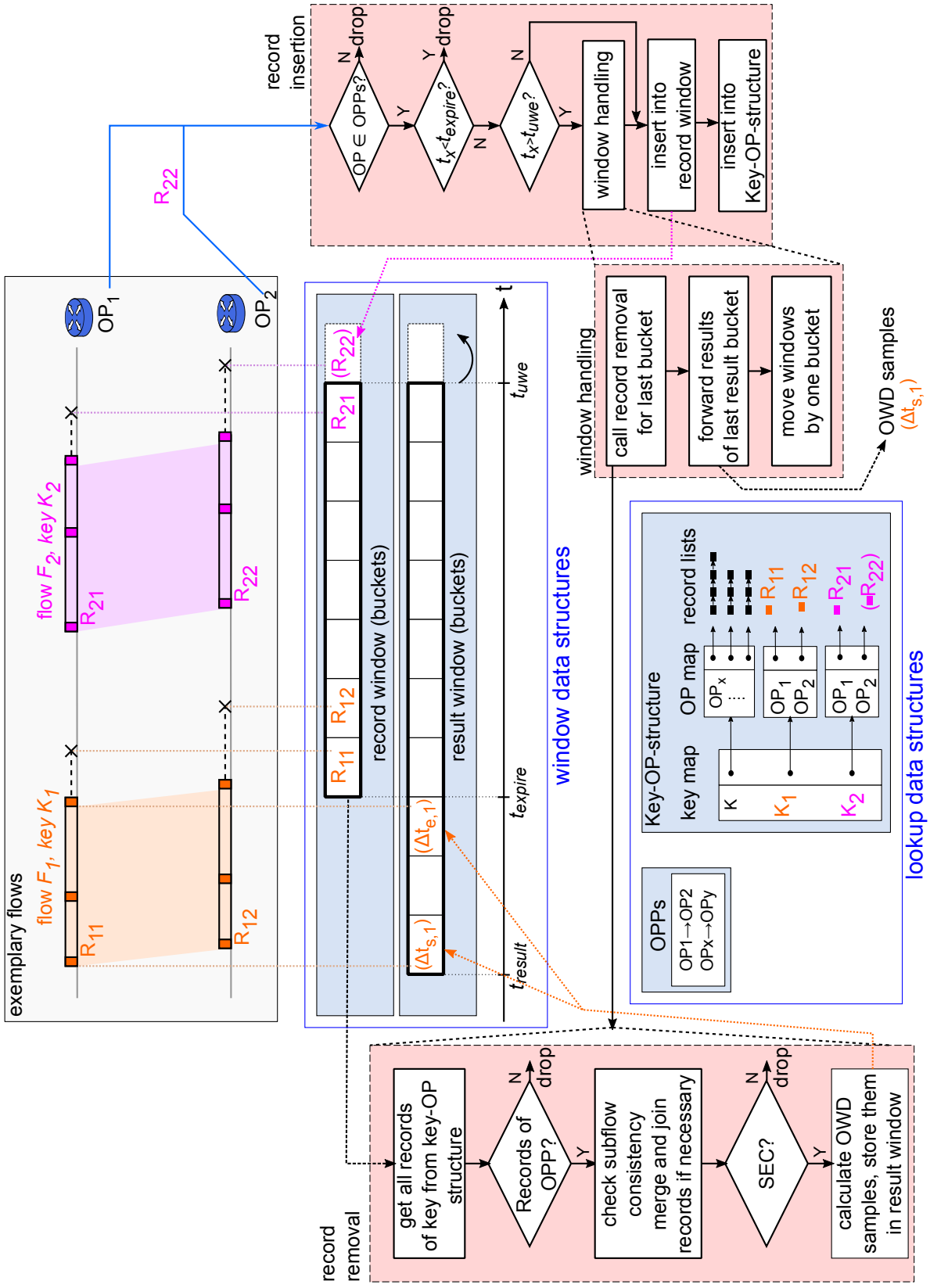


Figure 5.7: Integrated window-based delay extraction processing block

Each bucket can contain several references to record or result objects. At each movement, different tasks in the functional blocks are triggered. The window is bucket based since from an implementation perspective this is more efficient and easier to handle compared to a sliding window due to the following reasons:

- Inserting records into the window (based on timestamps) is more efficient if the bucket can be directly addressed. Since the window primarily is an expiration mechanism and timestamp based element access is not necessary, no sorting of records/results within the bucket is required.
- Deleting records does not need a timestamp based search in data structures since always all objects contained in a bucket are expired/handled.
- With a sliding window, each record arrival could trigger expiration/handling of data. A bucket-based window will trigger such actions less frequently. However, each expiration handling affects a higher number of records at once and, hence, is more efficient.

In addition to the window data structures, there are two look-up data structures shown at the bottom of Figure 5.7. The first data structure contains a list of the OPPs for which OWD calculation should be performed. This specifies the order of OPs for correct difference calculation and allows for dropping records that do not belong to an OP contained in this list. The second data structure is the Key-OP-structure that contains records that have been added to the record window. This structure is optimized for accessing records efficiently based on their key and OP. It consists of a flow-key based hashmap that links to a treemap, which contains for each OP a list of records.

As shown earlier in Section 4.3.1, records might have to be joined or subflows might have to be merged due to network and flow capturing effects. Due to the window-based mechanisms this processing block is not able to join records of long flows, where not all records fit into the record window, but will expire before the last record of the flow arrives. Therefore, this block always joins the records for a flow key and if the SEC check holds, it calculates OWD samples. Subflows are only merged if SEC does not hold on a per-subflow basis.

When a record enters the processing block, it arrives in the record insertion block on the right side of Figure 5.7. This block checks if the record belongs to a configured OPP and whether the record is too late to fit in the record window. If the export timestamp is larger than the timestamp of the upper window edge (t_{uwe}), the window handling block is called to advance the window. Afterwards, the record is added to the window and the Key-OP-structure.

In case of window movement, window handling functions will call the record removal functions (block on the left of Figure 5.7) for the last bucket and forward the results of the last bucket of the result window. Only after these steps have been performed, both windows are advanced by one bucket.

Only if records for both OPs of the configured OPPs are available, subflow consistency is checked and records are joined and merged if necessary. If the SEC check that takes into account byte count errors from profile values (see Section 4.3.2.1) succeeds, OWD samples are calculated and placed into the result window. It might happen that records are removed for

which the data in the Key-OP-structure has already been removed. In such cases, the records have already been processed and will be dropped.

After all blocks have been introduced, we now follow the mechanisms for the exemplary flow records shown in Figure 5.7. There are two flows that traverse the OPs of the OPP and create one flow record each, for which SEC holds. The figure shows the time instant before $\mathbf{R}_{22}$ arrives. Values that are added after the arrival of $\mathbf{R}_{22}$ are depicted in brackets. Records of $\mathbf{F}_1$ and $\mathbf{R}_{21}$ of $\mathbf{F}_2$ are already in the record window according to their export timestamp. However, results for these records have not been calculated. When $\mathbf{R}_{22}$ arrives, its export timestamp is larger than the upper window edge (t_{uwe}), thus it will trigger window handling. This will cause the calculation of OWD samples for $\mathbf{F}_2$, which are then placed into the result window. Afterwards the windows are moved by one bucket, and $\mathbf{R}_{22}$ is placed into the record window and the Key-OP-structure. Due to the moving windows, $\Delta t_{\text{s},1}$ will leave the processing block as OWD sample.

The design of this processing block has been implemented based on modular window handling functionality and has proven to work reliably for a high amount of records. The results presented in Chapter 7 have been obtained based on this processing block.

5.3 Profile-based Window Dimensioning

According to the requirements presented in Section 5.1.2, the system must operate reliably while providing as many correct measurement samples as possible. An important point for reliable operation is limiting the maximum memory consumption, which means that the window sizes have to be set in a way that the data buffered in windows never exceeds a certain limit. In order to estimate this limit, the record arrival behavior has to be known, which is specific for the network and flow capturing devices considered. In the following a profile based dimensioning approach is presented, that takes parameters for record arrival into account that can be obtained a priori from flow data (for profile creation see Chapter 6). These profile parameters allow estimating the memory consumption as well as the obtainable OWD samples based on the window size settings. These estimations can be used as input for resource dimensioning in order to find the desired trade-off between memory consumption and sample count (i.e., whether a memory upgrade would be worthwhile) or for applying optimization methods that take the desired sample count on OPPs as input and provide corresponding window parameters.

First, we address the number of OWD samples that can be gained depending on the window size. This value depends on the per-OPP export jitter, as already mentioned in previous subsections where this dependency was illustrated in Figure 5.5 and expressed in Equation (5.6).

The two OPs of an OPP in general handle timeouts for the same flow differently due to implementation or configuration differences as well as due to different load. This results in the effect that certain per-OPP export jitter values are more likely to occur more than others, which can be expressed in an export jitter PDF $f_R(J)$ that describes the probability that two matching records have a certain export jitter value J. Figure 5.8 illustrates an asymmetric export jitter PDF as it often occurs. In the example of Figure 5.8, a large part of records is first exported by OP2 and then by OP1. Hence, on average for the record export time t_{x} the condition $t_{\text{x,OP2}} < t_{\text{x,OP1}}$ holds for the same record of a one-record-flow. Due to such effects, *per-OP window dimensioning*

can be advantageous, i.e., using a different window size for each OP that suits its export characteristics in order to save memory. Figure 5.8 shows the window sizes $L_{\mathrm{OP1,max}}$ and $L_{\mathrm{OP2,max}}$ that guarantee that all records can be matched. Window sizes are defined to be always positive.

Based on these export jitter considerations, we now quantify the number of OWD samples that can be obtained using a certain window size. Here, it is not sufficient to take the amount of records that can be matched into account, since from two records that are matched either two samples or one sample (in case of one-packet-record) can be calculated. As one-packet-records exhibit a different export behavior than longer records, this distinction is necessary and a sample distribution $f_S(J)$ that considers this effect is required. This PDF has the same bounds as $f_R(J)$, but shows differences especially in regions where multi-packet-flows are exported (see histograms in Figure 7.10 in Chapter 7).

For calculating the sample count S we take the maximum number of samples $S_{\max}$ that can be calculated using the maximum window size $L_{\max}$ from Equation (5.5) as input. Based on this value and the sample-based PDF $f_S(J)$, the amount of samples can be calculated for other per-OP (asymmetric) window settings as:

$$S = S_{\max} \cdot \int_{-L_{\mathrm{OP2}}}^{L_{\mathrm{OP1}}} f_S(J)\, dJ \tag{5.7}$$

If using a single record window (symmetric case), L has to be used as integral limits in Equation (5.7). The discretized versions (histograms) of $f_R(J)$ and $f_S(J)$ are denoted as $h_R(J_i)$ and $h_S(J_i)$. They will be usd in the exporter profile and for evaluations based on measurements.

Next, we address the memory consumption for a given window size. First, dimensioning rules considering the overall record rates are derived, thereafter, more sophisticated approaches for

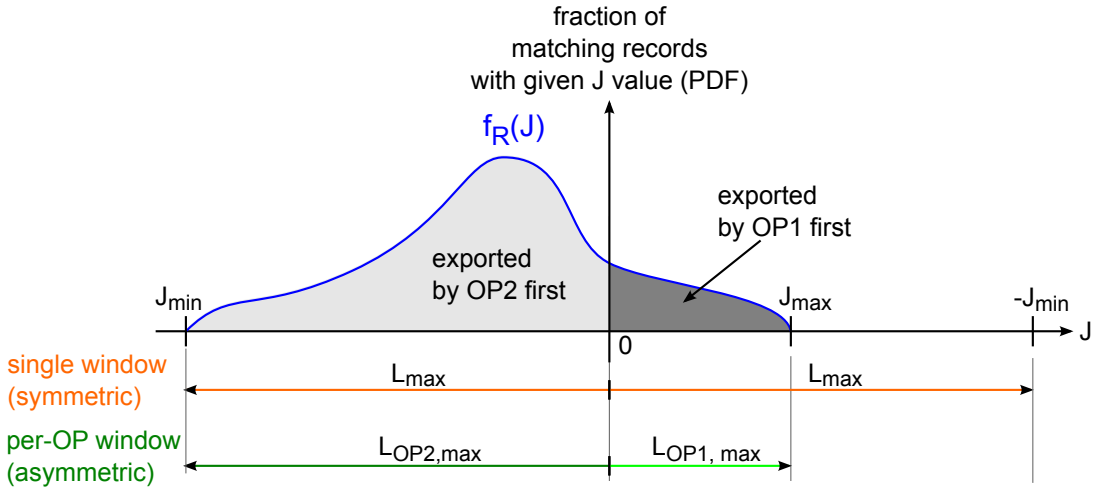


Figure 5.8: PDF of export jitter per record and resulting window sizes

per-OP windows are considered. For a window that is based on export timestamps, the amount of records in the record window $|\mathscr{R}_L|$ depends on the record rate λ_R and record window size L:

$$|\mathscr{R}_L| = \lambda_R \cdot L \tag{5.8}$$

For dimensioning purposes the maximum number of records in the window is relevant in order to guarantee that the system can always deal with the memory available. This is a worst case assumption since records for which a matching record for OWD sample calculation have been found can be removed from memory and will not stay for the complete window length in memory. Due to the typical implementations of record expiration and export, the arrival rate is not constant, but exhibits certain burstiness. This is an important point to consider for the maximum amount of records in a window, especially for small windows: the smaller the window, the lower the effect that bursts will average out. This can be illustrated with an example where we consider two bursts with 100 records that each arrive in a 10 seconds interval, i.e., the mean rate is 20 records per second. A large window of size 10 seconds will contain at maximum 200 records. However, a window of size 1 second will not contain at maximum 20, but 100 or even 200 records if the bursts arrive within the same second. In order to care for such burst effects, the maximum expected record rate computed based on the time interval that corresponds to the window size ($\lambda_{R,\max}(L)$) has to be taken into account. Therefore, we get:

$$|\mathscr{R}_{L,\max}| = \lambda_{R,\max}(L) \cdot L \tag{5.9}$$

The bursty nature of record arrivals can lead to effects where a reduction of the record window size below a certain limit does not further reduce maximum memory consumption, but the maximum memory consumption stays high even for smaller windows. With the knowledge on the overall maximum record rates $\lambda_{R,\max}(L)$, Equation (5.9) can be used to quantify the memory necessary for symmetric window settings for system dimensioning purposes.

When it comes to asymmetric windows, the record rate of each OP has to be considered. Here, Equation (5.9) has to be extended in order to reflect the impact of different window sizes in the calculation of the maximum amount of records in the record window. In this case it is no longer sufficient to consider the overall maximum record rate, but the record rate of each OP has to be taken into account. A simple solution would be taking the maximum rate of each OP. However, the maximum rates do not occur on all OPs at the same time, thus the simple solution could heavily overestimate the required memory. Especially in global networks, the maximum record rate will occur at very different times at different OPs due to the different working hours. This fact is exemplarily illustrated in Figure 5.9 that shows the record rate over a working day for different OPs of a global enterprise network with most traffic created by European locations. OP1, OP2, and OP3 export records from routers of different continents. The record rate consequently is higher during working hours and low during the night. For locations where data centers are connected to the global network (OP4), the behavior is different and a wider peak results since data center services are often accessed from global locations.

In enterprise networks, such characteristics do typically not change a lot, but similar record rate patterns occur on each working day, while only weekend characteristics differ. Record-rate

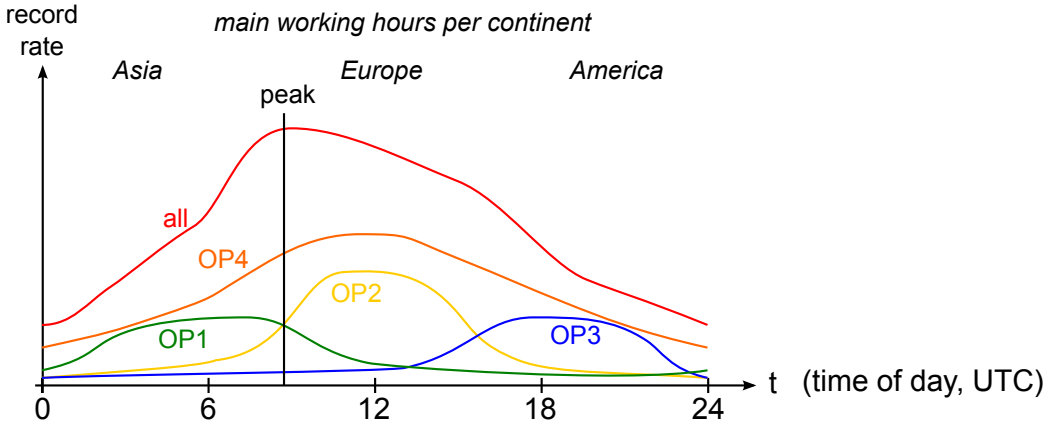


Figure 5.9: Typical record rate per OP over a working day

impacting events present in the Internet like large-scale port scans, attack traffic, or flash crowd effects are typically not present in enterprise networks. Of course, there might be irregular bulk data transfers in enterprise networks. However, they only affect the network utilization pattern, but do not change the record rate significantly since such transfers causes only few additional flows. Consequently, a record rate pattern recorded for one working day can be taken as input for window dimensioning and will require only updates when major configuration or organizational changes occur.

In order to take this daily traffic pattern into account, the record rate for different time intervals T_j has to be considered, i.e., $\lambda_{R,\max}(L,T_\mathrm{j})$. A coarse grained consideration is sufficient, thus each interval can, e.g., represent one hour ($j = 1..24$). The worst case maximum amount of records in the window can therefore be derived from the maximum record count that can occur in each of the time intervals, which is the sum of maximum records kept from each of the n OP_i:

$$|\mathscr{R}_{L,\max}| = \max_j \left(\sum_{i=1}^{n} \lambda_{R,\mathrm{OP_i},\max}(L_{OP_\mathrm{i}},T_\mathrm{j}) \cdot L_{OP_i} \right) \tag{5.10}$$

Both dimensioning formulas (Equation (5.7) and Equation (5.10)) require knowledge on the export behavior as input. The approach in this work is to provide these parameters as parts of the exporter profile. Profile parameters required for the dimensioning guidelines introduced previously are the sample distribution $f_S(J)$ per OPP and the record rates $\lambda_{R,\mathrm{OP_i},\max}(L_{OP_\mathrm{i}},T_\mathrm{j})$. The latter is required for each Observation Point on an hourly basis for a typical working day and different window sizes L in order to take the burstiness of record arrival into account.

5.4 Summary

This chapter detailed how the delay extraction model is efficiently implemented and it also considered design as well as dimensioning trade-offs. Starting from the problem of export

characteristics, the requirements for processing records of a certain time interval was highlighted. Based on four fundamental requirements five implementation considerations have been addressed that underline why a multi-window-based online processing scheme which is realized by using multiple multithreaded processing blocks is most suitable for OWD extraction.

The second section discussed four design questions on sliding window implementations based on a model for per-OP and per-OPP export characteristics including the definition and application of export jitter. An architectural and functional overview of the integrated window-based delay extraction block has shown how all previously introduced concepts could be realized in a processing block that is sufficient for processing the data from a large enterprise network.

In the third section a profile-based window dimensioning approach has been presented that estimates the sample count to be achieved and memory consumption to be expected for different window settings. The dimensioning approach is based on two exporter profile parameters that reflect per-OPP export jitter and per-OP record rates for different times of day during a working day. This shows that exporter profiles are not only necessary for error compensation and quantification, but are also input for optimizing OWD extraction processing block.

6 Exporter Profile Creation

Flow data exhibits several properties with respect to attributes and timestamps that can lead to errors in OWD calculation, as discussed in Chapter 4. If such properties are known before matching records, calculating timestamps, and creating OWD samples, errors in OWD samples can be compensated, quantified, or avoided. Furthermore, knowledge on record rate and sample distribution allows for dimensioning of online processing for better resource utilization (see Chapter 5). Using exporter profiles that quantify these properties consequently improves the accuracy and performance of OWD measurements. This chapter details the exporter profile concept and a method for obtaining profile parameters. The first section introduces the profile approach as well as fundamental properties and dependency classes. In the second section the parameters of the profile are defined, while in the third section approaches for profile creation are discussed. The fourth section details a method for obtaining profile parameters directly from flow data.

6.1 Profile Approach

Several properties of flow data depend on export devices as well as on network configuration and network properties. These properties lead to errors and effects that have to be taken into account in flow data processing, especially in OWD measurements. The approach chosen in this work is using *exporter profiles*. Exporter profiles contain metadata on flow data captured by exporters in a certain network.

In general a static and a dynamic profile approach can be distinguished: Either profile values can be predefined (static) by configuration/from reference measurements, or they could be dynamically adjusted by taking current flow data, traffic, and exporter properties into account. With dynamic profile creation, the profile parameters are created from current properties, i.e., by using a limited history in which data is evaluated. While in general this has the advantage that no predefined profiles are required, it could also lead to brittle system behavior, e.g., if special traffic or load situations arise and the dynamically created parameters become imprecise. Additionally, profile creation may require high processing effort, which is not available in flow data processing applications for dynamically creating profiles in parallel to flow data processing. Due to these reasons, *static profiles* are used in the following, which are created and checked offline.

As briefly mentioned before, the exporter profile approach does not only focus on component parameters, but also takes network and traffic properties into account. This means that profile

parameters are influenced by different dependencies. The following list introduces the fundamental *parameter dependency classes* that have to be taken into account in profile design and creation. A parameter may also have more than one dependency.

- **design/implementation-dependent parameters**

  These parameters result from design and implementation of the flow capturing device hardware and software. Hence, these parameters apply to a certain device model or software release.

- **component-dependent parameters**

  Within the same device model, electronic components may exhibit different behavior (e.g., skew/drift of quartz oscillators). Thus, such parameters differ from flow capturing device to flow capturing device, even within the same device model.

- **configuration-dependent parameters**

  These parameters are network or device configuration specific and therefore change if the configuration is changed.

- **traffic-dependent parameters**

  These parameters depend on the traffic characteristic like arrival rate or flow length.

In current standards, there are some flow data properties specified focusing on accuracy properties. However, they do not allow for such extensive error compensation and quantification for OWD calculation as presented in this thesis. The Internet Draft on an IPFIX configuration model [75] defines a template that can be used by an IPFIX device to report accuracy information. The information elements of this accuracy information are defined in the PSAMP-RFC "Information Model for Packet Sampling Exports" [RFC 5477], which has been originally mainly defined for exporting data from packet sampling using the IPFIX protocol [RFC 5476]. Information elements on accuracy are relative error, absolute error, confidence bounds, as well as confidence level. This information is sufficient for quantifying random errors[1] of timestamps or other flow attributes, but further parameters required in the context of this thesis are not considered. Additionally, the aim of these information elements is to report errors of which a device is aware of and not to create profiles on device properties. The idea of an extensive exporter profile that describes several effects and that also provides parameters useful for flow data processing was presented by the author at the IRTF NMRG meeting in July 2010 [3]. The detailed concept was published in [6]. Trammell et al. also argue in [74] to provide "implementation-specific metadata alongside flow data", but an extensive profile is not considered further.

6.2 Profile Content

The exporter profile contains a defined list of parameters. Its format is given in Table 6.1[2] and will be detailed throughout this section. As a general design principle, the profile always

[1] [RFC 5476] highlights that an absolute error of 2 ms means +/- 2 ms, i.e., a random error. Systematic errors are not explicitly considered, i.e., it is assumed that known systematic errors are already corrected.

[2] This exporter profile is an extended version compared to the initial one presented in [6]

contains parameters describing device properties and not derived parameters wherever possible. E.g., it does not contain a systematic error parameter, but it contains the resolution values that are input to systematic error calculation. Furthermore, the profile is kept general and captures all flow data errors and effects discussed previously. Only a small part of the profile is NetFlow v5 specific. With slight changes in the timestamp resolution part, it can be applied also to NetFlow v9 or IPFIX.

The parameters of the exporter profile are associated with different *profile components*, as shown in Table 6.1. This table gives for each parameter the parameter dependency class (see Section 6.1) and the data type for completeness and better understanding. The three profile components are introduced in the next paragraph before each component and its parameters are explained in detail. An exporter profile describing flow data effects in a network consists of multiple instances of each component.

Table 6.1: The exporter profile

			parameter dependency class				
profile comp.	**symbol**	**content**	**impl.**	**comp.**	**config.**	**traffic**	**data type**
Exporter Component (EC)	nodeID	ID of the node this exporter belongs to	-	-	x	-	ID
	$\omega_{0,u}$	RTC offset to UTC at t=0	-	x	x	-	float
	$\varphi_{0,u}$	RTC skew	-	x	x	-	float
	ρ_{sec}	resolution of r_{sec}	x	-	-	-	int
	ρ_{nsec}	resolution of r_{nsec}	x	-	-	-	int
	ρ_{su}	resolution of r_{su}	x	-	-	-	int
	ESA	Export-Systime-Alignment	x	-	-	-	bool
Observation Point Component (OPC)	expID	ID of the exporter this OP reports to	-	-	x	-	ID
	$b_{\text{p,max}}$	largest packet size at OP	x	-	x	-	int
	$b_{\text{p,min}}$	smallest packet size at OP	x	-	x	-	int
	$\varphi_{0,y}$	system clock skew	x	x	x	-	float
	ρ_{first}	resolution of r_{first}	x	-	-	-	int
	ρ_{last}	resolution of r_{last}	x	-	-	-	int
	$\lambda_{R,\text{max}}(L, T_{\text{j}})$	max. record rate per window size and time of day	-	-	x	x	2D array
OPP component (OPPC)	opID1	ID of first OP of this OPP	-	-	x	-	ID
	opID2	ID of second OP of this OPP	-	-	x	-	ID
	$h_S(J_i)$	Export jitter histogram per OWD sample	-	-	x	x	array

The *Exporter Component* (EC) of the profile exists once per exporting process, i.e., typically once per NetFlow-enabled router and contains exporting process-specific parameters. It carries a node ID in order to uniquely identify the component and for mapping it to the network node, e.g., to the router. Since in a router several different metering processes can exist at different observation points, each with possibly different characteristics (see Section 3.3), the *Observation Point Component* (OPC) of the profile is defined independently of the EC. In order to reflect the 1:n relation between OPC and EC, each OPC contains an expID that reflects which exporter process exports the metering data obtained at this OP. Several parameters are obtained *per path* between two OPs. These characteristics are part of the third component: the *Observation Point Path Component* (OPPC). The direction of this path is reflected by two parameters that indicate its first and second OP based on the respective IDs.

After an overview of the profile components has been given, the component parameters are now described in detail. The parameters and symbols used have been introduced in Chapter 4 and Chapter 5. The EC contains two parameters describing the RTC offset ($\omega_{0,u}$) and skew ($\varphi_{0,u}$). If the exporter is properly synchronized, both parameters are close to zero. Thus, these parameters are configuration-dependent, i.e., they depend on whether time synchronization mechanisms are properly configured and running. On the contrary, in case of a non-synchronized RTC, these parameters reflect the skew of the RTC, which depends on characteristics of the *electronic components*, e.g., quartz oscillators, and differs between exporters. The following three parameters of the EC specify the resolution of the RTC and system uptime raw timestamps (ρ_{sec}, ρ_{nsec}, and ρ_{su}), which are implementation-dependent. The implementation-dependent property whether export times are aligned with system clock resolution or not (ESA) is specified by the Boolean ESA parameter.

The OPC contains the minimum ($b_{\text{p,min}}$) and maximum ($b_{\text{p,max}}$) packet size observed on this OP. These values are important for detecting byte count errors (see Section 4.3.2.1). They are implementation-dependent, since they reflect errors in byte count values, as well as configuration dependent, since the maximum and minimum packet size depends on the network technology of the interfaces. Furthermore, the OPC contains all clock related parameters that can depend on the metering process, such as the system clock skew $\varphi_{0,y}$. This parameter depends on implementation, i.e., whether there is a separate system clock for raw timestamps at all, or whether timestamp values are always taken directly from the RTC. If there is a separate system clock, its skew against the RTC depends on electronic components. The OPC parameters ρ_{first} and ρ_{last} specify the resolution of the corresponding raw record timestamps. These resolution parameters depend on the implementation of the metering process. The last OPC parameter $\lambda_{R,\text{max}}(L, T_{\text{j}})$ gives the maximum record rate per window size and time of day, which depends on configuration (timeout settings), and on traffic.

The third component of the Exporter Profile, the OPPC, first specifies the OPP by giving the opIDs of its two OPs in correct order. These two parameters therefore describe the path between two OPs from which flow data can be matched for OWD calculation. Since the OPP depends on network topology, these two parameters are configuration-dependent. The last parameter $h_S(J_i)$ is a discretized version of the per-OPP export jitter PDF for OWD samples. It depends on the timeout settings (configuration) of both OPs as well as on the traffic characteristic.

This section gave an overview of exporter profile parameters and the structure of the exporter profile. How these parameters can be obtained is discussed in the following sections.

6.3 Approaches for Profile Creation

Up to now the exporter profile concept and its parameters have been introduced. How these parameters can be obtained is considered in this section. The following list details and compares five different *profile creation approaches* to obtain profile parameters.

- **Profile parameters provided by device manufacturers**

  The manufacturers of flow capturing devices could provide detailed descriptions of device properties, e.g., in machine readable standardized formats. This could include design and implementation specific properties that are valid for a certain device model or certain hardware/software revisions of a device. Additionally, the manufacturer could also provide parameters that are not consistent across models or revisions in a per-device fashion (e.g., stored in each device or provided in lists based on device serial numbers).

  While there are first approaches of standardized formats for accuracy description (e.g., in [RFC 5477]), such information is not available for today's devices and datasheets also do not specify such parameters. Even if in future such information will be available, there will still be old devices in place where such information cannot be obtained for and other approaches have to be used. With respect to parameter dependency classes, information from device manufacturers may contain implementation and component-dependent parts. However, configuration- and traffic-dependent parameters still have to be obtained from other sources.

- **Profile parameters from laboratory measurements**

  By setting up laboratory measurements, each flow capturing device's parameters could be obtained by systematic measurements in a defined environment. With software updates, however, parameters might change. Especially, since updates of device software might also include new microcode images for Network Processors or images for configurable hardware components, such as *Field Programmable Gate Array* (FPGA) components. Hence, each software update could change low-level functionality like timestamp creation and therefore demand new measurements.

  Implementation-dependent profile parameters could be obtained easily with high accuracy using such lab measurements. In terms of component-dependent parameters, measurements per device are necessary, i.e., each device would have to be taken to a measurement lab. In large networks on a global scale, this would be a huge effort since each device might have to be transported to a laboratory first and at least cannot be used during the measurement in the production network. Furthermore, new lab measurements are necessary after each software update. Configuration and traffic dependent parameters cannot be obtained using solely this approach.

- **Profile parameters from configuration databases or provided by network administrators**

  In large networks device configurations are often stored in a central configuration database. Using such a database would allow for obtaining configuration-dependent parameters. However, care has to be taken that this information is always consistent and no manual effort has to be spent in case of missing information.

Table 6.2: Suitable profile creation approaches for different parameter dependencies

		parameter dependency class			
		implementation	component	configuration	traffic
profile creation approach	manufacturer	X	X		
	lab measurement	X	X[1]		
	config DB/admin			X	
	active measurements	X	X	X	
	flow data	X	X	X	X

[1] high effort: each device has to be taken to lab

Since this approach can only deliver configuration-dependent parameters, it has to be combined with other approaches in order to obtain the parameters of other dependency classes.

- **Profile parameters from active measurements as reference**

  An approach similar to the laboratory measurements, but with lower effort, is to perform active measurements in the network, which have a high accuracy and compare these measurements with results obtained from flow data. This requires active measurements from the same paths where flow based measurements will be performed.

  Compared to the laboratory measurement approach, this approach allows each device to stay in operation in the network. Additionally, such measurements could be simply repeated in case of software updates in order to validate parameters. Using reference measurements in the network allows for obtaining configuration-dependent parameters in addition to the implementation- and component-dependent parameters. For obtaining traffic-dependent parameters, flow data has to be evaluated separately.

  One problem of this approach is that for each OPP a comparable active measurement is required. While setting up measurements between exporters is feasible, the results from currently available active measurement technology are often not directly comparable to flow-based measurements. This is due to the fact that flow capturing and active measurement traffic are often handled at different positions in routers and hence no flow records from measurement traffic are available or there can be additional queuing delay between the positions.

- **Profile parameters from flow data**

  Using this approach, flow data is used for extracting the required parameters using suitable algorithms. It is a completely passive approach and can be applied to flow data that is captured for normal monitoring purposes of production traffic.

  This approach cannot provide as high accuracy as lab measurements or active measurements, but does not require extra measurements and results therefore in less effort. Determining parameters might require the processing of large amounts of flow data in order to obtain a certain degree of confidence. While flow data has to be used for obtaining traffic-dependent parameters, it is also possible to obtain parameters of the other parameter dependency classes.

As indicated in the descriptions of the profile creation approaches, some approaches have considerable drawbacks when it comes to implementation in production networks. The approaches relying on the configuration DB, active measurements, and flow data are the most promising ones. As a next step, the five profile creation approaches and the parameter dependency classes they can address are considered. The summary given in Table 6.2 shows that some approaches can obtain only parts of the exporter profile parameters and an approach based on flow data is always necessary in order to obtain the traffic-dependent parameters. Furthermore, flow data based creation can also be used to obtain other parameters, thus the profile parameters could be completely obtained by this approach without relying on other information. This makes this approach very attractive since it reduces the risk of errors that typically arise when combining information from several sources. In the following only the *flow data approach* is considered and the next sections focus on methods for flow data based profile creation and indicate limitations of this approach.

6.4 Profile Creation from Flow Data

This section introduces the approach to systematically create exporter profile parameters from flow data only, i.e., with as little as possible additional knowledge. The first subsection highlights the overall concepts by describing the iterative profiling steps and the methods applied for obtaining profile parameters. Profiling steps that require more detailed consideration of mechanisms and algorithms are detailed further in separate subsections.

6.4.1 Profiling Concept: Steps and Dependencies

The exporter profiling method uses reference flow data sets to obtain profile parameters in an offline fashion. Reference data sets consist of flow data from one day and provide enough data for reliable detection of profile parameters and daily behavior. Offline profile creation allows for using processing intensive algorithms that are not feasible for online processing while performing OWD extraction or dynamic profile creation. Since the profile parameters are used to support OWD measurement at a later point in time, the reference data set should be chosen in a way that it contains flow data collected from the same network topology and flow capturing components for which OWD measurement will be performed. If the topology or components change, a new reference data set can be chosen to obtain updated parameter sets.

Exporter profiling is a seven step iterative process, where steps partly depend on parameters obtained in previous steps. Figure 6.1 gives an overview of the seven profiling steps and the profile parameters obtained. Symbols used in this figure refer to profile parameters that have been defined in Table 6.1. The figure shows on the left pre-processing blocks that perform filtering and compensation actions before the flow data is fed into the profiling blocks (thick blue arrows). Resulting profile parameters are stored in the exporter profile database, shown on the right side. During the process, some of the parameters obtained in early steps are required for pre-processing as well as in the profiling blocks, as indicated by the thin orange arrows in the figure. Profiling steps are not always dedicated to a certain profile component, but similar parameters of different profile components might be obtained in the same step (e.g., resolution parameters of the EC and OPC are determined in the same step).

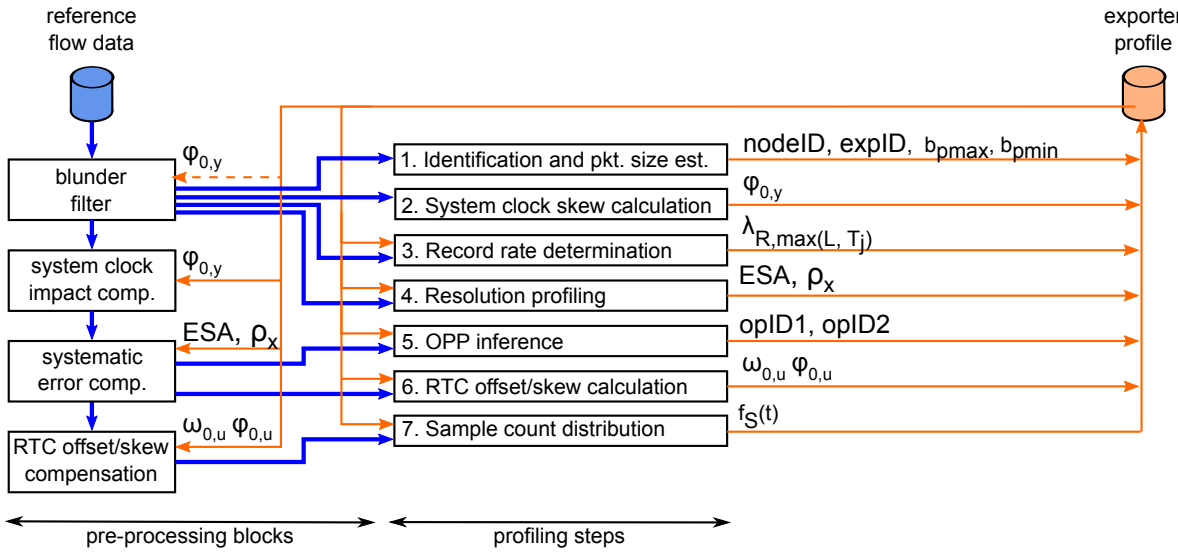


Figure 6.1: Profiling steps for exporter profile creation

In order to save memory or allow for parallel or iterative processing, some steps are further subdivided into sub-steps. Furthermore, for the same reasons processing steps can be performed on only parts of the data, e.g., for certain OPs/OPPs only. This requires pre-filtering of the respective data parts in the pre-processing block, which is not shown in the figure.

The reference data set is the complete flow data from one weekday of the considered enterprise network. Some processing steps can provide better results if data from times where the network is lightly loaded only is taken. A weekend data set can be used in such cases, which is highlighted for the respective profiling steps in the following.

In the following the seven profiling steps are detailed.

Profiling Step 1: Identification and packet size estimation

The first step identifies all exporters and OPs from the reference data set and sets up the respective EC and OPC data structures of the profile. Along with this identifcation, estimates of the minimum and maximum packet sizes are obtained for determining the upper and lower bounds of byte count error (Section 4.3.2.1).

The largest and the smallest packet size can be directly obtained from records that report one packet only. However, due to traffic characteristics and timeout mechanisms, such records may not exist, especially not for maximum sized packets that are very unlikely to occur as single packet of a flow. Therefore, the packet sizes are estimated for each OP based on all records $\mathscr{R}_{\mathrm{OP}}$ observed at this OP. For each OP the minimum packet size $b_{\mathrm{p,min,OP}}$ and maximum packet size $b_{\mathrm{p,max,OP}}$ are calculated based on the overall minimum/maximum packet size estimate of each record:

$$b_{\mathrm{p,min,OP}} = \min_{r \in \mathscr{R}_{\mathrm{OP}}} \left\lfloor \frac{b_r}{p_r} \right\rfloor \tag{6.1}$$

$$b_{\mathrm{p,max,OP}} = \max_{r \in \mathscr{R}_{\mathrm{OP}}} \left\lceil \frac{b_r}{p_r} \right\rceil \tag{6.2}$$

Using the second estimate, the largest packet size is only correct if there are flow records that report flows with almost only maximum-sized packets. Evaluations based on enterprise network flow data of a complete day showed that the maximum packet size can be properly estimated in this way.

This step also performs more general statistics on records, packets and bytes reported from OPs. These statistics are not part of the profile itself, but are used for determining how much data is available for each exporter and OP in the reference data set. If the amount of data is too low or there are only very few time intervals in the reference data set where data is available, these OPs and exporters are dropped and not considered in the following steps, since reliable determination of parameters is not possible in such cases.

The output parameters of this step are the generated IDs nodeID and expID for the EC and OPC as well as the parameters $b_{\mathrm{p,max}}$ and $b_{\mathrm{p,min}}$ that are contained in the OPC. Step 1 is the only step that does not need any parameters of the exporter profile as input. The blunder filter in flow data processing is configured with static plausibility check rules only and cannot detect boot time discontinuity blunder (see Section 4.4.4), which has no impact on this basic step.

Profiling Step 2: System clock skew calculation

This step calculates the skew between the system clock and the RTC for each OP, based on the list of OPs for which profile components have been previously instantiated. The system clock skew can be calculated without applying any preprocessing blocks for error compensation. This is due to the fact that the system clock skew is defined as skew against the RTC of the exporter and not as skew against UTC. Additionally, the system clock skew can be calculated for each OP independently of other OPs, since the calculation is possible based on flow data from a single source. Thus, no cross-OP considerations or flow matching is necessary. System clock skew is calculated by linear interpolation of the boot time values from records in the reference data set. The output parameter of this block is $\varphi_{0,y}$ for each OPC. The system clock offset $\omega_{0,y}$ could also be calculated, but this parameter is not required for the exporter profile.

Profiling Step 3: Record rate determination

The record rate serves as input for window size dimensioning and is determined per OP in this step. This step takes only records into account that passed the blunder filter after boot time discontinuity check and other checks. In this way exactly the record rate that also would be used for OWD extraction in online processing can be obtained. Since this profiling step can use timestamps of record arrival at the collector, RTC offset or skew detection and compensation is not necessary as input data. Record rate determination is further detailed in Section 6.4.2.

Profiling Step 4: Resolution profiling

In this step the resolutions of the raw timestamps and record timestamps are determined, as well as whether ESA holds or not. These parameters are important for quantifying systematic errors and are thus the basis for pre-processing steps that compensate such

errors in following profiling steps. The impact of system clock skew on timestamp values is not compensated prior to resolution profiling, since this would destroy the original timestamp values and could hamper resolution detection. Thus, the only flow data pre-processing for this step is the blunder filter. The algorithms used in this step are presented in Section 6.4.3.

Profiling Step 5: OPP inference

Up to now, only parameters per exporter (EC) or per OP (OPC) have been considered. In order to determine path characteristics, the paths (OPPs) have to be determined first, which is done in this step. Similarly to step 1, only OPPs are considered for which a certain amount of data is available. Section 6.4.4 describes the algorithms used for this step in more detail. The parameters created by this step are opID1 and opID2 that describe the OPP. Since the OPP inference is timing based, it is important that all systematic errors on timestamp have been compensated in pre-processing steps. Due to the algorithm's properties, RTC offset or skew does not impact the results. Thus, RTC offset/skew compensation is no prerequisite.

Profiling Step 6: RTC offset/skew calculation

This step checks whether RTC timestamps of exporters exhibit skew or whether they are properly synchronized to UTC. In case of non-synchronized clocks, it tries to quantify offset and skew. This enables skew compensation in pre-processing blocks.

Skew and offset determination requires a reference clock, which is synchronized to UTC. Determining offset and skew solely based on flow data means that a properly synchronized reference exporter is required, from which offset and skew to other exporters can be calculated. Such an exporter can be, e.g., a router which resides in a data center where it is directly attached to GPS-based clock synchronization mechanisms. The ExpID of this exporter has to be given as input to the profiling step. Regarding the overall profiling approach, this profiling step is the only one that requires external information.

RTC skew detection is based on Paxson's algorithm that has been introduced in Section 2.3.4. For clock skew calculation the difference between reported export timestamp and reception time at the collector is taken. Clock offset is calculated from OWD samples obtained from the OPPs between the profiled and the reference exporter. Reliable RTC offset calculation requires symmetric routing for these OPPs. This step benefits from using a weekend data set where due to the low traffic queuing delays are small.

If an RTC skew $\varphi_{0,u}$ of less than 10^{-6} (1 ppm) and an offset $\omega_{0,u}$ of less than 5 ms results, it is assumed that exporter clocks are properly synchronized. Experience with flow data from an enterprise network [3] shows that offset values are several orders of magnitude higher in case of unsynchronized clocks. This is due to the fact that typically clocks run unsynchronized for a longer time and hence such larger offsets results. If an exporter just lost clock synchronization, this can hardly be detected by the profiling step and if it happens after profiling, it cannot be detected at all. This is a limitation of this approach and can be addressed by adding corresponding mechanisms (see Section 2.3.4) to the online OWD calculation mechanisms.

Profiling Step 7: Sample count distribution

The last step obtains $h_S(J_i)$, which gives the amount of samples for different per-OPP export jitter values. Since this requires the extraction of OWD samples, this step requires

all filtering and compensation parameters for the respective pre-processing steps shown in Figure 6.1. This step uses the integrated delay extraction processing block presented in Section 5.2.3 extended by functionality that adds the per-OPP export jitter value J to each extracted OWD sample. Based on this data a histogram is calculated and stored in an array. The sample count distribution is input for window dimensioning.

As highlighted, the exporter profiling process requires several different processing blocks. Some pre-processing blocks are necessary in several processing steps. In order to allow for reusability and flexibility, the profiling process has been implemented based on the flexible flow processing framework developed in the context of this thesis [5].

6.4.2 Record Rate Determination

The record rates that are determined in the third profile creation step are input for the profile-based window dimensioning method (Section 5.3) of the online processing approach. In order to find a suitable window size, the record rate for different record window sizes L is required. Furthermore, this record rate is obtained for different time intervals of a working day T_j, thus the record rate is a two-dimensional array that contains a record rate value for each window sizes and working day time interval: $\lambda_{R,\max}(L,T_j)$.

For obtaining the record rate, records are filtered for the considered OP and forwarded to rate calculation processing. Depending on memory and processing power the rates can be calculated for each OP sequentially or in parallel if several rate calculation functional blocks are used.

Rate calculation is based on record arrival times at the collector and is, thus, independent of timestamp errors or compensation. A straightforward rate processing approach is using a sliding window for each window size L that is considered and get the maximum amount of records that arrive within the window in the considered time interval. However, using a sliding window is processing intensive and so it was impossible to use this approach with typical data set despite this task is not time critical in offline processing.

An alternative to a sliding window is a jumping window, i.e., there is one window after the other for which records are counted. However, packet bursts that arrive at window boundaries will not be captured by this mechanism. A solution to this problem is using windows of length $\frac{L}{2}$ as illustrated in Figure 6.2. By summing up the number of records that arrive in two adjacent windows, the record arrivals within a window of size L are calculated. Hence, overlapping windows result and by taking the maximum within the considered time interval the maximum arrival rate can be estimated. This approach is only an estimation compared to the sliding window method, but the accuracy proved to be sufficient. Furthermore, the estimation accuracy can be improved by using a higher number of test windows, e.g., of length $\frac{L}{4}$.

Record rate determination is performed for different window sizes, such as for $L = 1..20$ s. Using test windows of $\frac{L_{\min}}{2}$ allows calculating the rate of higher window lengths by summing up the arrivals of the respective amount of test windows. This is another advantage compared to the sliding window approach that requires one sliding window of each window size that is considered.

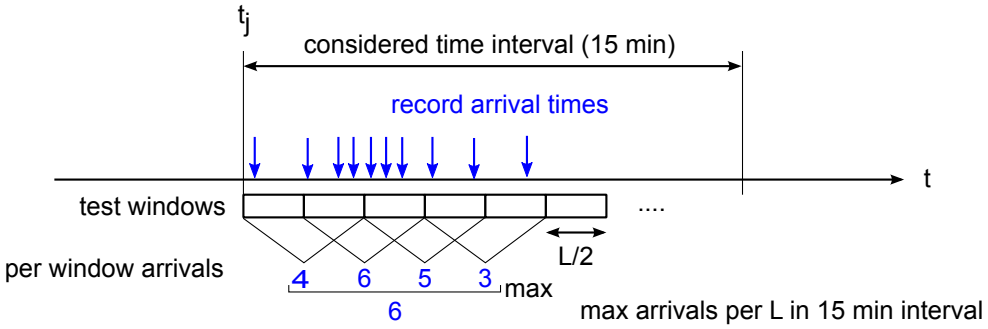


Figure 6.2: Record rate determination

6.4.3 Obtaining Timestamp Resolutions

Timestamp resolutions play an important role in the exporter profile for quantification of systematic and random errors. The exporter profile contains resolution parameters for several raw timestamps that need to be properly detected. Timestamp values that are input to the resolution detection are taken from the reference data set that contains flow data from a production network. Hence, they depend on traffic as well as on exporter characteristics and no pattern or characteristic must be assumed in the resolution detection profiling step. Especially, raw timestamps with adjacent values may not exist and resolution outliers as well as unaligned resolutions from hidden clocks (introduced in Section 4.4.3.1) are further challenges.

Resolution estimation is a known problem in network measurement and was recently addressed by Arlos and Fiedler [124, 125]. They devoloped methods for estimating the timestamp resolution of measurement hardware and software tools. However, the methods require a laboratory setup and are based on artificial high-volume traffic where packets are sent back-to-back. An alternative method is based on highly accurate and synchronized clocks of the packet generator and the system under test. Both methods could be used for the profile creation approach that is based on laboratory measurement, which was discussed in a previous section. However, they are infeasible for resolution detection based on flow data from a production network because of the missing control over traffic and lack of highly accurate and synchronized clocks.

For the given resolution detection problem with its special properties no suitable algorithm has been found. Simple approaches for resolution detection, such as calculating the greatest common divisor or testing the remainder of divisions with different resolutions values (modulo-based algorithms), proved not to work reliably. This is due to the resolution outliers and unaligned resolutions from hidden clocks in NetFlow data (introduced in Section 4.4.3.1). Hence, an algorithm based on the *Discrete Fourier Transform* (DFT) and heuristic peak detection has been developed that obtains the resolutions of the different timestamps. Resolution detection is related to the problem of estimating the fundamental frequency (also called F0 or pitch) in audio signals. The cepstral method for pitch detection in speech signals (evaluated in [126]) is similar to the method presented in the following. It is based on a DFT and a heuristic peak detection algorithm. However, audio signals are different from time series presented here (e.g., there are zero crossings) and therefore such methods for audio signals are not directly applicable.

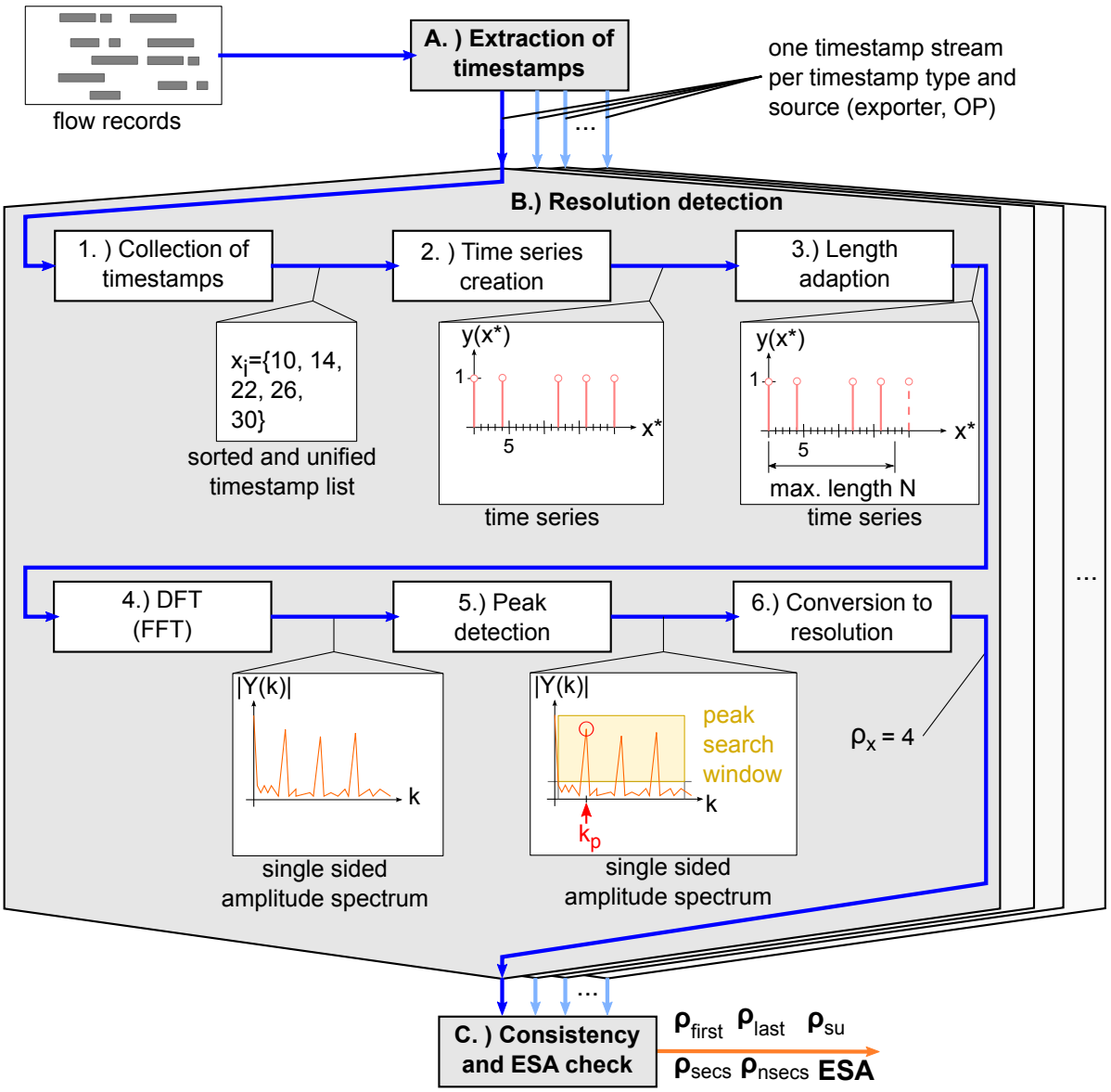


Figure 6.3: Resolution profiling procedure

The method developed in this thesis is shown in Figure 6.3. The resolution profiling procedure consisting of steps A,B,C obtains all resolution parameters. Step B is the resolution detection itself, which is further subdivided into six stages.

The first step (step A) takes records of the reference data set and extracts raw as well as record timestamps. Furthermore, timestamps are classified by source, i.e., per exporter or OP depending on whether the resolution parameter belongs to the EC or the OPC of the exporter profile. Per timestamp type and source the timestamp streams are forwarded to a separate resolution detection algorithm (step B) that calculates the resolutions ρ_x. Step C finally checks the timestamps of the same timestamp source for consistency and performs a check on whether ESA holds. Resolution values are consistent, if record and raw timestamps have the same resolution (e.g., resolutions of start and end timestamps (ρ_s, ρ_e) have the same value as record duration

(ρ_{d}) and raw resolutions (ρ_{first}, ρ_{last})). If resolutions are consistent, they are put into the exporter profile, otherwise an error will be raised. Export time and system time are assumed to be aligned if the resolution of the export timestamps ρ_{x} and the resolution of the raw system time ρ_{su} are the same. In such cases the ESA flag is set to true.

In the following the resolution detection step B is described in detail. This resolution detection algorithm consists of the following six stages:

Stage 1: Collection of record timestamps

This stage collects the timestamp values in a data structure that removes duplicates and also sorts timestamps on insertion. If 100,000 samples are present in the data set or 10 million timestamps have been received, the sorted timestamp list is forwarded to the next stage.

Stage 2: Time series creation

All timestamp values are converted into a time series with a y-value of one where a timestamp exists and a y-value of zero otherwise. The time series is aligned at zero by subtracting the value of the first sample from all x-values. The resulting x^* values as well as the $y(x^*)$ values of this time series are forwarded to the length adaptation stage.

Stage 3: Length adaptation

The time series length is adapted to the length N that is a power of two, which is a requirement for FFT. Due to counter wraps and other effects the time series could be very long and lead to high processing effort in the next stage. Furthermore, the length cannot be known while collecting samples at stage 1 and therefore has to be reduced in this stage. This stage reduces the time series to a predefined maximum (2^{20} in the current implementation).

Stage 4: Discrete Fourier Transform

On the adapted length time series a DFT is applied to transform the time series into the frequency domain. The DFT is implemented as *Fast Fourier Transform* (FFT) for efficiency reasons. From the complex frequency spectrum a single sided amplitude spectrum $|Y(k)|$ of length N is generated for further evaluation.

Stage 5: Peak detection

The amplitude spectrum contains peaks for frequency components found in the time series. In general, the first peak with $k > 0$ denotes the fundamental frequency, which indicates the resolution of the timestamps. However, fundamental frequency detection and hence peak detection requires more sophisticated algorithms since peaks often have a certain width and there is a wide range of possible resolutions. A suitable peak detection algorithm that reliably detects the peaks is detailed in Appendix D. It returns the position of the peak k_{p} .

Stage 6: Conversion to resolution

In the last step the k_{p}-value is converted into the resolution value ρ of the considered timestamp using the following equation that results from DFT properties: $\rho = \frac{N}{k_{\mathrm{p}}}$. If $k = 0$

no fundamental frequency is found, which means that there are no resolution effects and thus $\rho = 1$ is returned.

As highlighted, the resolution detection algorithm starts as soon as a certain amount of records or samples has been collected. The profiling step can stop as soon as enough data was available for running the resolution detection for all timestamp types and source classes. Hence, not the whole reference data set needs to be processed. At the end, the resolution values as well as the ESA flag are written to the exporter profile.

6.4.4 Observation Point Pair Inference

For OWD measurement, as well as for extracting profile parameters, it is important to know the order of OPs in an OPP, i.e., which OP is traversed first and which second. This cannot be determined ad-hoc based on OWD samples created from flow data for two reasons. First, a certain fraction of OWD samples on correct paths can be negative due to random errors, especially if the range of the error distribution is wider than the true OWD value. Second, there might be unsynchronized exporter clocks. This does not allow using OWD samples for determining which OP is traversed first and which second. However, quantifying clock skew requires knowing topology and thus knowing OPPs. Hence, OPPs are inferred in profiling step 5, i.e., before RTC skew calculation is determined in profiling step 6.

The OPP inference algorithm takes properties of traffic into account, especially the property that a forward flow causes a reverse flow due to the request-response nature of most protocols. For each exporter passed by both flows, RTT values are calculated. Thus, no timestamps across exporters have to be compared, which relieves from the problem of unsynchronized clocks. Furthermore, detection of forward and reverse flow does not take knowledge about server port numbers into account. Such an approach is often not sufficient, e.g., if non-well-known port numbers are used. Additionally, the algorithm handles several OPPs on a path at once.

Figure 6.4 depicts an exemplary scenario of three exporters and shows how the algorithm is applied. In this example client A sends a request (forward flow) to server B, which sends the corresponding response (reverse flow). At each exporter, the RTT can be calculated from flow timestamps of forward and reverse flows that traverse the OPs. The ordering of exporters and hence also the OPs on the path is possible by sorting OPs according to the RTT values obtained. Due to random errors, however, RTT-based ordering will not be the same for all pairs of forward and reverse flows, but a certain amount of RTT values has to be collected for averaging out random errors and for achieving stable results. The algorithm performs a data collection run on a flow data set, which contains all one-record-flows of the data set. After the collection it determines the correct OP ordering based on the large amount of RTT samples considered, which enables to eliminate the random errors.

For each flow key tuple, the following steps are performed and eventually a set of OPPs is returned:

1. Get all flow records of the selected tuple as well as all flow records of the reverse tuple.

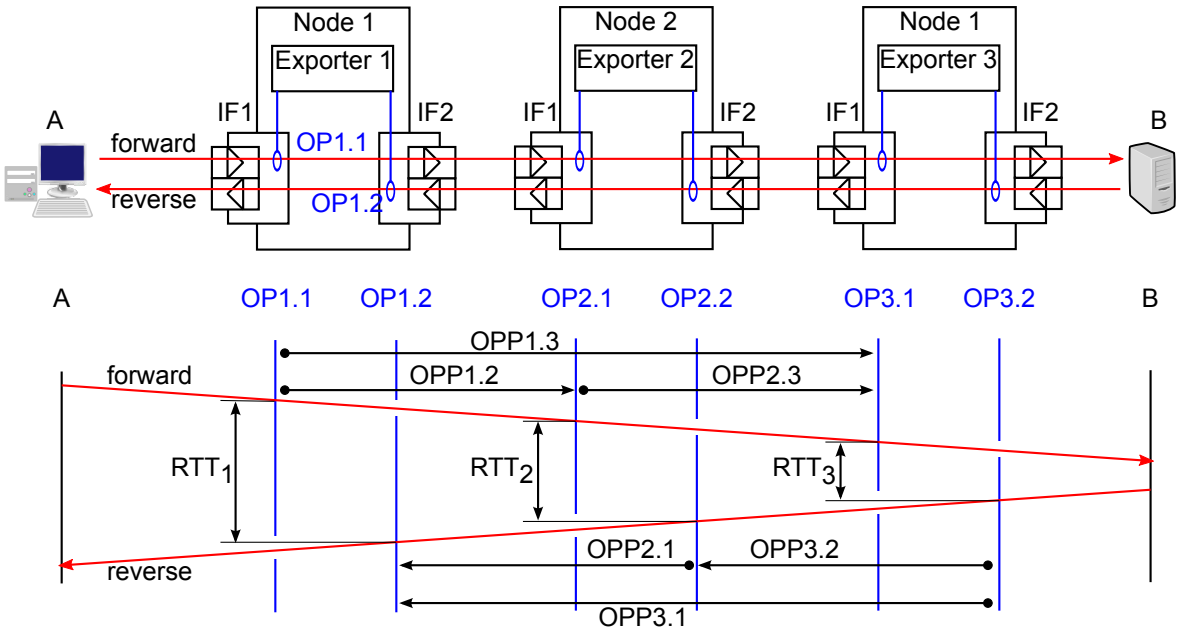


Figure 6.4: OPP inference

2. Add all OPPs to the OPP result set that are not yet contained. OPP ordering is initially defined based on opID ordering and will be redefined later.

3. Calculate RTT values for all exporters based on the record start timestamps that report the forward and the reverse flow.

4. Take the RTT value with the largest absolute value and use the timestamps of this RTT value to (re)define forward and reverse flow and change the sign of RTT values obtained if necessary.

5. Order the exporters according to RTT values and increment for each OPP a counter whether its current opID-based OP ordering is correct or not.

6. Remove all flow records with the tuple or reverse tuple considered from the data set. Thus, neither forward nor reverse tuples will be considered in the following iterations.

After the collection run, the result set is evaluated and for each OPP where the counter for correct ordering is lower than the counter for incorrect ordering, the OPs are switched. OPPs for which no distinct decision is possible or not enough data is available are removed from the result set. OPPs left in the result set are stored as part of the OPPC in the exporter profile.

This approach of OPP inference requires that flow data is generated for each flow that traverses the exporters, i.e., that flow capturing is enabled on the respective interfaces. Furthermore, it requires that the forward and reverse flow passes the same router, at least for a fraction of traffic, i.e., that there is symmetric routing. The first requirement can be achieved by enabling flow capturing on the respective interfaces. The second requirement is often fulfilled, however, if not it might require some effort to achieve. If any of the requirements cannot be fulfilled, the

OPP ordering information has to be taken from other sources, such as configuration DBs or from administrators. There is also the possibility of adding missing OPPs based on OPPs that could be inferred. This functionality has not been considered further, but it can be done manually at the current stage. Another limitation is that OPPs within an exporter cannot be inferred with the current algorithm. This happens if flows pass two flow capturing enabled network interfaces of the same router.

6.5 Summary

The exporter profile contains parameters that improve OWD measurement accuracy as well as processing efficiency. Parameters in the profile do not only depend on the flow capturing device implementation, but also on electronic components, configuration and traffic. This leads to the definition of parameter dependency classes based on which profile parameters can be classified. With the definition of the exporter profile these dependencies have been systematically evaluated and profile parameters have been grouped into the Exporter Component (EC), Observation Point Component (OPC) and Observation Point Path Component (OPPC). These components specify for which entities the parameters have to be obtained. For profile creation, five different approaches have been considered, each addressing a different set of parameter dependencies. Based on an evaluation of the five approaches the profile creation approach that is only based on flow data and covers all parameters has been chosen.

For flow data based exporter profiling a seven step profiling method has been developed, which iteratively determines profiling parameters. While some profiling steps are based on well-known algorithms or result directly from considerations of previous chapters, other steps required the development of more sophisticated methods. Respective approaches for record rate determination, timestamp resolution profiling, and OPP inference have been developed. The profiling method has some limitations under certain circumstances, e.g., if there is predominantly asymmetric routing.

The profiling method has been applied on data of a global enterprise network. The results that have been gained and the impact of profile parameters on OWD measurement accuracy and processing efficiency are considered in the next chapter.

7 Applicability and Evaluation

This chapter describes how the concepts introduced in the previous chapters were applied in an enterprise network scenario and presents evaluations of these concepts based on reference measurements. The concepts applied are error compensation and quantification concepts from Chapter 4, online processing approaches and dimensioning from Chapter 5, as well as the exporter profile and its creation from Chapter 6.

The first section presents the evaluation scenario and measurement setup for obtaining flow and reference data. In the following two sections, the evaluation of OWD measurement supported by exporter profiles is detailed. The second section focuses on errors and their compensation and quantification. The third section presents results and validations for the window dimensioning method. Both of the latter sections take exporter profile parameters into account that have been obtained based on the method introduced in Section 6.4. For the sake of better readability, not all profile parameters are listed at a single point, but right in the section where their impact is evaluated. The last section summarizes the evaluation results.

7.1 Evaluated Network Scenario

In order to validate and evaluate the flow-based OWD measurement approach, data from a global enterprise network was collected. This section details this evaluation scenario, the data collection process, and the properties of the data set.

The measurements in the enterprise network consisted of two components: active reference measurements and collection of flow data exported by the routers of the network. For detailed analysis, three locations in different countries have been selected, as shown in Figure 7.1. At the location in Germany, a measurement machine was connected to the network that performed active TCP-based RTT measurements with peer machines at the locations in France and Australia. The measurement machine also collected flow data (NetFlow v5) by means of a Flow-Tools [99] and patched NfDump [101] collector. Flow data was copied in the Linux kernel and sent to both collector processes, which wrote flow data in compressed file formats to two separate hard disks. In order to retrieve this data, a central collector of the enterprise network was configured to mirror the flow data it received to the measurement machine.

Active measurements and flow data collection ran from afternoon of Wednesday, 3rd November 2010 to noon of Monday, 8th November 2010. The NetFlow v5 data set contains flow data from several hundred routers and its overall size is 180 GB in compressed Flow-Tools format,

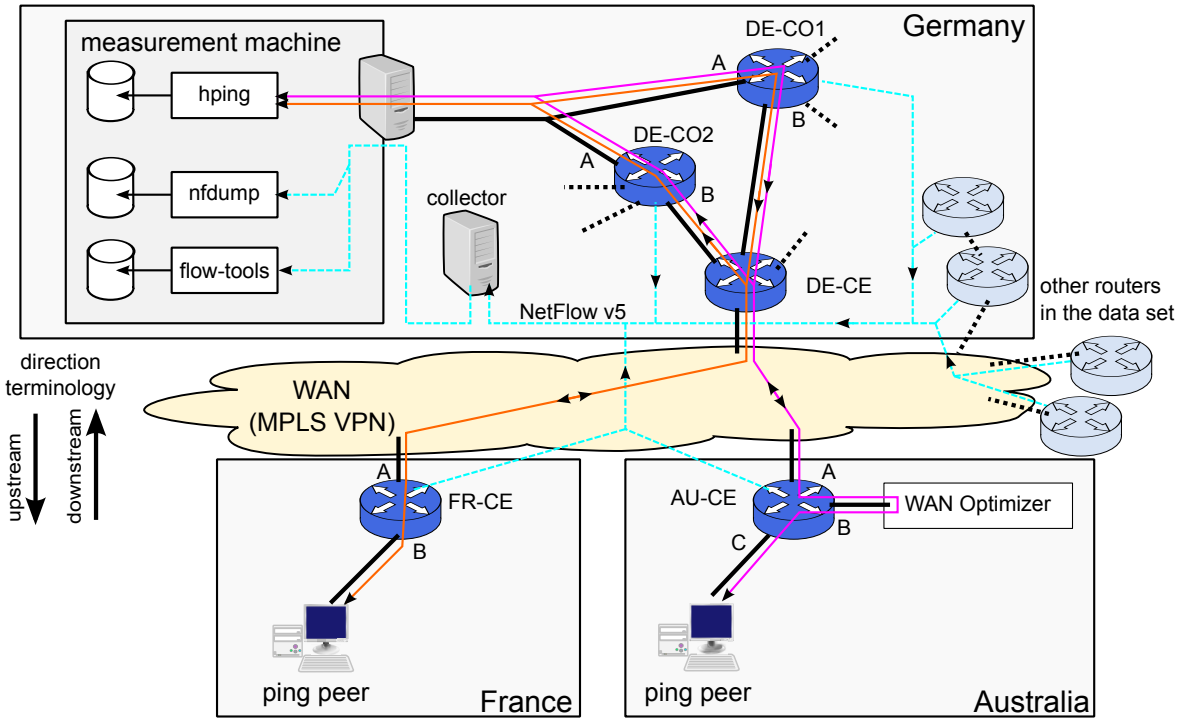


Figure 7.1: Evaluation scenario

or 786 GB uncompressed. From the exported flow data, the topology as shown in Figure 7.1 was derived and confirmed by the enterprise's network department. The flows from the active measurements traverse the MPLS CE routers at each location as well as two core routers (DE-CO1 and DE-CO2) at the German location with asymmetric routing. The interfaces traversed are indicated by capital letters (A,B,C). At the Australian location, the flows traverse the CE router twice due to a WAN optimizer connected to interface B of this router. There is also a WAN optimizer at the German location. However, there is no NetFlow data from the interfaces it is connected to and therefore this component is not shown in Figure 7.1. The same applies to all other routers and switches that did not export flow data. Interface B of the Australian location is not considered for OWD measurement.

Unless otherwise noted, results presented in this chapter are based on the Flow-Tools data set, since Flow-Tools stores the raw timestamps by default. NfDump data is used if the collection timestamp of records is required, which NfDump records due to a software patch applied. In the following, only the routers shown in Figure 7.1 and only flow records from UDP or TCP flows are considered unless noted otherwise.

7.2 Profile-based Error Compensation and Quantification

This section shows the applicability of the exporter profiling method (see Section 6.4) for compensation and quantification of errors in OWD extraction. It first presents the results of the profiling run and which error corrections or quantifications can be derived from the error-related parameters. Second, a comparison of active measurement data with OWD results obtained from

exporter	interface	$b_{\mathrm{p,min}}$	$b_{\mathrm{p,max}}$
DE-CO1	A	39 bytes	1500 bytes
DE-CO2	B	46 bytes	1500 bytes
AU-CE	A	21 bytes	1500 bytes
AU-CE	C	22 bytes	1500 bytes
FR-CE	A	25 bytes	1500 bytes
FR-CE	B	23 bytes	1500 bytes

Table 7.1: Minimal and maximal byte count of relevant observation points

NetFlow data shows that exporter profile usage leads to a considerable improvement of accuracy. The third part covers the impact of resolution effects on confidence interval calculation for OWD values and improvements that can be achieved by taking profile parameters into account. In the fourth part, the impact of system clock skew and its compensation is highlighted based on an additional evaluation scenario.

7.2.1 Error-related Profile Parameters

Exporter profile values were created based on data from the third measurement day (Friday, 5th November 2010). In the following, the profiling results for byte count, system clock skew, timestamp resolution, OPP inference, and RTC clock skew/offset are presented.

Byte Count Parameters

For record matching it is important to consider potential byte count errors (see Section 4.3.2.1). The values required for determining these errors are the minimum and the maximum byte count per packet ($b_{\mathrm{p,min}}$ and $b_{\mathrm{p,max}}$), which are obtained per OP in the first profiling step. As shown in Table 7.1, these byte count values differ for the different OPs in the scenario and the exporter DE-CO2 shows the 46-bytes limitation on interface B, i.e., it reports the Ethernet payload size instead of the IP packet size. Interestingly, also some interfaces of DE-CO1 show this behavior (not shown in the table), while other interfaces of the same exporter do not show the limitation. The latter is the case for interface A of DE-CO1, which is relevant for the evaluation scenario. This clearly indicates that byte count parameters have to be obtained per OP.

System Clock Skew

In the second profiling step, the system clock skew $\varphi_{0,y}$ (see Section 4.4.2.2) was determined for each OP . Table 7.2 shows $\varphi_{0,y}$ values for the OPs of the evaluation scenario. These values are consistent across OPs of the same exporter, which indicates that the considered routers use a single system clock for NetFlow timestamp generation.

The table shows that $\varphi_{0,y}$ varies in a certain range for the considered OPs. Additionally, the last column of the table shows the calculated systematic error e_{s} for the record start time caused

exporter	interface	$\varphi_{0,y}$	e_s for $\delta_s = 60\,s$
DE-CO1	A,B,C	$2.16 \cdot 10^{-6}$	0.13 ms
DE-CO2	A, B	$7.58 \cdot 10^{-6}$	0.45 ms
DE-CE	A,B,C,D	$1.12 \cdot 10^{-5}$	0.67 ms
AU-CE	A,B,C	$-1.63 \cdot 10^{-5}$	−0.98 ms
FR-CE	A,B	$2.61 \cdot 10^{-5}$	1.57 ms

Table 7.2: System clock skew $\varphi_{0,y}$ values and resulting theoretical start time error e_s for evaluated observation points

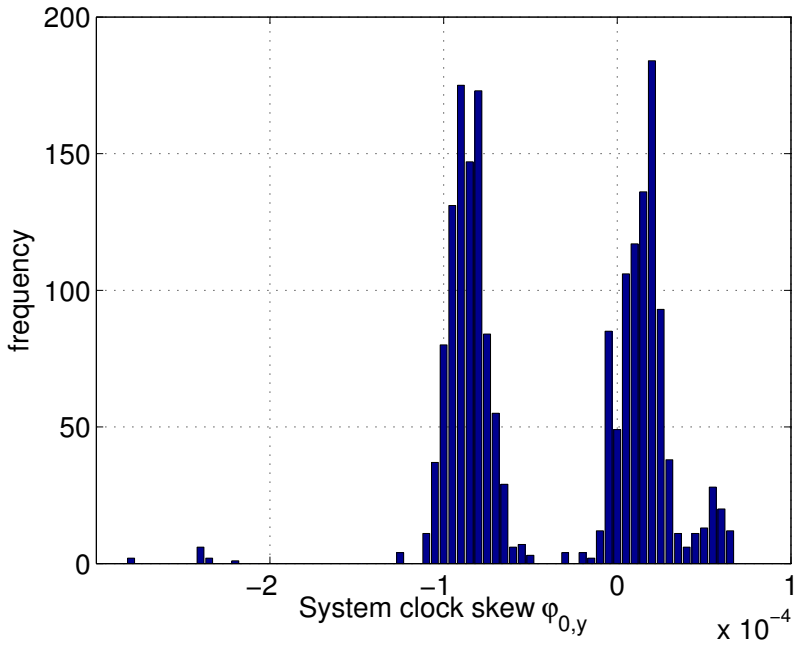


Figure 7.2: Histogram of system clock skew ($\varphi_{0,y}$) values from all OPs in the data set. Bin size $5 \cdot 10^{-6}$.

by $\varphi_{0,y}$ if the start time export delay δ_s is 60 s (a typical value for records expired by active timeout). As shown this error is up to 1.57 ms per OP in the evaluation scenario. For an OPP the overall error is the difference of both e_s values of the OPs. As an example this maximum error is therefore 2.55 ms for the OPP AU-CE/FR-CE in the evaluation scenario.

An evaluation of the $\varphi_{0,y}$ values of all OPs in the data set, however, shows that $\varphi_{0,y}$ values can be much higher than in the evaluation scenario. The histogram in Figure 7.2 shows that a significant number of OPs have $\varphi_{0,y}$ values of about $-1 \cdot 10^{-4}$ or $0.5 \cdot 10^{-4}$, which is up to five times as much as the values observed for the evaluation scenario routers. Consequently, there are OPPs where the error caused by $\varphi_{0,y}$ can be more than 10 ms. This shows that the impact of system clock skew is much more severe on routers that are not part of the evaluation scenario and that within the evaluation scenario system clock skew causes only small errors. Therefore, Section 7.2.4 will present a separate evaluation of system clock skew impact of an OPP with high system clock skew differences that is not part of the evaluation scenario.

exporter	ρ_{sec}	ρ_{nsec}	ρ_{su}	ρ_{su} aligned	ρ_x	ESA
DE-CO1	1.00 s	15,197 ns	4.00 ms	yes	4.00 ms	yes
DE-CO2	1.00 s	15,197 ns	4.00 ms	yes	4.00 ms	yes
AU-CE	1.00 s	15,197 ns	4.00 ms	yes	4.00 ms	yes
FR-CE	1.00 s	36,158 ns	999.6 ms	no	999.6 ms	yes

Table 7.3: Exporter Component (EC) resolution values of considered exporters

exporter	interface	ρ_{first}	ρ_s	ρ_{last}	ρ_e	ρ_d
DE-CO1	A	64.00 ms	64.00 ms	64.00 ms	64.00 ms	64.00 ms
DE-CO2	B	64.00 ms	64.00 ms	64.00 ms	64.00 ms	64.00 ms
AU-CE	A	4.00 ms	4.00 ms	4.00 ms	4.00 ms	4.00 ms
AU-CE	C	4.00 ms	4.00 ms	4.00 ms	4.00 ms	4.00 ms
FR-CE	A	4.00 ms	4.00 ms	4.00 ms	4.00 ms	4.00 ms
FR-CE	B	4.00 ms	4.00 ms	4.00 ms	4.00 ms	4.00 ms

Table 7.4: Observation Point Component (OPC) resolution values of relevant observation points

Resolution Parameters

The timestamp resolutions of raw and record timestamps are created in the resolution profiling step of the exporter profile creation (see Section 6.4.3). As some timestamps are created by the exporting process and other timestamps are created by the metering process at the observation point, the timestamps are either part of the *Exporter Component* (EC) or of the *Observation Point Component* (OPC) of the exporter profile.

Table 7.3 shows the EC resolutions of the considered exporters, which are UNIX time second (ρ_{sec}), nanosecond (ρ_{nsec}), and system uptime (ρ_{su}) raw timestamps resolutions. Furthermore, this table lists whether the system uptime resolution is aligned (see Section 4.4.3.1) and gives the resolution of the export timestamp (ρ_x). The last column shows whether ESA (see Section 4.4.3.1) holds. While ρ_{sec} is 1.0 s as expected, ρ_{nsec} differs between 15,197 ns and 36,158 ns. With both values, however, the export time has still a much higher resolution than the millisecond resolution of the other timestamps. Hence, a better resolution detection algorithm is not necessary. Tests with modulo-based detection methods or tuning of the FFT-based detection revealed an aligned resolution of 15,258 ns for all cases.

As shown ρ_{su} equals ρ_x for each exporter, hence, it is assumed that ESA holds (see Section 6.4.1). The exporter FR-CE has a very high ρ_x value of 1 s, which is reported as 999.6 ms due to rounding errors that do not have further impact. As for the other exporters, ρ_x equals ρ_{su} and ESA holds for FR-CE. However, ρ_{su} and ρ_x are both unaligned for FR-CE and the limited resolution could therefore not have been detected with simple (e.g., modulo-based) resolution detection methods. Since ESA holds for all considered exporters the exporter timestamp resolutions do not impact record timestamp resolution and hence do not cause any systematic or random timestamp errors.

Observation Point		bias	random error	
exporter	interface	β_s, β_e	σ_s, σ_e	τ_s, τ_e
DE-CO1	A	−32 ms	18.475 ms	64 ms
DE-CO2	B	−32 ms	18.475 ms	64 ms
AU-CE	A	−2 ms	1.155 ms	4 ms
AU-CE	C	−2 ms	1.155 ms	4 ms
FR-CE	A	−2 ms	1.155 ms	4 ms
FR-CE	B	−2 ms	1.155 ms	4 ms

Table 7.5: Calculated bias and random error parameters per OP resulting from limited resolution

Observation Point 1		Observation Point 2		bias	random error	
exp.	interf.	exp.	interf.	$\beta_{\Delta t_s}$, $\beta_{\Delta t_e}$	$\sigma_{\Delta t_s}$, $\sigma_{\Delta t_e}$	$\tau_{\Delta t_s}$, $\tau_{\Delta t_e}$
DE-CO1	A	AU-CE	A	30 ms	18.511 ms	68 ms
DE-CO1	A	FR-CE	A	30 ms	18.511 ms	68 ms
AU-CE	C	DE-CO2	B	−30 ms	18.511 ms	68 ms
FR-CE	B	DE-CO2	B	−30 ms	18.511 ms	68 ms

Table 7.6: Calculated bias and random error parameters per OPP resulting from limited resolution

Table 7.4 shows the values of the OPC raw and record timestamp resolutions. With 4.00 ms and 64.00 ms there are only two different values and resolutions are consistent in each row. Slight differences between the values are revealed only in the fourth decimal digit (not shown), thus all resolution values passed the consistency check. At some interfaces of DE-CO1 and DE-CO2, resolution values smaller than 64.00 ms have been obtained. This shows that per-exporter resolution profiling of these values is infeasible, but per-OP profiling as described in the exporter profiling method is necessary.

From the resolution limits reported in the OPC, the known systematic error (bias) and the random error for start and end timestamps per OP can be calculated. The results are shown in Table 7.5. The bias β, as well as the standard deviation σ and range τ of the random error are obtained according to the equations introduced in Section 4.4.3. Based on these per-OP parameters, the per-OPP parameters in Table 7.6 can be calculated. As Section 7.2.2 will show, the errors obtained by comparison with reference measurements are very close to the calculated errors.

Observation Point Pair Inference

The OPP inference profiling step (see Section 6.4.4) found the OPPs as shown in Table 7.7. The last column of this table shows the fraction of RTT values that indicated the given direction of the OPP. Although there is asymmetric routing in the evaluation scenario, three of the four pairs can be correctly identified. Apparently not all traffic between the OPs is routed asymmetrically

OP1 exporter	OP1 interface	OP2 exporter	OP2 interface	direction correct
DE-CO1	A	AU-CE	A	100.00 %
DE-CO1	A	FR-CE	A	50.70 %, 82.60 %
AU-CE	C	DE-CO2	B	99.98 %
FR-CE	B	DE-CO2	B	89.14 %

Table 7.7: Results of OPP direction inference

exporter	$\varphi_{0,u}$	$\omega_{0,u}$
DE-CO1	$1.53 \cdot 10^{-7}$	0.5 ms
DE-CO2	$1.24 \cdot 10^{-7}$	reference
DE-CE	$-2.76 \cdot 10^{-7}$	0.7 ms
AU-CE	$9.77 \cdot 10^{-8}$	−0.3 ms
FR-CE	$1.29 \cdot 10^{-7}$	0.5 ms

Table 7.8: RTC skew and offset

like the probe traffic, but some of it takes a symmetric route. The only OPP for which not enough symmetric traffic was available is the OPP between interface A of DE-CO1 and interface A of FR-CE. This OPP has been added manually based on the two OPPs detected between DE-CO1 and the edge router DE-CE, as well as between DE-CE and FR-CE. This is the reason why this row in the table shows two values for the correct direction, 50.70 % for the OPP between the routers in Germany and 82.60 % for the OPP between DE-CE and DE-FR. Since the geographical distance between DE-CO1 and DE-CE is very small and both routers have 64 ms timestamp resolution, detecting this OPP correctly is very hard. Nevertheless this detection was successful, however, only with a very low value of 50.70 %. Interestingly, most other OPPs between the DE-CE and DE-CO1/DE-CO2 routers have been detected correctly (not shown in the table). Strict OPP consistency checks could improve OPP inference in such scenarios. However, such checks are not considered in this work.

Real-Time Clock Skew

The sixth profiling step revealed the offset and skew values for the RTC that are shown in Table 7.8. Skew was determined based on the export and collection timestamps. DE-CO2 was chosen as reference exporter for offset calculation. For each exporter, the OPPs with most traffic from/to the reference exporter was selected for offset calculation.

Skew and offset values are very low, i.e., it is assumed that the clocks are properly synchronized. In terms of exporter profiling, all skew values are below the unsynchronized clock threshold of $1 \cdot 10^{-6}$ (1 ppm). The offset values are below the threshold of 5 ms.

7.2.2 Comparison with Active Reference Measurements

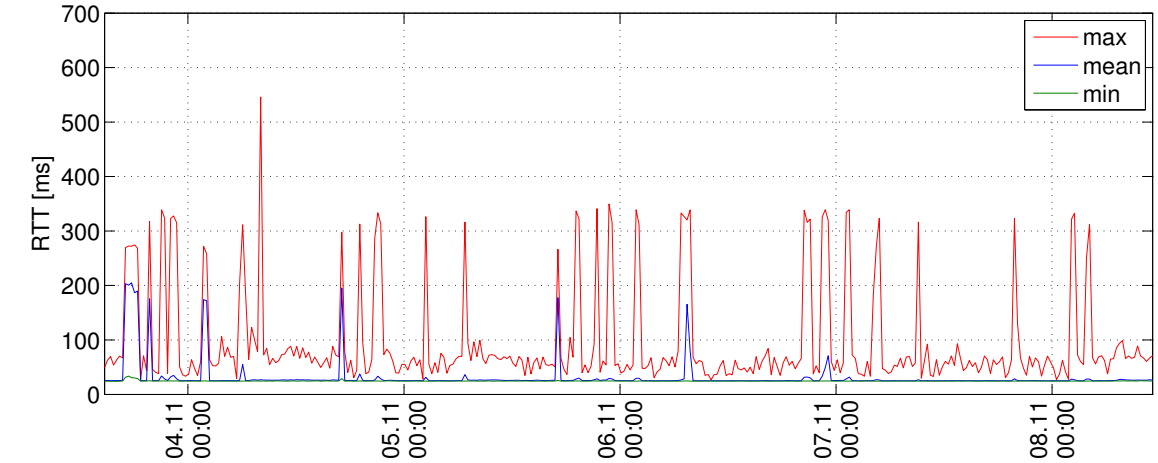
The accuracy of the flow-based OWD measurement approach was evaluated by comparison with active measurements. This requires that measurement samples from active and flow-based measurements that result from the *same packets* flowing through the network are available and can be compared. In principle there is the possibility of performing active OWD measurements for reference purposes. However, this would have required additional precisely synchronized equipment at each measurement location, which could not be set up in the enterprise network.

Another possibility is to rely on measurement results available from active measurement mechanisms for QoS monitoring that are already in place. Such a mechanism is IPSLA, which sends probe packets between routers and which was configured for some paths in the enterprise network. IPSLA OWD measurements have initially been taken into account for comparison [2]. However, IPSLA reports OWD as the mean value of samples obtained from several UDP packets. Since all UDP packets use the same five tuple, only one flow record is available from this active measurement. Thus, only two flow-based OWD samples could be compared to a mean value calculated from the OWD of each packet within this flow, which does not allow for a detailed evaluation. Furthermore, routers that send and receive IPSLA packets do not export flow records for these flows. This is another problem of the IPSLA-based evaluation approach, which limits this approach to paths with at least two flow capturing enabled routers between two IPSLA-enabled routers.

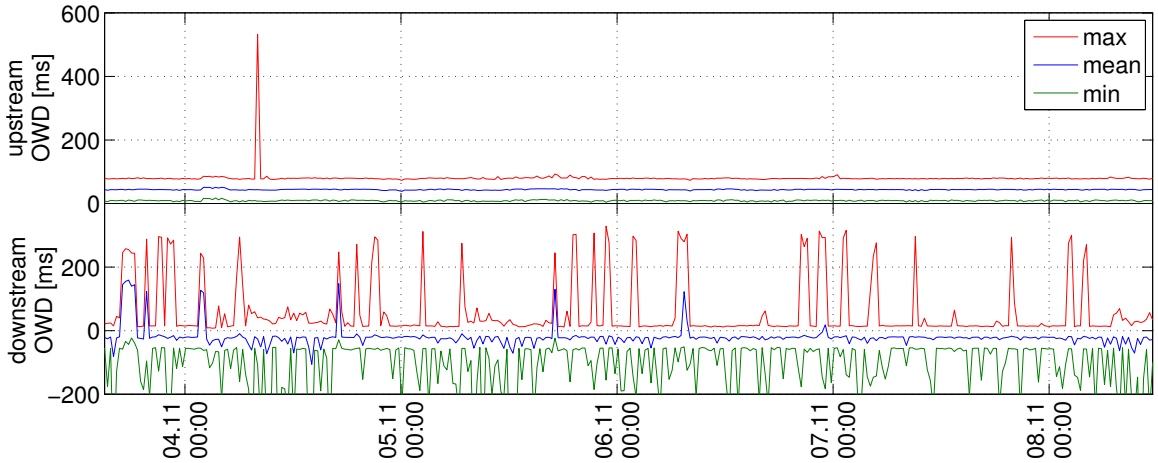
Since active OWD measurements could not be used for comparison, the evaluation of the accuracy of flow-based OWD measurement presented in the following is based on active TCP-based RTT measurements. For this purpose, the measurement machine introduced in the evaluation scenario (Figure 7.1) uses the hping tool [127]. It sent TCP ACK to the ping peers, which responded with a TCP RST. Since the ACK messages do not belong to an established TCP connection, they do not match a defined protocol state on arrival at the ping peer and are thus immediately answered by the ping peer OS with RST. The ping peers ran Windows XP with disabled firewall and had enough free CPU resources. While such measurements generally differ from ICMP-ping-based RTT measurements [128], using TCP pings is a common practice [129]. In both cases, i.e., ICMP or TCP pings, the OS kernel sends back the response immediately. During the measurements, the measurement machine sent every 20 minutes one burst of 1000 TCP ACK packets to each ping peer with a rate of 10 packets per second. TCP source port numbers have been changed sequentially within a burst while the destination port number was kept constant.

Each RTT sample obtained by TCP pings can be compared directly to two flow-based OWD samples, which allows to evaluate accuracy in terms of random errors and outliers. The quantification of systematic errors is limited, since systematic errors in upstream and downstream direction may compensate each other. Results of RTT, flow-based OWD without correction, and flow-based OWD with correction are provided below each other for better comparison in Figure 7.3 for the path Germany-France and in Figure 7.4 for the path Germany-Australia.

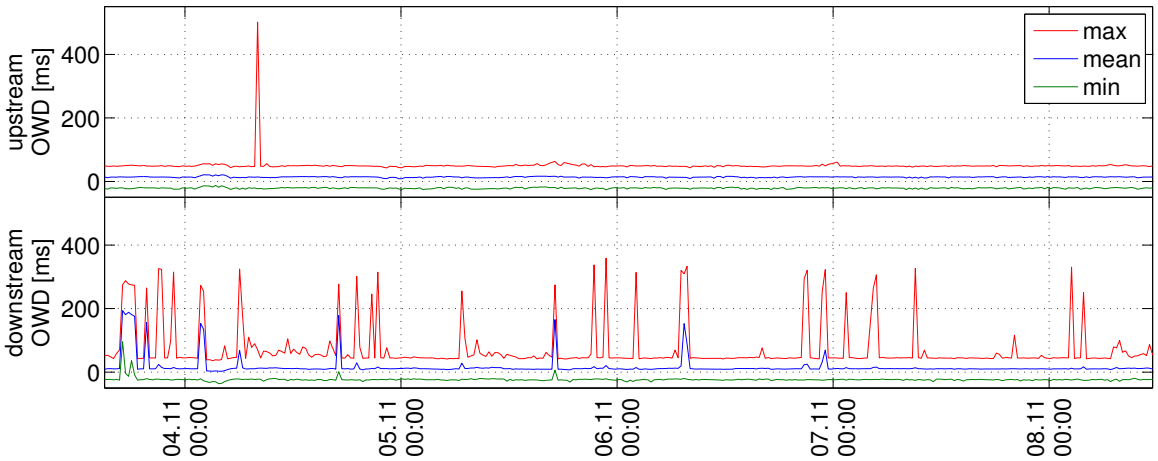
The results of the active reference measurements are presented in Figure 7.3(a) and Figure 7.4(a). For each burst the charts show the minimum, mean and maximum values. The RTTs for the path between Germany and France (Figure 7.3(a)) show a typical mean value of 24.5 ms with a maximum value of 546 ms. In some bursts, the mean value reaches 200 ms, which shows that



(a) Active RTT measurements

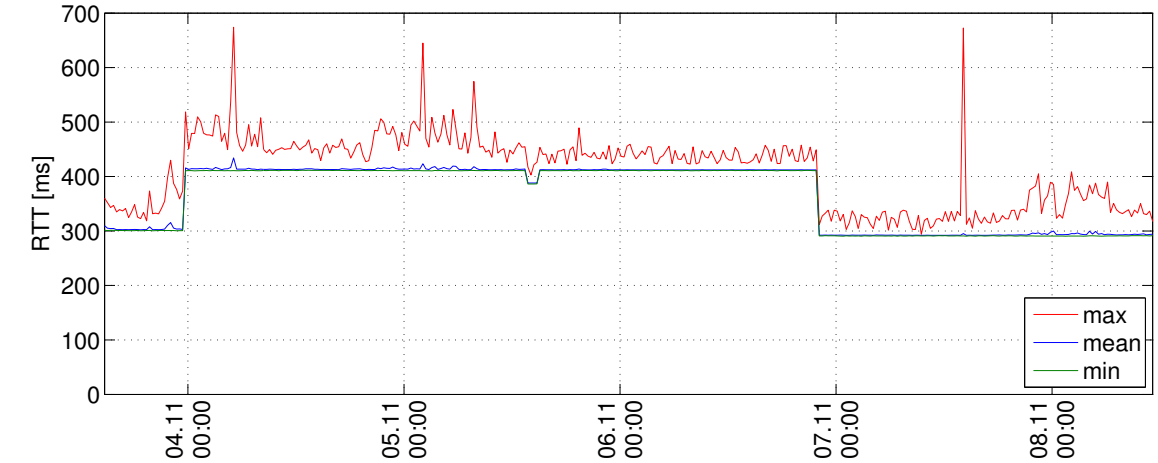


(b) Uncorrected flow-based OWD of RTT measurement traffic

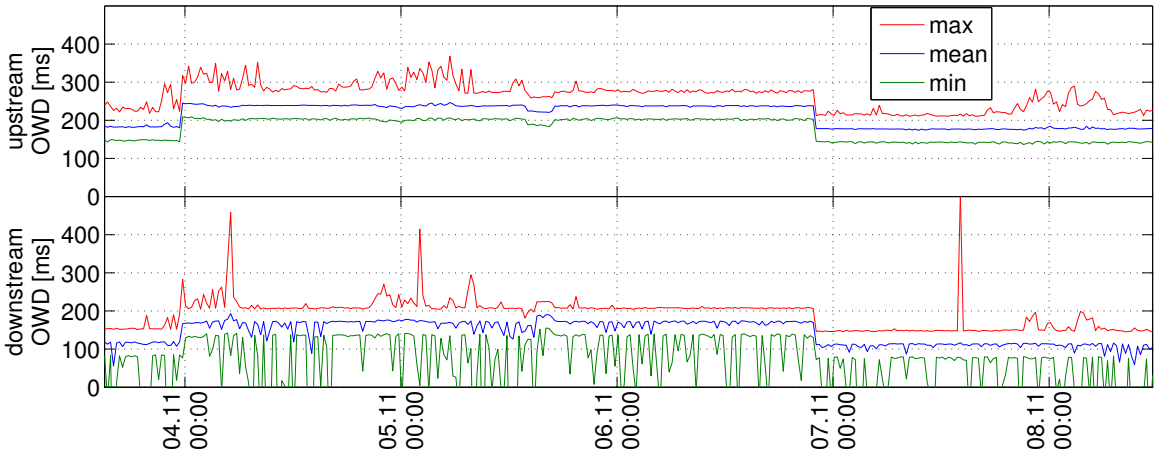


(c) Corrected flow-based OWD of RTT measurement traffic

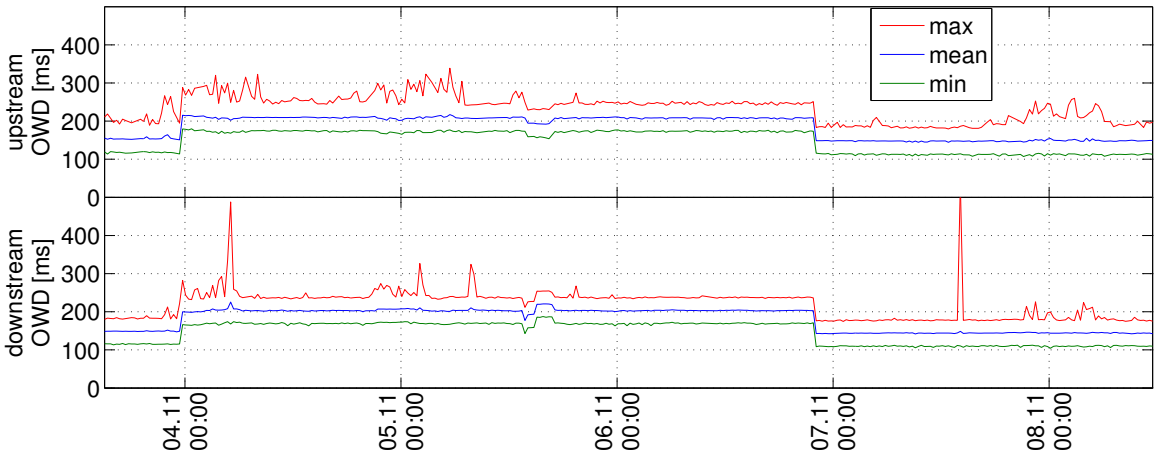
Figure 7.3: Path Germany-France: Comparison of active RTT and flow-based OWD



(a) Active RTT measurements



(b) Uncorrected flow-based OWD of RTT measurement traffic



(c) Corrected flow-based OWD of RTT measurement traffic

Figure 7.4: Path Germany-Australia: Comparison of active RTT and flow-based OWD

there is a high delay variation and a high number of packets is affected by queuing delay. As expected, the RTT values on the path Germany-Australia are much higher (Figure 7.4(a)). The smallest value is 290 ms and there are sporadic peaks of 674 ms. Over three days, the median RTT is considerably higher (410 ms) than in the rest of the considered period (295 ms). This indicates that there have been routing changes in the underlying MPLS network that impacted propagation delay, which lead to larger RTT values.

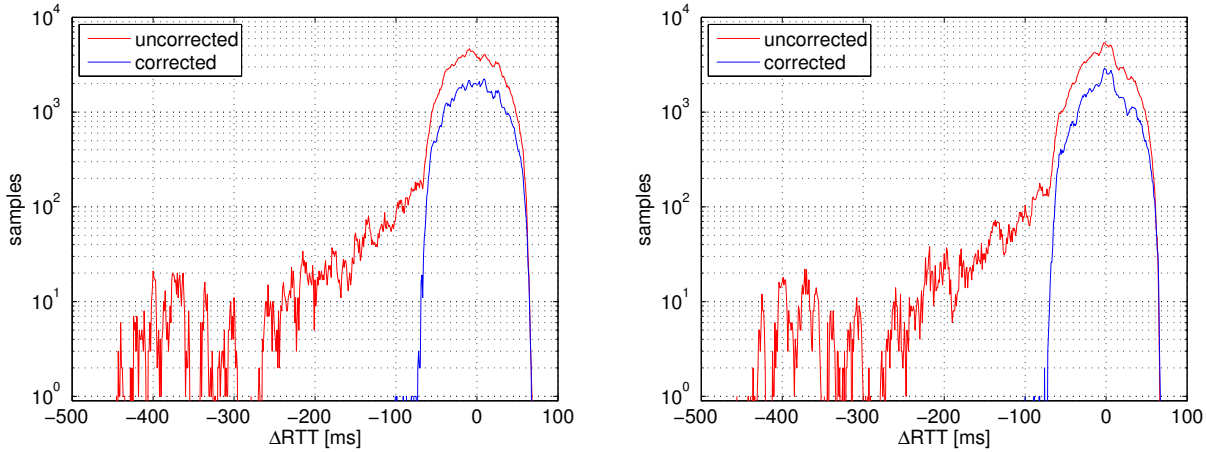
The NetFlow records that report flows of the measurement path have been taken from the flow data collected by the measurement machine. In order to obtain comparable results, only flow records of the RTT measurement traffic have been fed into the online processing framework for OWD sample creation (see Chapter 5). Figure 7.3(b) and Figure 7.4(b) show the resulting uncorrected OWD values, i.e., no blunder filter, systematic error compensation, or other correction methods have been applied. Like with the RTT evaluation, the OWD values are evaluated per measurement burst and for each burst, the minimum, mean, and maximum OWD is shown. Upstream and downstream OWD are displayed separately. The systematic error of 30 ms that corresponds to the calculated bias shown in Table 7.5, can clearly be seen in Figure 7.3(b) as it leads to negative mean values for the downstream direction. Figure 7.3(b) and Figure 7.4(b) show maximum value peaks that correspond to the peaks observed in the active RTT measurements of Figure 7.3(a) and Figure 7.4(a). Additionally, there are negative minimum values that are implausible and are caused by blunder from exporter-internal delays (see Section 4.4.4).

Figure 7.3(c) and Figure 7.4(c) shows the flow-based OWD results with all correction and filtering methods applied in the online processing chain. In Figure 7.3(c) upstream and downstream direction have the same mean value due to the compensation of the systematic error. Additionally, the implausible minimum values caused by exporter-internal delays have disappeared, while the peaks of maximum values that correspond to the peaks are also present in the RTT measurement results are still present. In Figure 7.3(c) the benefit of OWD measurements compared to RTT measurements becomes clearly visible: delay is caused mainly in downstream direction while there is only one large peak in upstream direction. Such observations are valuable input to troubleshooting and performance reporting. Figure 7.4(c) clearly shows that the OWD changes due to MPLS routing effects do not impact OWD in upstream and downstream direction in the same way at certain points in time. This indicates that there may have been partially asymmetric routing in the MPLS network.

In order to evaluate the per-sample impact of sporadic and random errors, active RTT measurement samples and flow-based OWD samples have been directly compared. The samples have been mapped to each other based on time stamp and TCP port numbers. The difference between active probe-traffic based RTT samples and NetFlow-based OWD samples is defined as

$$\Delta RTT = RTT_{flow} - RTT_{probe} = OWD_{flow,upstr.} + OWD_{flow,downstr.} - RTT_{probe} \tag{7.1}$$

Figure 7.5 shows the result of this sample-by-sample comparison as a histogram with logarithmic Y-axis for the uncorrected and corrected case. The charts show that all samples with a large difference between active and NetFlow-based measurement have been eliminated by the profile-based corrections. The remaining error of the corrected samples is caused by the random error caused by limited timestamp resolution. Due to the blunder filtering, some correct OWD



(a) Germany-France (b) Germany-Australia

Figure 7.5: Difference between RTT from active measurement and flow-based OWD

RTT path	ΔRTT_{min}	$\overline{\Delta RTT}$	ΔRTT_{max}	s	$\tau_{\Delta RTT,99.99}$
France uncorrected	−444.4 ms	−6.95 ms	68.0 ms	37.88	491
France corrected	−100.6 ms	−1.10 ms	67.9 ms	26.70	138
Australia uncorrected	−455.1 ms	−7.29 ms	67.0 ms	37.47	491
Australia corrected	−99.0 ms	−0.86 ms	67.5 ms	25.78	137

Table 7.9: RTT difference comparison

samples are also dropped (green line is always lower than blue line). This is, however, a good trade-off regarding the accuracy gained.

Table 7.9 gives a summary of these comparisons. It shows the minimum, mean, and maximum ΔRTT as well as the standard deviation s of ΔRTT. The difference of the mean value is only about 1 ms for the corrected case, which most likely corresponds to the sum of OWDs of the unobserved paths between the last routers and the end systems. The random error distribution for this error measurement corresponds to the convolution of all four uniform distributions of each OP, which results in a bell-shaped PDF. Its standard deviation is $\sigma = 26.17$ calculated from the resolution values of Table 7.4 using Equation (4.21). This calculated value is very close to the empirical standard deviation s given in Table 7.9. The table also gives the range of the error distribution symmetric to the mean value within which 99.99 % of samples are located. This is denoted as $\tau_{\Delta RTT,99.99}$ and indicates the range obtained with outliers removed (e.g., the outliers on the left of the green distribution in Figure 7.5). $\tau_{\Delta RTT,99.99}$ for the corrected cases is very close to the theoretical range resulting from resolution values, which is 136 ms (resulting from addition of values from Table 7.5).

In summary, the profile-based correction leads to highly accurate OWD samples, which only show the random error that can be known a priori from the profile parameters.

7.2.3 Confidence Interval Calculation

The OWD samples created after applying all error correction steps are still subject to random errors caused by limited timestamp resolution. In order to reduce the impact of random errors, several samples can be collected to get a better estimate of the mean value, attributed with a confidence interval. As highlighted in Section 2.2.2, the most widely used formula for calculating the confidence interval is based on the actual standard deviation of the collected samples and the Student-T distribution (Equation (2.6)). The Student-T based formula is problematic, however, with bimodal distributions (see Section 4.4.3.3). Thus, a formula based on the normal distribution and known standard deviation (Equation (2.7)) is favored. The standard deviation is calculated based on convolutions of uniform distributions, i.e., by applying Equation (4.20), Equation (4.22), Equation (4.23). In this section, confidence intervals with a 95 % confidence level are calculated for OWD values of the evaluation scenario. The approach based on the Student-T distribution (called *method T*) and the approach using the normal distribution (called *method N*) are both used and the results are compared.

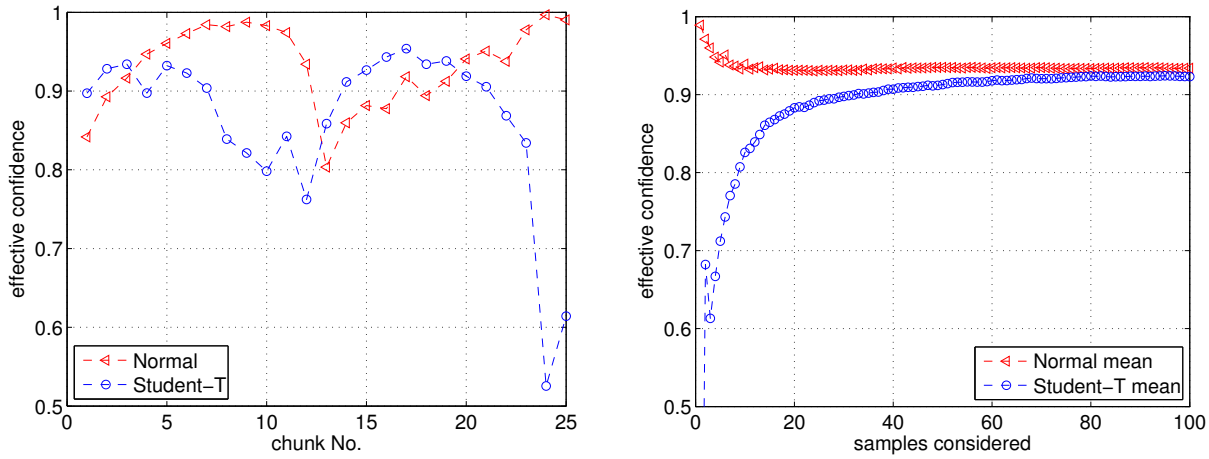
Two OWD sample data sets obtained from NetFlow data of routers shown in Figure 7.1 will be considered:

- **OWD-64-64**
  OWD samples obtained from the OPP consisting of one interface of DE-CE and one interface of DE-CO2. According to the profile, the first OP has ρ_s=ρ_e=64 ms and the second OP ρ_s=ρ_e=64 ms. This set contains 256,000 samples.
- **OWD-4-64**
  OWD samples obtained from the OPP consisting of one interface of AU-CE and one interface of DE-CO2. According to the profile the first OP has ρ_s=ρ_e=4 ms and the second OP ρ_s=ρ_e=64 ms. This set contains 30,000 samples.

Both data sets result from profile-based OWD calculation, i.e., error correction and filtering mechanisms have been applied. NetFlow records resulting from the active probing traffic have been removed, since this traffic causes correlation effects due to the roughly constant IAT and thus impacts the resulting error distributions. Sporadic high OWD values have been removed as well (less than 1 % of samples).

Each data set has been split into chunks of several samples for which mean values have been calculated. OWD-64-64 has been split into 25 chunks, OWD-4-64 into 30 chunks. Mean values of the chunks do not differ by more than 2 ms. Within each chunk, confidence intervals have been calculated for samples within a sliding window using the two different approaches. For each chunk it was evaluated how often the confidence interval contains the "true" mean value of the chunk. This frequency will be called *effective confidence*. The size of the sliding window was varied from 1 to 100 in order to study the confidence interval calculation for different numbers of samples.

First the results of the data set OWD-64-64 are considered. In this case both OPs have the same resolution. Thus, bimodal distributions can occur, which can heavily impair the method T confidence interval calculation (see Section 4.4.3.3). Figure 7.6(a) shows this problem for the



(a) Effective confidence for 15 samples per window over chunks

(b) Mean values for window sizes of up to 100 samples

Figure 7.6: Comparison of effective confidence bounds using Student-T and normal distribution based confidence interval calculation

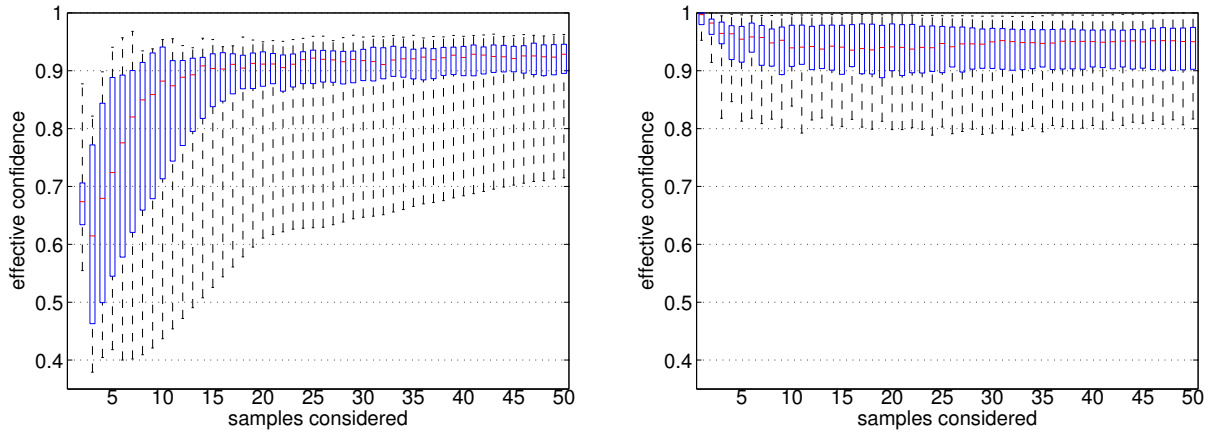
exemplary case where the window for calculating confidence intervals always contained 15 samples. The effective confidence of the method N results shows variations between 0.8 and almost 1.0. However, the method T based results show much more variations and effective confidence results of less than 0.6 for chunk No. 24. Most likely, the samples in these chunks follow a bimodal distribution. In such cases extreme errors result from cases where the T-based method returns a confidence interval size of zero if all samples are on the same peak of the bimodal distribution.

Figure 7.6(b) shows the effective confidence for all window sizes. Neither the method T nor the method N deliver the confidence level 0.95 that was intended. However, the method N provides a much better approximation. Looking at the boxplots[1] in Figure 7.7, method T results generally lead to a much larger range of effective confidence and are thus less stable than the method T results, which stay consistently in a smaller range.

The same evaluation for the OWD-4-64 data set (Figure 7.8) shows a much smaller resulting effective confidence range for both methods. Method T is less problematic here, since the distribution of OWD values comes close to a trapezoidal distribution and due to the high number of peaks (see Figure B.1 in Appendix B.1), the probability that all samples within a window are on the same peak is very low. In this case the method T even provides slightly better results than the method N.

In summary, this evaluation shows that calculating the confidence intervals based on the normal distribution (method N) in general provides more stable results than the calculation based on the Student-T distribution (method T). While the method T provided slightly better results for the OWD-4-64 data set, its results are clearly not tolerable for the OWD-64-64 data set. Hence, using the method N, which requires a priori knowledge about standard deviations from the exporter profiles (resolution values), is the best solution.

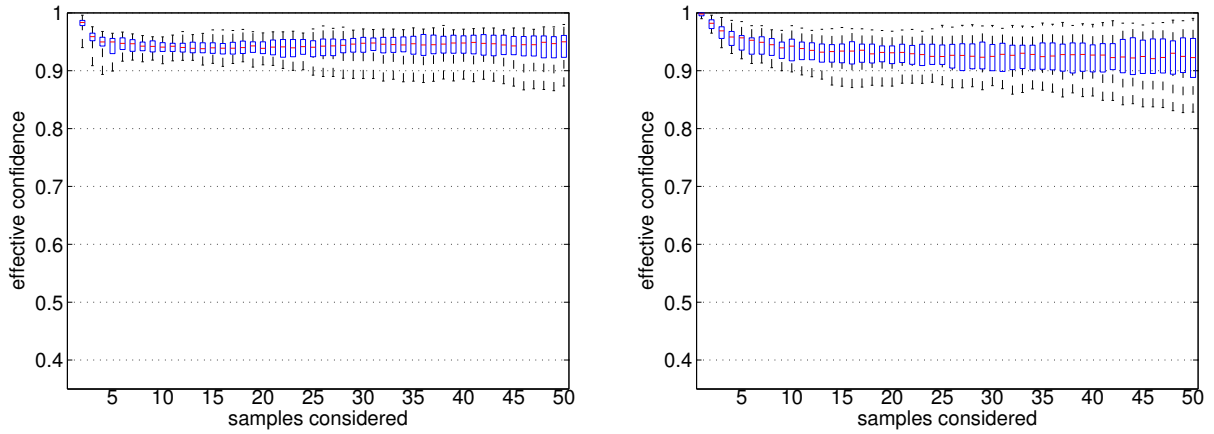
[1] Boxes indicate the interquartile range, whiskers the minimum and maximum values. The red line marks the median value.



(a) Confidence interval calculated based on the Student-T distribution

(b) Confidence interval calculated based on the normal distribution

Figure 7.7: OWD-64-64: Effective confidence obtained for different amount of samples



(a) Conf. interval using Student-T distribution

(b) Conf. interval using normal distribution

Figure 7.8: OWD-4-64: Distributions of effective confidence intervals obtained for different amount of samples

7.2.4 System Clock Skew Compensation

As indicated in Section 7.2.1, the system clock skew of the OPs considered in the evaluation scenario is small compared to other routers in the data set. Additionally, the active probe traffic consisted of one-packet flows, thus the inactive timeout triggered always the record export after roughly 10 s, and hence δ_s is often very small. Thus, the error caused by system clock skew was very small in the previous evaluations compared to cases with active timeout. There, the error caused by $\varphi_{0,y}$ becomes more severe.

In order to show the impact of system clock skew compensation, an OPP with system clock skew values of $\varphi_{0,y,OP1} = -7.83 \cdot 10^{-5}$ and $\varphi_{0,y,OP2} = 3.14 \cdot 10^{-5}$ was selected. As shown in the histogram in Figure 7.2, these are very common values. From the 24 h of flow data of this OPP, OWD samples have been created with and without system clock skew compensation. For each OWD sample, the export delay of either OP was determined, i.e., δ_s of OP1 and OP2 if the

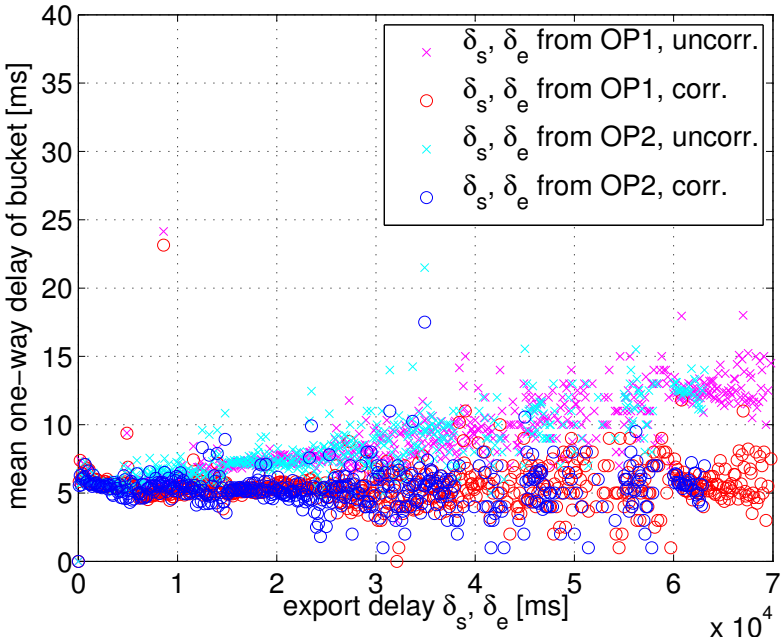


Figure 7.9: System clock skew compensation impact

sample was created from record start time, or δ_e of OP1 and OP2 if the sample was created from record end time. Each OWD sample was stored in a data set for OP1 and a data set for OP2 along with the respective export delays. All samples of a data set have been put into buckets according to export delay. 700 buckets with 100 ms width have been used in order to cover the typical export delay values from 0 s to 70 s. For each bucket the mean OWD was calculated.

Figure 7.9 shows scatter plots with the export delay value of buckets on the X-Axis and the mean OWD value of each bucket on the Y-Axis. For the samples indicated by a cross, system clock skew was not compensated. A clear linear dependency between export delay values and resulting OWD can be seen. If system clock skew effects are compensated (samples indicated by a circle), there is no more such dependency, but the majority of samples is close to a horizontal line at 5 ms. Figure 7.9 shows outliers due to two reasons: first, OWD values themselves may sporadically be high. Second, there is random error and its impact on the mean value depends on the amount of samples per bucket, which is not equal. The random error has not been evaluated further for this scenario.

From Figure 7.9 it becomes clear that system clock skew can lead to errors of up to 10 ms, which is severe compared to other systematic errors. Thus, system clock skew compensation is an important part of error compensation.

7.2.5 Summary of Profile Impact on Measurement Accuracy

This section has presented how using the exporter profile in OWD extraction improves the accuracy of flow-based OWD measurements. The profile parameters that have been created from the flow data of the evaluation scenario show on which OPs there is a byte count limitation and that exporter clocks are synchronized. Furthermore, parameters on the system clock skew

and the resolutions of the timestamps have been created. They are required for blunder filtering, correcting the system clock skew impact, and for calculating systematic and random errors.

An evaluation, which has been largely based on active measurements in an evaluation scenario, has revealed the following errors in flow-based OWD measurements and respective profile-based corrections or quantifications

- Exporter-internal delays lead to errors of several 100 ms, but such samples can be properly detected and discarded based on the system clock skew from the profile.
- Mixed timestamp resolutions at the OPP caused a systematic error of 30 ms. Bias calculation from profile parameters and correction compensates this error.
- The standard deviation and range of the random error distribution corresponds to the values calculated from the exporter profile.
- Calculating confidence intervals benefits from profile parameters due to the known standard deviation. It has been shown that the Student-T based confidence interval calculation method, which does not take a known standard deviation into account, leads to inaccurate confidence intervals due to the multi-modal distributions that occur.
- System-clock skew can lead to errors of up to 10 ms, but these errors can be corrected based on profile parameters.

In summary, the exporter profile highly improves the accuracy and enables the quantification of errors in flow-based OWD measurements. Its impact on dimensioning of the window size for online processing is considered in the next section.

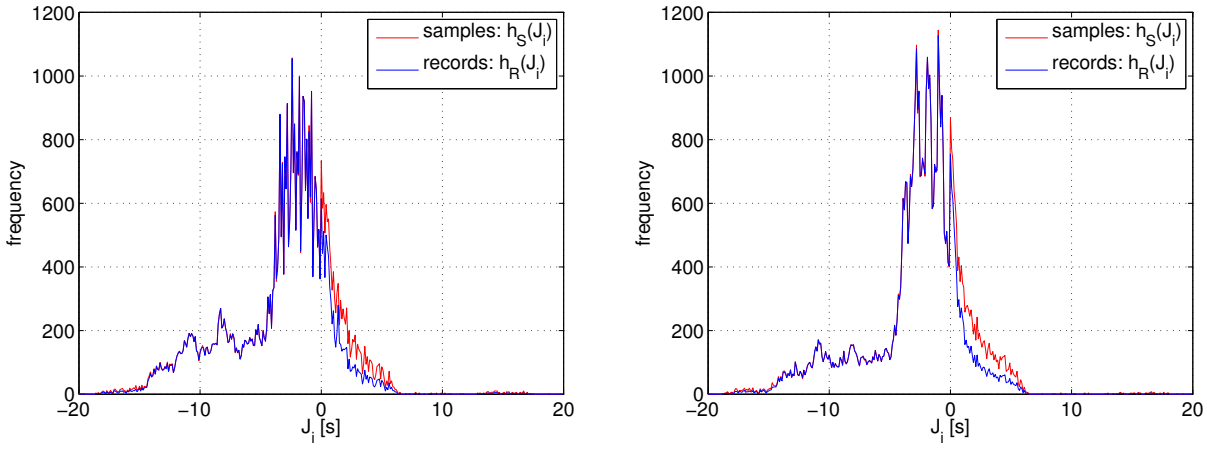
7.3 Profile-based Window Size Dimensioning

In addition to error compensation and quantification, profile parameters serve the purpose of dimensioning the window size in online processing (Section 5.3). The respective profile parameters are obtained from weekday data sets in order to cover the high record rates during peak times. For window dimensioning, it is important to quantify how the window size parameters impact the amount of OWD samples that can be calculated and the amount of memory consumed in order to find a suitable trade-off.

For evaluating the OWD sample count estimation and estimation of records buffered in memory, OWD samples of the OPP between DE-CO1, interface A and AU-CE, interface A are considered. The sample count estimation is based on profile data of week days, thus the data from a Thursday and a Friday will be considered in the following.

7.3.1 Estimating the Amount of Delay Samples

As indicated in Section 5.3, the export jitter PDF or its histogram as discretized version is input to the sample count estimation according to Equation (5.7). Figure 7.10 shows the per-OPP



(a) Thursday (b) Friday

Figure 7.10: Histograms of per-OPP export jitter per record ($h_R(J_i)$) and per sample ($h_S(J_i)$) with bin size 100 ms

export jitter histograms per record $h_R(J_i)$ and per OWD sample $h_S(J_i)$ for two different days. Both histograms look similar and in general OP2 exports records earlier than OP1 (see also Figure 5.8, which explains this effect). In the following, profile parameters are created from the Thursday characteristics and applied to Friday data.

In order to evaluate the estimation accuracy, the sample count estimated by Equation (5.7) is compared to the real amount of OWD samples that have been obtained using the online processing method (see Section 5.2.3) with different window sizes. Figure 7.11 shows the estimated sample count and the real sample count. For both days, similar results are obtained. Increasing the window size up to 5 s already delivers 80 % of the samples that can be obtained at maximum. With 15 s window size almost 100 % of the samples can be obtained. While the estimation for Friday is very close to the real sample count obtained, there is a small deviation for the Thursday data. If the Thursday profile data is taken for dimensioning, the real sample count is underestimated by 10 % for window sizes of 5 s, while there is less deviation for other window sizes. This estimation accuracy is considered to be sufficient for window dimensioning, since it only gives general guidelines for memory usage and thus does not need highly accurate results.

7.3.2 Estimating the Number of Records in Memory

For estimating the memory consumption of the online processing chain, the maximum amount of records that are present in memory for a given window size is important. This parameter is estimated based on the maximum record rate of each OP of the OPP (Equation (5.10)). Since it requires matching the records to know which flows will also traverse other OPs, all records that are created by an OP have to be considered.

The maximum record rates $\lambda_{R,\max}(L, T_j)$ in the profile have been obtained for every 15 minute interval of time of day T_j for different record window sizes L according to the method presented in Section 6.4.2. The record rates have been calculated based on the record arrival times at the

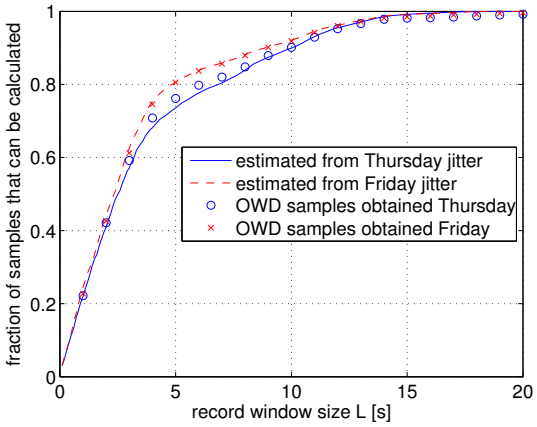
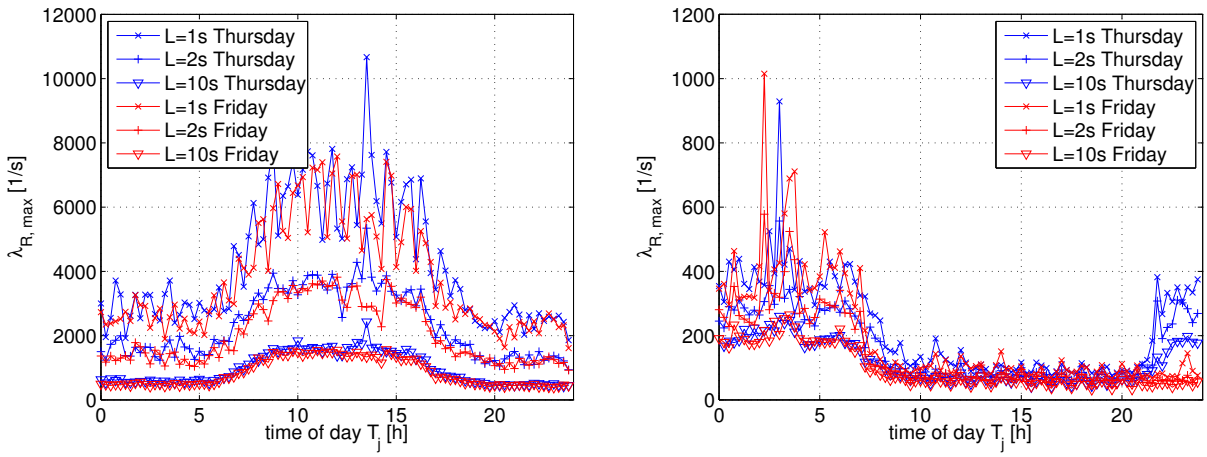


Figure 7.11: Samples calculated from per-OPP export jitter PDFs compared to samples obtained using different record window sizes

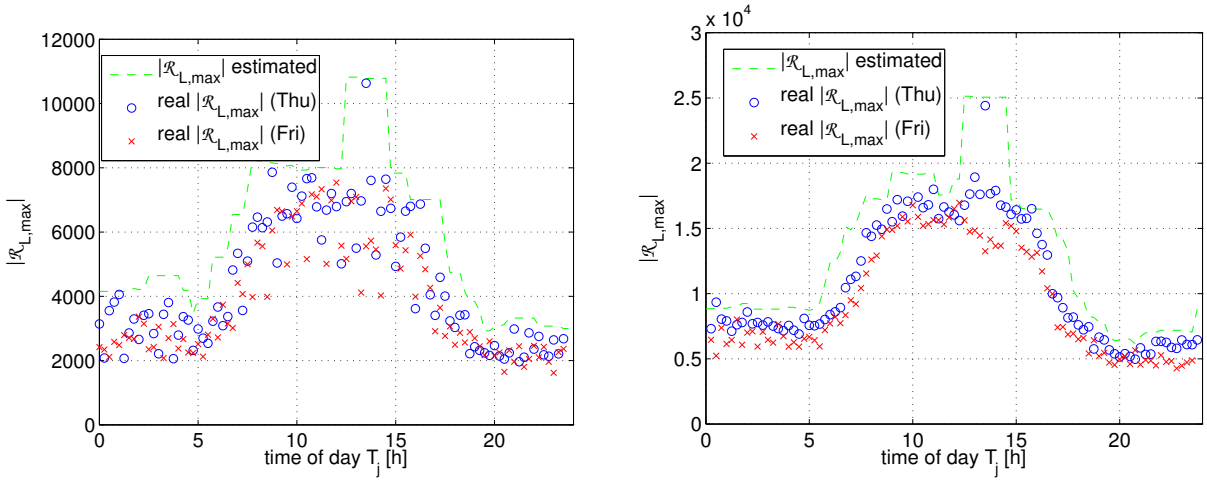


(a) OP DE (b) OP AU

Figure 7.12: Maximum record rate $\lambda_{R,\max,L}$ per 15 minute interval

collector recorded by the modified NfDump. Figure 7.12 shows the different rates obtained at the OPs. This figure clearly shows the burstiness of NetFlow traffic: the smaller the window the higher the maximum rate. Comparing Figure 7.12(a) and Figure 7.12(b) the different peak times in different continents can be seen. This effect and its impact was detailed in Figure 5.9.

Based on the maximum record rates, the maximum amount of records in memory $|\mathscr{R}_{L,\max}|$ can be calculated by Equation (5.10). It is compared in the following to the real maximum number of records that is present in memory during OWD calculation. For two windows sizes, Figure 7.13 exemplarily shows that the worst case estimation indicated by the green line holds and that during real processing hardly more records will reside in memory. An interesting observation in Figure 7.13 is that increasing the window by a factor 10 leads to approximately 2.3 times more maximum records in memory only. This shows that reducing the window size to save memory does not prove to be as effective as it might seem at the first glance due to the burstiness of NetFlow traffic. However, in summary, the profile-based estimation of records buffered in memory provides good results for dimensioning.



(a) L=1s (b) L=10s

Figure 7.13: Calculated max. records in memory compared with real values for different L

7.4 Summary

This chapter has presented the application of the exporter profile concept to data from an evaluation scenario. It has shown to which extent OWD sample accuracy and window dimensioning for OWD extraction benefit from the exporter profile.

The error-related profile parameters indicated that observation points in the evaluation scenario are subject to byte count errors and have different system clock skew values. Furthermore, resolution parameters of the profile can be reliably detected. Despite the asymmetric routing in the scenario, OPP inference worked well with the exception of one OPP that was added manually based on other OPPs. Real-time clock checks revealed that exporter clocks were synchronized.

Based on these exporter profile parameters the OWD accuracy can be highly improved. Especially the profile-based blunder removal and systematic error compensation lead to considerable improvements. As predicted in Chapter 4, the confidence bound estimation based on the Student-T distribution is not feasible because it is impaired by the bimodal distributions that can occur. It has been shown that the confidence bound calculation based on the normal distribution with support from profile parameters provides much better approximations. However, a difference to the desired confidence level remains. An analysis of system clock skew values of the whole data set revealed that errors of up to 10 ms can occur in many cases, but correction of these errors is possible. The evaluations of measurement errors showed that error correction and quantification is possible solely based on profile parameters.

Due to the need to buffer flow records in memory for some time, online processing of NetFlow data for OWD measurement can cause high and unpredictable memory consumption. The method for window dimensioning based on profile parameters has been evaluated by first creating profile parameters for the record rates and sample distributions. Afterwards, it has been checked whether the estimates hold in online processing of the data. The evaluation of the

window dimensioning method showed that the estimates obtained by the exporter profile are appropriate and that the worst case number of records in memory is hardly exceeded.

In summary, this section showed that flow-based OWD measurement is feasible, but requires profile support for essential error corrections and for dimensioning of the online processing chain. This especially holds for the systematic errors, sporadic large errors (blunder), and the errors caused by system clock skew. Furthermore, the random error can be correctly quantified based on profile parameters.

8 Conclusions

8.1 Summary

Enterprise networks provide communication services to business critical applications and must therefore deliver high performance. This demands for thorough monitoring and management of such networks in order to detect performance problems at an early stage. One-Way Delay (OWD) is an important measurand in this domain since OWD directly impacts response time and is an indicator for congestion that causes higher queuing delay. This thesis has proposed a novel method for flow-based OWD measurement. It takes flow data as input, which is often already collected from routers for other purposes or flow capturing can be easily enabled. Compared to active OWD measurement approaches this means that no additional measurement traffic or setup of active measurements is necessary and OWD samples are directly obtained for production traffic.

Due to the properties of flow data and limitations of flow capturing devices, flow-based OWD measurement is by far no straightforward data processing approach, but has to deal with several **challenges**:

- **Effects from flow capturing**

  Due to the timeout-based flow capturing mechanisms and network effects, flow records from two observation points can differ in the amount of aggregated packets and also in terms of fields that are not affected by this aggregation. If such effects are not considered, record matching cannot be performed properly and wrong or no OWD samples can be obtained at all.

- **Errors in flow capturing**

  There are several errors and limitations in flow capturing implementations of routers that have not been documented before. These errors in timestamp values or byte count values heavily impact OWD results and record matching.

- **Large amount of flow data**

  Routers in large enterprise networks produce a large amount of flow data that needs to be processed efficiently. Processing of flow data for the extraction of OWD samples requires fine-grained flow data analysis and the matching of records from different observation points. Such tasks are typically not performed in today's flow data processing applications.

These challenges have been addressed by the following **methods** for finding corresponding solutions:

- **Recording of flow data and active measurements in a large enterprise network**

  Flow data from an enterprise network was collected for several times until the final measurements that are documented in this thesis could be performed. These measurements required the setup of a special machine for flow data collection, the implementation of active reference measurement facilities, and performance tests in a laboratory environment.

- **Design and implementation of a flexible processing framework**

  In the context of this thesis, a flexible flow data processing framework was designed and implemented. This was the basis for iterative studies of different flow data processing mechanisms and statistical evaluations.

- **Laboratory setups and measurements**

  Laboratory measurements with flow capturing enabled routers allowed for more precise timing analysis than enterprise network measurements. With several measurement setups and runs the behavior of the routers available in the laboratory have been studied.

- **Iterative modeling and validation**

  Based on the three previous methods, the models developed in this thesis have been iteratively refined and evaluated. The laboratory measurements provided detailed and precise results while the enterprise network measurements delivered data from a high number of different routers for model validation.

The resulting **solutions** and **main contributions of this thesis** address the challenges mentioned above:

- **Detailed overview on network and flow capturing effects**

  All effects and their impact on record matching and hence OWD measurement have been summarized in tables that show impact and countermeasures. This is an important input for the design of flow-based OWD measurement.

- **Timestamp creation model**

  A timestamp creation model has been developed that reflects all effects in flow capturing devices regarding the creation of NetFlow v5 timestamps. With different parameter settings, the model can be adjusted to match the timestamp creation behavior of different flow capturing devices.

- **OWD extraction model**

  The OWD extraction model reflects all steps that have to be performed for proper OWD extraction. This includes mechanisms for record matching as well as the correction and quantification of timestamp errors.

- **Online processing approach**

  An implementation of the OWD extraction model as online processing approach has demonstrated its feasibility. The processing approach is able to handle high data rates and is configurable in terms of resource consumption.

- **Exporter profile concept**

  The model parameters that describe the flow capturing behavior of a device are summarized in the exporter profile. A profile creation approach that systematically obtains these parameters based on flow data has been proposed and evaluated.

- **Evaluation of measurement accuracy**

  An evaluation based on active reference measurements and flow data collection showed which accuracy can be achieved using flow-based OWD measurements. This evaluation has also revealed the exporter profile parameters for an evaluation scenario of an enterprise network and therefore provides general information regarding flow capturing behavior and accuracy.

The aforementioned key points have been systematically addressed throughout this thesis, as the following summary on the chapters will show.

Chapter 2 introduced fundamentals of IP-based networks and delay measurement. The section on router functionality and architecture showed that depending on the router architecture a different number of forwarding engines and queues are present. This has impact on the positions where flow capturing can be performed and which queues can contribute queuing delay before or after a measuring point. Furthermore, network structures have been discussed and the differences between the global Internet and global enterprise networks have been highlighted. Enterprise networks typically use MPLS VPNs with several edge routers per branch location for resilience reasons. Edge routers are typical devices where flow capturing is be performed. A section on metrology introduced metrological terms and detailed the concept of measurement errors and error propagation. Before existing approaches for delay measurement have been discussed, the measurands and metrics have been clarified. It has been shown that there are different definitions for network delay, and a definition for this thesis has been specified. The overview of existing delay measurement approaches distinguished active and passive measurement approaches. It was highlighted that predominantly active measurement technologies are used in today's networks, but passive approaches have certain advantages and are of high interest in research activities. Finally, the problem of clock errors has been addressed by first clarifying related terms and then introducing clock synchronization mechanisms and algorithms for removing clock skew or drift from measurement data sets.

Chapter 3 dealt with flow capturing, an important mechanism for traffic monitoring in enterprise and provider networks. This chapter started with giving consistent definitions of the general term flow, more specific flow definitions, and flow capturing related terms. Then it introduced and compared several flow capturing mechanisms. Here, the main focus was on NetFlow, which is the most widely used de-facto standard for flow capturing today. For delay measurement, the very point at which flow data is captured in routers is important. Thus, flow capturing implementations in routers and probes have been discussed in detail. Related work in flow capturing

was addressed in terms of flow capturing application and deployment, as well as in terms of flow data processing approaches. It was shown that flow capturing is used predominantly for traffic reporting and accounting while little research towards obtaining QoS-related information has been performed. Flow data processing is predominantly performed in an offline processing approach, i.e., data is stored to disk or to a database before being analyzed.

Based on the fundamentals provided by Chapter 2 and Chapter 3, Chapter 4 has motivated the flow-based OWD measurement approach in Section 4.1. This approach has been compared to other active and passive delay measurement approaches and it has been found that this approach is advantageous due to the three criteria one-way, passive and low effort measurement. The following three sections have systematically addressed problems in flow-based OWD measurement that arise from different effects and errors. Section 4.2 started with considering network effects, Section 4.3 has dealt with flow capturing effects and errors, and Section 4.4 has detailed timestamp effects and errors in flow capturing devices by introducing a timestamp creation model. Systematically, each effect has been evaluated in terms of its impact on OWD extraction, and for each measurement error methods for correction or quantification have been given. Throughout these sections an OWD extraction model has been developed that takes flow records as input and creates OWD samples as output while handling all effects and errors appropriately. It has been shown that error correction and quantification requires parameters on flow capturing devices that can be provided by means of an exporter profile.

Chapter 5 detailed an online processing approach for the efficient implementation of the OWD extraction model. First, the export effects in flow data have been analyzed and, by taking processing requirements into account, a multi-stage window-based extraction mechanism has been proposed. This mechanism uses a result and a record window that define how long flow records and OWD results are buffered in memory. Second, the window-based processing approach has been detailed by evaluating the typical flow record export characteristics and deriving from the latter methods for window handling and dimensioning. An overview of a design for the window-based processing approach has been given. This design was implemented and has proven its technical feasibility. The third section has presented a profile-based method for window dimensioning that takes record rates and export jitter distributions into account for giving an estimation on the number of OWD samples and memory requirements with different window sizes. This method especially considers the effects in global enterprise networks regarding different record rates at different times of a day for different locations. The parameters required for dimensioning depend on traffic characteristics and flow capturing device configuration and could be given in an exporter profile.

The exporter profile approach that has been motivated by the need for measurement-error and export-effect related parameters in Chapter 4 and Chapter 5 has been detailed in Chapter 6. This chapter has systematically elaborated the various parameter dependencies and specified the exporter profile consisting of three parts that contain parameters on the flow capturing device, a certain observation point, or an OPP, i.e., a path between observation points. Section 6.3 has discussed various profile creation approaches based on parameter dependencies. It has been concluded that traffic-dependent parameters can only be obtained by a profile creation approach based on flow data and that this approach can also be used for obtaining the other parameters. Consequently, a flow data based profile creation approach has been proposed in Section 6.4. This approach is a seven step iterative offline processing approach that extracts

profile parameters from a reference flow data set. Due to the special properties of flow data three profiling steps demanded for the design of special algorithms. These algorithms for record rate estimation, obtaining timestamp resolution, and OPP inference have been presented in detail.

Chapter 7 has presented how flow-based OWD measurement with profile support was applied using flow data from an enterprise network. Furthermore, this chapter has evaluated measurement accuracy. The evaluation was based on NetFlow v5 data and active RTT reference measurements that were collected over five days from three different locations. OWD measurements have been evaluated on a path between two European locations as well as between a European and an Australian location of the global enterprise network. For each exporter of the scenario profile parameters were obtained. A comparison with active measurement has shown that measurement accuracy can be considerable improved using the profile-based correction methods. Furthermore, the flow-based OWD measurement results closely match the active measurement results and random errors can be correctly quantified based on profile parameters. The effect of exporter-internal clock skew has been evaluated separately and it has been shown that this effect can lead to a correctable error of up to 10 ms. In terms of profile-based window dimensioning for the online processing approach, it has been shown that the amount of samples as well as memory requirements can be well estimated.

8.2 Outlook

The flow-based OWD measurement approach and its mechanisms have several known limitations, which have been highlighted in the previous chapters. While the applicability in a large enterprise network with different types of flow capturing routers has been demonstrated, there may nevertheless exist network scenarios and flow capturing devices that require extensions to the current approach. Such extensions can be subject to future work and several ideas are presented in the following.

One point for extensions is the timestamp creation model and its adaption towards other flow capturing approaches. This thesis introduced different flow capturing effects and detailed the various timestamp representations. While the timestamp creation model can be adapted to other timestamp representations, it directly corresponds to NetFlow v5 timestamps and the evaluation in the enterprise network has also been based on NetFlow v5. This was the predominant flow capturing protocol used at the time the measurements have been conducted. Due to the increasing use of flexible flow capturing mechanisms and the migration to IPv6 (which NetFlow v5 cannot handle), NetFlow v9 and IPFIX will supersede NetFlow v5. This requires adaption of the timestamp creation model and exporter profile to other timestamp representations and additional timestamp correction steps, e.g., based on recent publications on timestamp errors in NetFlow v9.

In terms of processing of flow data for OWD extraction there is also room for extensions. One point is the possibility to take knowledge on routing into account. This can help to decide, based on the known OPP pairs, whether matching flow records from other OPs can still arrive or for a certain flow or not. Hence, flow records for which no matching records can arrive can be removed from memory very early, which could lead to considerable lower memory consumption. The amount of savings may depend on the network scenario and deployment of

flow capturing. Furthermore, knowledge on routing could be taken as input for a distributed flow-based OWD measurement scheme, where each computer of a compute cluster extracts OWD samples of a certain OPP set only. With knowledge on routing, it can be decided which flow records belong to which OPP and flow records can be forwarded to the respective compute nodes accordingly.

This thesis has evaluated flow-based OWD measurements based on a comparison to active RTT measurements in a global enterprise network. This evaluation showed that flow-based OWD measurement can highly benefit from profile based error correction and error quantification, which leads to correct results. However, the evaluation could be extended towards a comparison with active OWD measurements, which were not possible in the enterprise network considered due to the high effort required for such measurements. Furthermore, a comparison with active measurements that create multi-packet-flows could provide further details compared to the active RTT measurement using one-packet-flows that have been performed in this thesis. Another point in terms of evaluation is studying the long-term behavior of flow-based OWD measurements. The evaluations in this thesis have taken data from several week days into account. However, in order to ensure that the flow-based OWD measurement can provide the reliability and robustness required in productive network monitoring environments, further studies and long-term tests are necessary. This includes evaluations based on flow data from a longer time period on profile stability and processing effort for online OWD extraction as well as profile creation. Ideally, this also covers different network topologies with different flow capturing deployments.

In summary, this thesis has shown the feasibility of flow-based OWD measurement with profile support. The steps indicated as outlook can help to explore further details, extend the presented mechanisms, and clarify requirements for productive usage. Eventually, flow-based OWD measurement with profile support can become feasible as addition or replacement for active OWD measurements that are performed today. Furthermore, the approaches for improving timestamp accuracy can be taken as input for any other flow analysis application that relies on flow capturing timestamps.

A Details on Network Effects

A.1 Packet Size Distribution

As presented in Section 4.3.2.1, some exporters report byte counts in terms of IP packet size while other exporters report byte count in terms of Ethernet payload. This leads to errors for packets with size smaller than 46 Bytes. In order to validate whether this error is relevant, this section presents an evaluation of packet size distribution for small packets. Fragmentation is also partly considered.

Input data for this evaluation is flow data of the considered enterprise network from one day. In general, it is not possible to get the precise packet size distribution from flow data due to the aggregation of several packets into flow records. However, there are two possibilities to get a conservative estimate that in any case will be equal or less than the real number of packets with certain size. The first possibility is to filter for records with one packet only and evaluate the byte count reported in each record (*one-packet-filtering*). This is precise, but considers only a fraction of records. The second possibility is to calculate $b' = \lfloor \frac{b}{p} \rfloor$ for each record. b' is an estimate of the smallest packet's byte count. This method (*multi-packet-estimate*) also covers cases where several small packets are present in a multi-packet flow. However, the smallest packet estimate will be too large if there is much difference between packet sizes in a flow.

Figure A.1 shows the packet size distribution results obtained using the two different methods introduced above. The y-axis of the histograms is logarithmic and the bytes per packet on the x-axis, truncated at 60 bytes. All values have been normalized to the overall record count and are therefore comparable. Since all samples of the one-packet filter result in the same value for the multi-packet-estimate, the frequency of the latter is always higher than that of the one-packet-filter method.

Both charts show the highest peak 46 Bytes, which results from records that use Ethernet payload size as byte count and therefore report 46 Bytes also for all IP packets that are actually smaller. Interestingly, there are a couple of packets smaller than 40 Bytes. The smallest packets (21 Bytes) even carry only 1 Byte payload in the IP Packet (20 Byte Header). Most of the smallest packets for cases where the frequency is smaller than 1e-06 are UDP packets with source and destination point zero, i.e., they are fragments where no UDP header is present. The noticeable peak of 24 Bytes bytes is only present in multi-packet-estimate histogram. Most packets of this size are carrying *Generic Routing Encapsulation* (GRE) tuneling protocol. 24 Byte packets of this protocol are most likely keepalive messages, which are sent periodically in intervals smaller than inactive timeout and thus several of these packets are contained in a flow record.

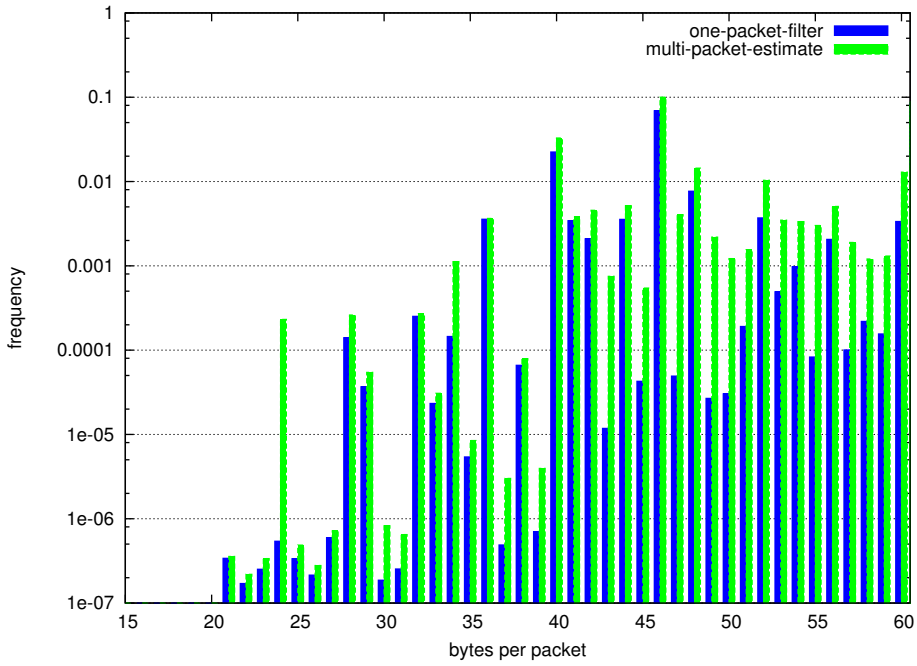


Figure A.1: Packet size distribution histograms. Truncated at 60 bytes, logarithmic Y-Axis.

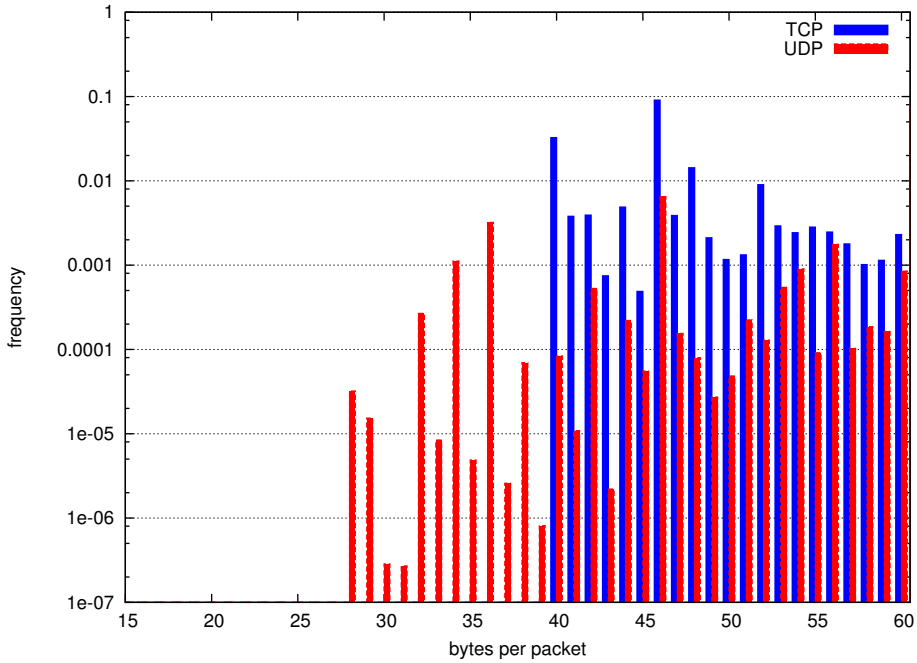


Figure A.2: Packet size distribution histograms from multi-packet-estimate for UDP and TCP flows with port number unequal to zero (no fragments). Truncated at 60 bytes, logarithmic Y-Axis.

Fragments from different observation points cannot be matched based on the five tuple, since the transport protocol header information is not available. The chart in Figure A.2 therefore shows the packet size distribution for TCP and UDP flow records with valid port numbers, i.e., no fragments. The first peak for UDP is at 28 bytes, which is the smallest UDP packet size,

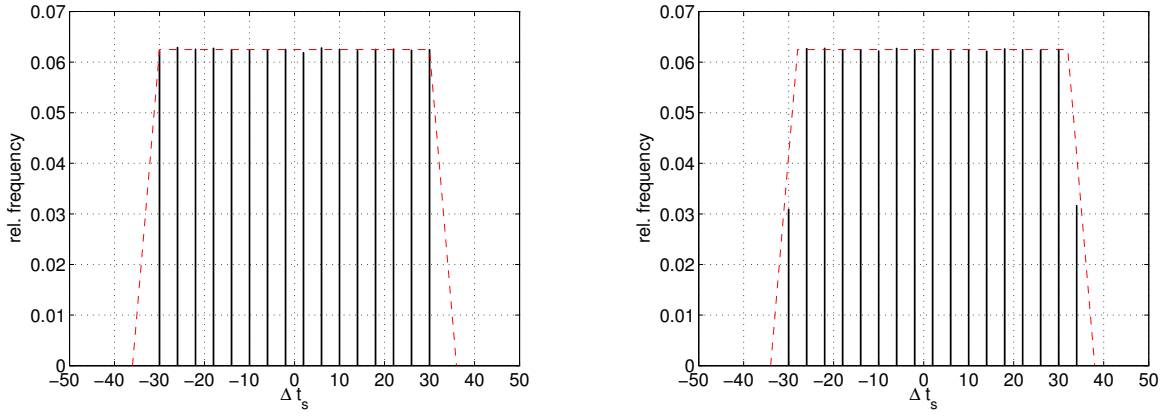
and there are several peaks below 46 bytes, which means that NetFlow's 46-byte problem in packet count is relevant for matching here. TCP packets start at 40 Bytes in both charts, which results from the IP packet size of the smallest TCP packets (Acknowledgments). Other peaks at 44 Bytes and 48 Bytes seem to be in most cases TCP packets without payload, but carrying TCP options.

B Details on Timestamp Effects

B.1 Multi-modal Distributions from different Timestamp Resolutions

The effect that the random error PDF shows multiple modes and is not a continuous distribution was introduced in Section 4.4.3.3 and will also be shown in Section B.2. Depending on the shift between the two timestamp metering points, a different shape of the PDF and a different number of peaks results.

As soon as there are different timestamp resolutions at the two OPs of the OPP, a trapezoidal PDF results for the continuous case. For the discretized case of OWD measurement, also a multimodal resolution will result, however, with a higher number of peaks. An evaluation of the distribution of a case that occurs often in flow data ($\rho_{\text{first}1} = 4$ ms and $\rho_{\text{first}1} = 64$ ms) is presented in Figure B.1. These results have been obtained by Monte Carlo experiments using the IKR simulation library [130] for modeling the resolution effects. This figure shows two distributions with different constant delay values. As expected we see 16 peaks in the perfectly aligned case and 17 peaks in the other case. Both distributions are symmetric and describe a discretized trapezoidal error distribution (systematic error compensated), as indicated by the broken lines.



(a) 0 ms constant delay

(b) 2 ms constant delay

Figure B.1: Δt_s distribution from Monte Carlo experiments. Random error characteristic for an OPP with $\rho_{s,\text{OP1}} = 4$ ms, $\rho_{s,\text{OP1}} = 64$ ms

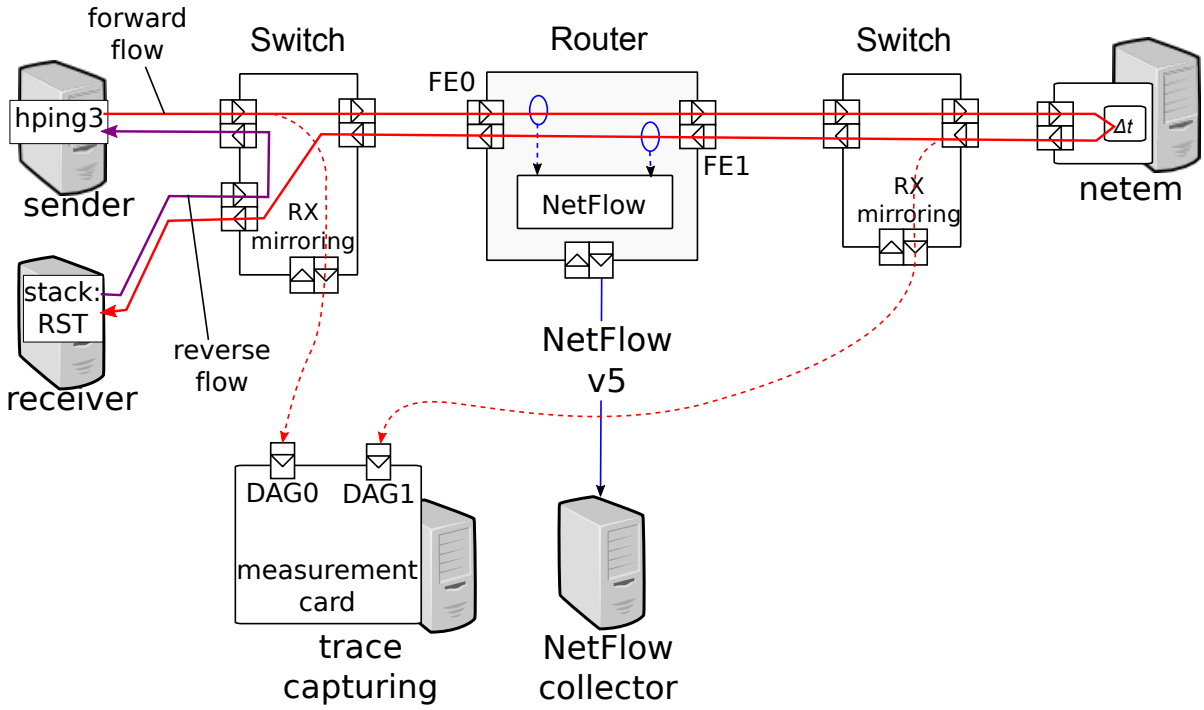


Figure B.2: Experiment setup

B.2 Model Validation by Laboratory Measurements

This section presents delay measurements that have been a network laboratory. Flow record timestamps have been obtained from a router via NetFlow v5 and a dedicated measurement card. Evaluations of the measurement show Export-Systime-Alignment (ESA) effects as well as impact of systematic and random error in scenarios where one or several OPs have a sub-millisecond resolution. Additionally, flow record timestamps created without truncating r_{nsec} are considered.

B.2.1 Measurement Setup

Figure B.2 shows the experimental setup used to perform the measurements. We used the hping3 tool [127] to create one-packet TCP flows. The sender sends TCP SYN packets to ports of the receiver for which no listening socket (e.g., server daemon) is registered. Thus, the receiver sends a TCP RST packet back and the sender can calculate the RTT based on this answer. Measurements considered in the following have only been performed on the forward flow. RTT values measured by the hping3 tool serve validation purposes only and are not considered in the following.

The forward flow (red) of this conversation is sent twice across a NetFlow-enabled router and a network emulation machine. The reverse flow takes a different path. The router used was a cisco systems 7204 VXR with NPE200 processor board running IOS Version 12.3(1a) and a Fast Ethernet line card with two interfaces (FE0 and FE1). On both interfaces of this router

(ingress) NetFlow v5 was enabled and NetFlow v5 packets were sent via another interface to a collector running the flow-capture daemon of the FlowTools tool set [99]. Additionally, policy routing with ACLs was used to setup the routing in a way that the forward flow passed the router twice. The network emulation machine used Linux netem [131] for emulating predefined delays.

Two Gigabit Ethernet switches (Netgear GS108T) with mirroring functionality were used to copy traffic received at certain switch ports to the measurement card *before* the packets reach the router interfaces. Due to the low volume of traffic and the usage of Gigabit switches in combination with Fast Ethernet router ports, effects of switch port contention due to mirroring are negligible. The measurement card used was an endace DAG 7.5G2 device that enabled us to capture traces in pcap format with 7.5 ns clock resolution from two ports (DAG0 and DAG1).

The router as well as all computers were synchronized via NTP to the laboratory server, which itself was synchronized via radio clock (DCF77) to UTC. The measurement card itself cannot synchronize via NTP, but was synchronized right before measurement start to the clock of the trace capturing machine. Thus, the measurement card's clock was free running during the experiments. Measurements revealed a skew of the card's clock of $4.2 \cdot 10^{-6}$ (about 15 ms per hour). This means that for measurements with durations in the range of 10 s, this skew causes an error of about 42 μs.

NetFlow v5 data exported from the router showed a limited aligned resolution of 4 ms in r_{first}, r_{last}, and r_{su}, i.e., $\rho_{\text{first}} = 4$ ms, $\rho_{\text{last}} = 4$ ms, $\rho_{\text{su}} = 4$ ms. The RTC ($u(t)$) had the typical resolution characteristics of $\rho_{\text{sec}} = 1$ s and $\rho_{\text{nsec}} = 15,258$ ns. The system clock skew of this router was determined to be $\varphi_{0,y} = 6.7 \cdot 10^{-7}$.

The measurement setup allows quantifying NetFlow's accuracy for OWD measurement by comparing it to values obtained from the measurement card. Since only one router is involved, there are no clock synchronization issues associated with NetFlow data from the two OPs at FE0 and FE1. The same holds for the trace-based reference OWD measurements where timestamps for DAG0 and DAG1 are taken from the same clock. Furthermore, NetFlow-based OWD measurement and trace-based OWD measurements are based on the same traffic and only the difference between the values obtained from the two methods is relevant. Thus, the accuracy of the emulated Δt delay does not impact the comparison. Measurement values presented in the following are based on a measurement run with $\Delta t = 1$ ms and 10,000 TCP-packets sent by hping3 with 1.4 ms inter packet gap.

B.2.2 Evaluation of Systematic Errors

Figure B.3 shows the mean values of Δt_{s} calculated from all samples between different OPs. As shown, the OWD values obtained from DAG and NetFlow differ by 80μs only. We assume that a large fraction of this error reflects the router-internal packet processing delay that occurs before NetFlow timestamps are taken. For the OPP considered, there is no bias on Δt_{s}, since the timestamp resolutions on both OPs are the same (same router with a central processing card).

The OWD values measured between DAG0 and FE0 as well as DAG1 and FE1 are measured on paths without artificial delay. However, the bias due to timestamp truncation is $\beta_{\Delta t_{\text{s}}} = -\frac{\rho_{\text{s}}}{2} =$

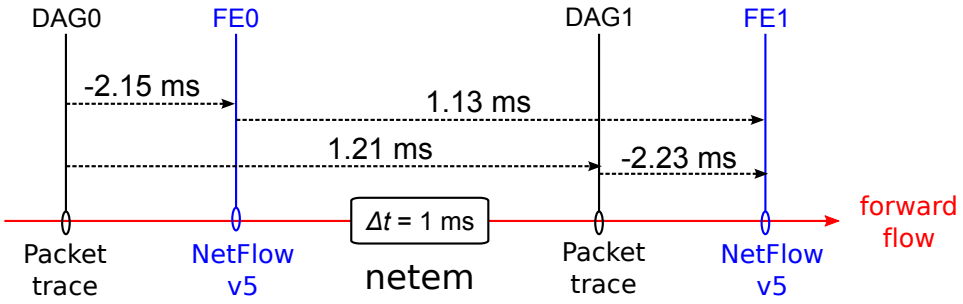
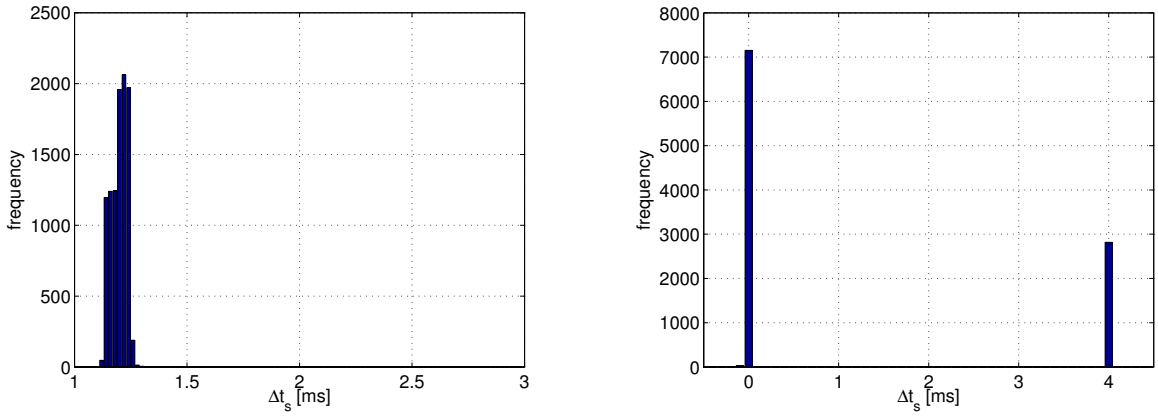


Figure B.3: Mean values of Δt_s measured for different OPPs



(a) Path DAG0-DAG1 (reference measurement), 20 μs bucket width

(b) Path FE0-FE1

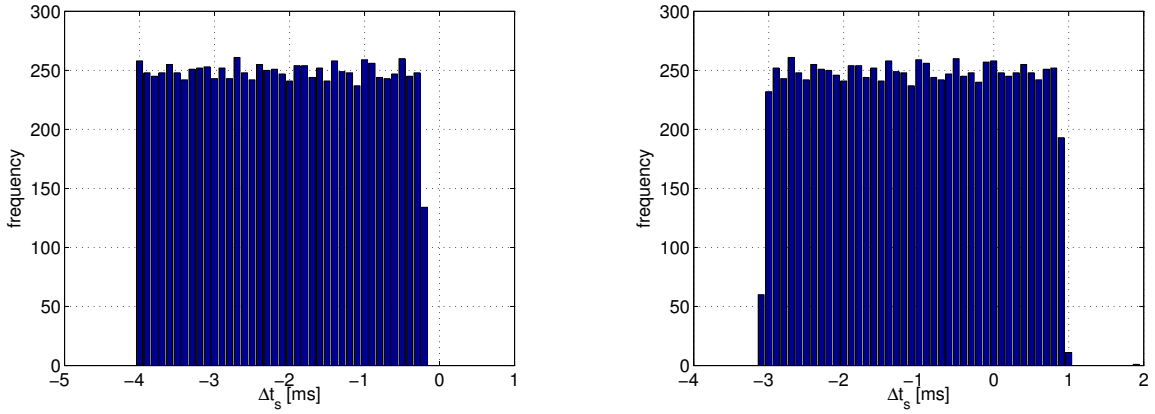
Figure B.4: Distributions of Δt_svalues, same OP types

−2ms, which is reflected in the result. The remaining error is most likely resulting from NTP clock synchronization and partly of the measurement card's clock skew. From the impact of β_s we conclude ESA holds, i.e., that flow export times are aligned with system clock resolution. If this had not been the case ρ_{su}, would cause a bias that would compensate β_s (ρ_{first}=ρ_{su}in this scenario).

B.2.3 Evaluation of Random Errors

In terms of random errors, we will consider the distributions of Δt_s values between different OPs. All timestamp processing and calculation is performed with 0.1 μs resolution (UNIX time as double precision floating point). Also r_{nsec} is taken as raw value stored from the collector and only truncated to 0.1 μs resolution and not to millisecond resolution, as it is usually done. Unless otherwise noted, histograms have been calculated using buckets of 100 μs width, with each bucket indicated in charts by the (inclusive) lower bound.

Figure B.4 and Figure B.5 show the distributions of Δt_s values and hence the error resulting from limited resolutions in flow record timestamps. Figure B.4 shows two charts with values obtained from the same type of OPs (pure DAG(a), pure NetFlow(b)), while the charts in Figure B.5 show



(a) Path DAG0-FE0, i.e., without emulated delay

(b) Path DAG0-FE1

Figure B.5: Distributions of Δt_s values, different OP types

results obtained by using different type of OPs (DAG and NetFlow timestamps). Table B.1 presents characteristic values of the distributions shown in the figures.

The error distribution of the reference measurement using the DAG card (Figure B.4(a)) indicates the delay distribution, which results from emulation inaccuracy as well as from other effects on the path and DAG card clock skew. The range of the distribution is 120 μs neglecting outliers. A completely different distribution results from the NetFlow data (Figure B.4(b)). Here, the two peaks show the impact of the limited resolution in NetFlow timestamp values (ρ_{first}) in conjunction with completely synchronized clocks between OPs as a result from the measurement scenario using one router only. In case of uncorrelated data, a triangular distribution with the highest frequency value at the mean value of the distribution would result, which is obviously not the case. Left of each peak Figure B.4(b) shows a very small number of samples, which result from r_{nsec} not being truncated, i.e., here the record timestamps are generally not exactly aligned at 4 ms.

If Δt_s is calculated from DAG and NetFlow timestamps, a uniform distribution as in Figure B.5(a) results. Since there is no emulated delay on the path DAG0-FE0, this chart shows

Table B.1: Characteristic values of measurement results

Path	$\overline{\Delta t_s}$	$Var(\Delta t_s)$	$s(\Delta t_s)$
DAG0-DAG1 (reference)	1.2113 ms	0.0049	0.0703
FE0-FE1 (NetFlow only)	1.1297 ms	3.2429	1.8008
DAG0-FE0 (no delay)	−2.1473 ms	1.3338	1.1549
DAG0-FE1	−1.0178 ms	1.3352	1.1555
FE0-FE1 (r_{nsec} truncated)	1.1297 ms	3.2429	1.8008
DAG0-FE0 (r_{nsec} truncated)	−2.7062 ms	1.3336	1.1548
DAG0-FE1 (r_{nsec} truncated)	−1.5767 ms	1.3350	1.1554

the random error resulting from timestamp characteristics only. Thus, the distribution is almost perfectly rectangular and its empirical standard deviation of $s = 1.1549$ is very close to the standard deviation calculated from timestamp resolution, which is $\sigma_{\Delta t_s} = \sqrt{\frac{\rho_{\text{first}}{}^2}{12}} = 1.1547$.

Taking the path with the emulated delay into consideration (DAG0-FE1), the random error is not perfectly rectangular anymore due to the impact of the delay variation (Figure B.5(b)). Hence, this distribution reflects a convolution of the delay distribution Figure B.4(a) and the rectangular distribution from NetFlow timestamp errors (Figure B.5a).

In summary, the results show that there is no impact on random errors caused by a limited resolution of the system uptime (r_{su}), which is another indication that ESA holds for the OPs considered in this setup. This has also be validated by checking t_{b}, which is aligned to 4 ms timestamps besides the jitter introduced by r_{nsec}. In case ESA would not hold, there would be a random error on t_{b} of some milliseconds, thus this value would vary much more.

B.2.4 Impact of Truncating NetFlow's Nanosecond Timestamp

The impact of truncating the raw timestamp r_{nsec} to milliseconds was discussed in Section 4.4.3.2. This truncation is equivalent to a limited resolution of 1 ms of this timestamp and therefore likely to cause a systematic error of $\beta_{\text{nsec}} = -0.5$ms. However, if only NetFlow data with truncated r_{nsec} timestamps is used, this error is compensated in OWD measurements. When using high resolution timestamps as in this experiment, this error can be quantified.

The NetFlow and trace data was processed without truncating r_{nsec} timestamps for the results presented above. Using the same data, but truncating r_{nsec} for NetFlow timestamps results in the values presented on the bottom of Table B.1. The mean value for the Netflow-only path (FE0-FE1) does not change, since the truncation of r_{nsec} is performed for both OPs and thus the systematic error is compensated. Compared to (Figure B.4(b))) there are only two large peaks in the distribution (not shown) and no small peaks are left.

As soon as one NetFlow OP and one DAG OP are considered, the systematic error introduced by r_{nsec} truncation becomes visible. Comparing the mean values resulting for the paths DAG0-FE0 and DAG0-FE1, there is a difference of -0.5589 mseach, which is very close to the bias calculated from $\rho_{\text{nsec}} = 1$ ms ($\beta_{\text{nsec}} = -0.5$ms). Thus, r_{nsec} truncation introduces a systematic error as expected. Table B.1 also shows differences in the standard deviation, however, there is only very little impact of r_{nsec} truncation on that.

C Details on Export Characteristics

C.1 Export Characteristics

Figure C.1 and Figure C.2 show the CDF for the export delay from record start δ_s, export delay from record end δ_e, and the record duration d taken from two exporters. From each exporter more than 100,000 records have been taken to plot the CDF. At both exporters an active timeout of 60 s is configured. The export jitter at both exporters clearly differs: on exporter 1 some δ_s values are greater than 70 s while at exporter 2 the maximum δ_s is 62 s.

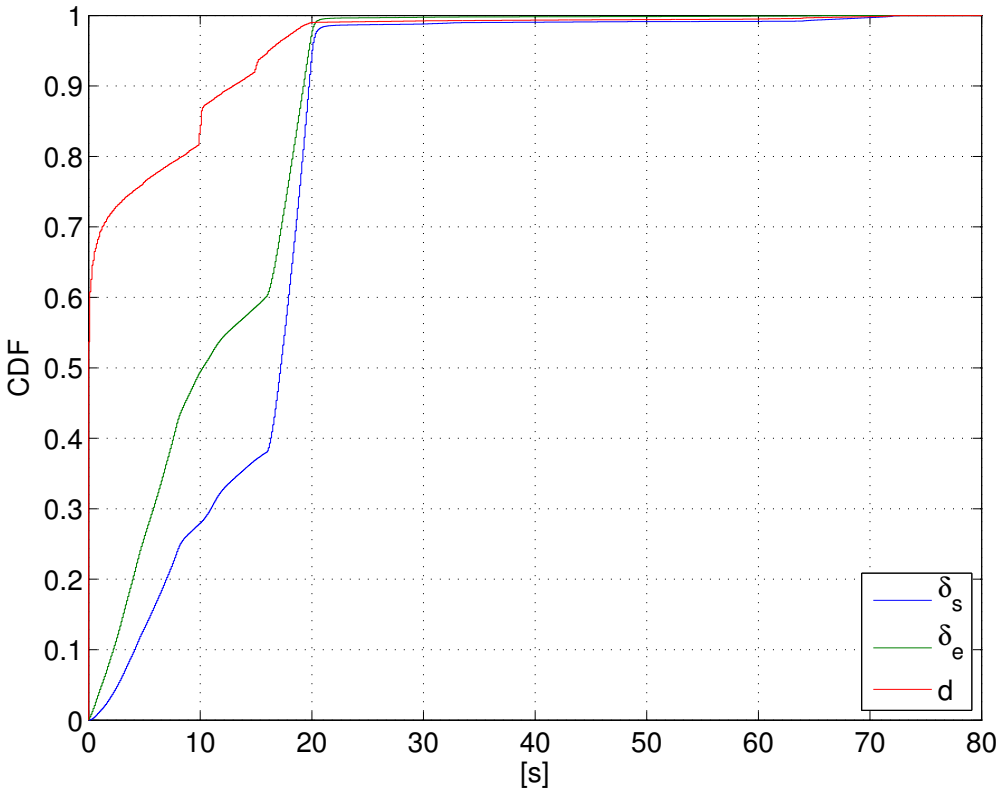


Figure C.1: CDF of export delay and record duration for exporter 1

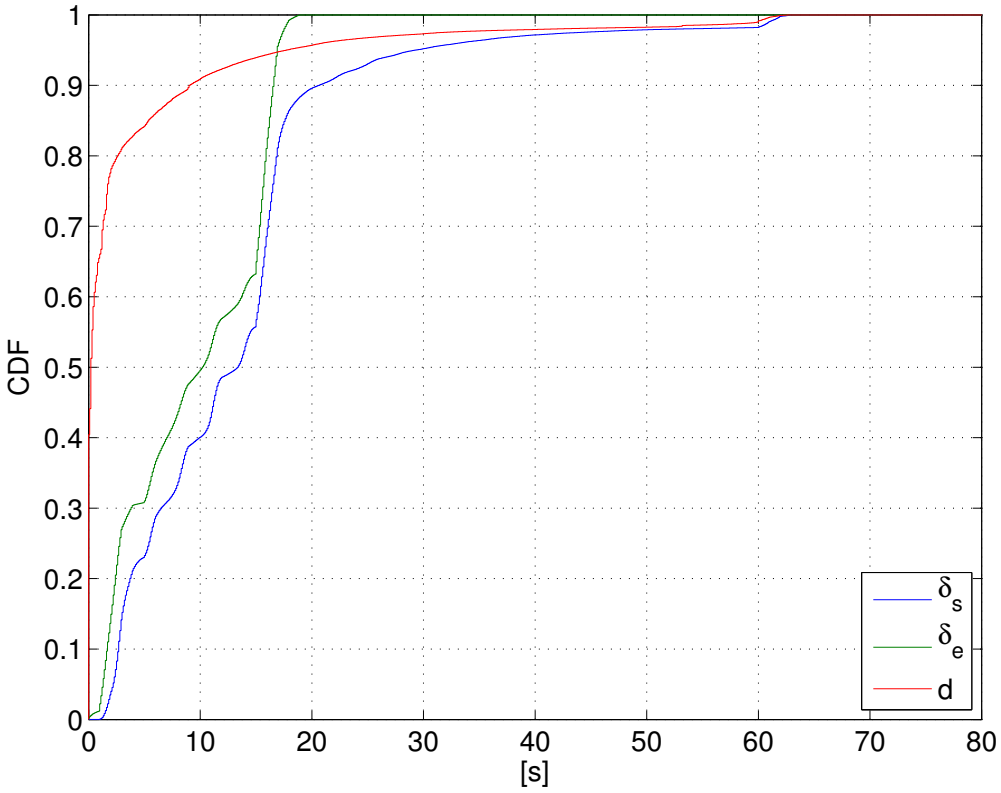
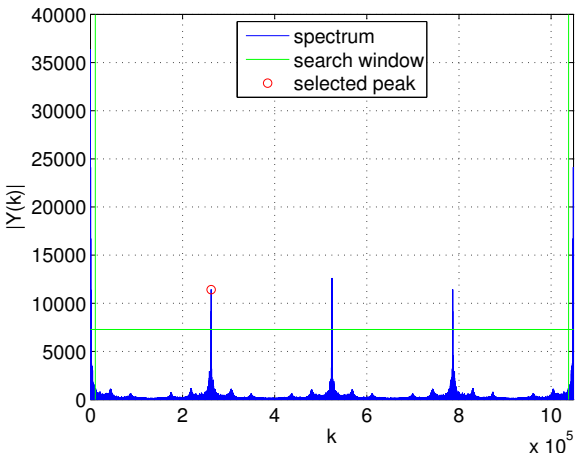


Figure C.2: CDF of export delay and record duration for exporter 2

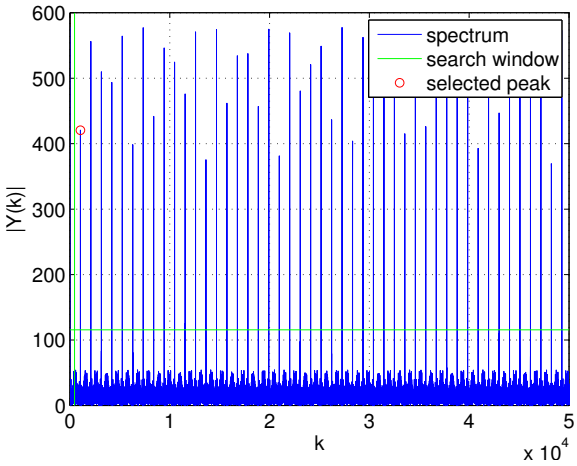
D Peak Detection Algorithm for Resolution Detection

The peak detection algorithm is highlighted using two example spectra created in the resolution profiling procedure (see Section 6.4.3). Figure D.1 shows exemplary spectra of timestamp streams. The left spectrum shows wide peaks as they occur for high resolutions and the right spectrum shows lots of small peaks as they occur for low resolutions. Note that the left spectrum shows all N values while the right spectrum only shows less than 5% of it. Especially with high resolutions the peak at k=0 has a certain width and thus peak search cannot start immediately for low k values. Contrary, peak search must start immediately with low resolution values (right spectrum).

These considerations lead to a peak search algorithm that uses a peak search window. As shown in Figure D.1 a peak has to be at least of height $0.2 \cdot |Y(k0)|$. The right and the left search window border depend on the number of peaks n_p that are larger than this threshold. The left border of the window starts at $k_l = \lfloor N/n_\mathrm{p} \rfloor$, the right border is at $k_l = N - \lfloor N/n_\mathrm{p} \rfloor$. If $n_\mathrm{p} < 1000$, the k_p value where $|Y(k_\mathrm{p})| \cdot (N - k_\mathrm{p})$ delivers the highest value within the search window is selected as peak value. Otherwise, if $n_\mathrm{p} >= 1000$ the first peak within the search window that is larger than the threshold is selected. Only if immediately higher values follow, the selection is changed to them. This algorithm reliably selected the correct peak, as exemplarily demonstrated by the



(a) Spectrum of record export timestamps t_x with ρ_x=4 ms



(b) Spectrum of system uptime timestamp r_su with ρ_su=1000 ms

Figure D.1: Exemplary timestamp spectra

two spectra in Figure D.1 and as validated by results delivered for the evaluated scenario that passed all consistency checks reliably. In case there are problems with resolution detection, spectrum data can be dumped for manual inspection. Especially for low resolutions there might be slightly inaccurate results due to rounding errors.

Bibliography

[1] J. Kögel. Including the Network View in Application Response Time Diagnostics using NetFlow. Poster presentation, 2009 USENIX Annual Technical Conference, San Diego, June 2009.

[2] J. Kögel. Extracting Performance Metrics from NetFlow in Enterprise Networks. 2nd EMANICS Workshop on NetFlow/IPFIX Usage. Bremen, Germany, October 2009.

[3] J. Kögel. Characterization of Accuracy Problems in NetFlow Data and Approaches to Handle Them. IRTF NMRG 3rd NetFlow/IPFIX Workshop, IETF 78, Maastricht, July 2010.

[4] J. Kögel. Flow-based delay measurements: Error quantification and validation by active measurements. 4rd NetFlow/IPFIX Workshop, 28th IRTF NMRG Meeting, Paris, March 2012.

[5] J. Kögel and S. Scholz. Processing of Flow Accounting Data in Java: Framework Design and Performance Evaluation. In *Proceedings of the 16th EUNICE/IFIP WG 6.6 Workshop (EUNICE 2010)*, 2010.

[6] J. Kögel. One-way Delay Measurement based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling. In *Proceedings of the 25th International Conference on Information Networking (ICOIN 2011), Kuala Lumpur, Malaysia*, January 2011.

[7] T. Kleefass, S. Leinen, J. Kögel, and D. Schatzmann. Passively Detecting Remote Connectivity Issues Using Flow Accounting. 2nd EMANICS Workshop on NetFlow/IPFIX Usage, October 2009.

[8] D. Schatzmann, S. Leinen, J. Kögel, and W. Mühlbauer. FACT: Flow-based Approach for Connectivity Tracking. In *Passive and Active Measurement conference (PAM 2011)*, Atlanta, Georgia, USA, March 2011.

[9] T. Kleefass. Passively Detecting Remote Connectivity Issues Using Flow Accounting. Diploma thesis, Institute of Communication Networks and Computer Engineering, University of Stuttgart,, 2009.

[10] S. Scholz. Erweiterung und Leistungsbewertung einer Software zur Verarbeitung von Netzmessdaten. Student project (in German), Institute of Communication Networks and Computer Engineering, University of Stuttgart, 2010.

[11] M. Kling. Untersuchung des Verhaltens von TCP-Verbindungen anhand von Flow-Daten. Diploma thesis (in German), Institute of Communication Networks and Computer Engineering, University of Stuttgart,, 2010.

[12] E. S. de Rojas. Analysis and Compensation of NetFlow's Measurement Errors in Enterprise Networks. Master thesis, Institute of Communication Networks and Computer Engineering, University of Stuttgart, 2010.

[13] W. R. Stevens. *TCP/IP illustrated (vol. 1): the protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[14] *Communication Networks II lecture script*. Institute of Communication Networks and Computer Engineering, University of Stuttgart, 2008.

[15] J. Kögel. Design and Implementation of a Burst Assembly Unit based on a Network Processor. Master thesis, Institute of Communication Networks and Computer Engineering, University of Stuttgart, 2005.

[16] Cisco carrier routing system, 2011. URI: `http://www.cisco.com/en/US/prod/collateral/routers/ps5763/prod_brochure0900aecd800f8118.pdf` [Retrieved on June 25, 2011].

[17] A. Singhal and R. Jain. Terabit switching: A survey of techniques and current products. *Computer Communications*, 2002.

[18] Cisco 12000 series internet router architecture: Packet switching, 2005. URI: `http://www.cisco.com/image/gif/paws/47320/arch12000-packetsw.pdf` [Retrieved on June 25, 2011].

[19] Cisco 12000 series internet router architecture: Switch fabric, 2005. URI: `http://www.cisco.com/image/gif/paws/47240/arch12000-swfabric.pdf` [Retrieved on June 25, 2011].

[20] P. Southwick, D. Marschke, and H. Reynolds. *Junos Enterprise Routing: A Practical Guide to Junos Routing and Certification*. O'Reilly Media, 2011.

[21] S. McFarland, M. Sambi, N. Sharma, and S. Hooda. *IPv6 for Enterprise Networks*. Cisco Press, 2011.

[22] JCGM 200:2008 International vocabulary of metrology - Basic and general concepts and associated terms (VIM) [online]. URI: `http://www.iso.org/sites/JCGM/VIM/JCGM_200e.html` [Retrieved on January 10, 2011].

[23] Bureau international des poids et mesures (BIPM) Homepage [online]. URI: `http://www.bipm.org/` [Retrieved on January 10, 2011].

[24] International Organization for Standardization (ISO) [online]. URI: `http://www.iso.org/iso/home.htm` [Retrieved on January 10, 2011].

[25] International Electrotechnical Commission (IEC) Homepage [online]. URI: `http://www.iec.ch/` [Retrieved on January 10, 2011].

[26] Joint Committee for Guides in Metrology (JCGM) Homepage [online]. URI: http://www.iso.org/sites/JCGM/JCGM-introduction.htm [Retrieved on January 10, 2011].

[27] Joint Committee for Guides in Metrology (JCGM) Charter [online]. 2009. URI: http://www.bipm.org/utils/en/pdf/JCGM_charter.pdf [Retrieved on January 10, 2011].

[28] *International vocabulary of metrology - Basic and general concepts and associated terms (VIM); German-English version ISO/IEC Guide 99:2007*. Beuth Verlag, 3. edition, 2010.

[29] *JCGM 100:2008 Guide to the expression of uncertainty in measurement*. Joint Committee for Guides in Metrology (JCGM), 1st ed. edition, 2008.

[30] M. G. Cox, M. P. Dainton, , and P. M. Harris. Software Support for Metrology Best Practice Guide No. 6: Uncertainty and Statistical Modelling. Technical report, Centre for Mathematics and Scientific Computing, National Physical Laboratory (NPL), Teddington, United Kingdom, September 2006.

[31] B. R. L. Siebert and K.-D. Sommer. *Uncertainty*, volume 2 of *Handbook of Metrology*, chapter 13. Wiley-VCH, April 2010.

[32] R. Kacker. *An interpretation of the Guide to the expression of uncertainty in measurement*. U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2000.

[33] J. R. Taylor. *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. University Science Books, 2nd edition, August 1996.

[34] M. Drosg. *Dealing with Uncertainties: A Guide to Error Analysis*. Springer, 2009.

[35] *Teletraffic Theory and Engineering lecture script*. Institute of Communication Networks and Computer Engineering (IKR), University of Stuttgart, 2007.

[36] L. Fahrmeir, R. Künstler, I. Pigeot, and G. Tutz. *Statistik*. Springer-Verlag Berlin Heidelberg, 6 edition, 2007. (in German).

[37] B. Claise and R. Wolter. *Network Management: Accounting and Performance Strategies*. Cisco Press, June 2007.

[38] M. Crovella and B. Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

[39] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. V. Mieghem. Analysis of End-to-end Delay Measurements in Internet. In *Proceedings of the Passive and Active Measurement Workshop - PAM2002, Fort Collins*, 2002.

[40] A. Hernandez and E. Magafia. One-way Delay Measurement and Characterization. In *Third International Conference on Networking and Services. ICNS*, June 2007.

[41] L. De Vito, S. Rapuano, and L. Tomaciello. One-Way Delay Measurement: State of the Art. *IEEE Transactions on Instrumentation and Measurement*, Dec. 2008.

[42] D. Constantinescu and A. Popescu. Modeling of One-Way Transit Time in IP Routers. In *International Conference on Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced*, Feb. 2006.

[43] IP Performance Metrics (IPPM) WG - Charter [online]. URI: `http://datatracker.ietf.org/wg/ippm/charter/`.

[44] Cisco IOS IP Service Level Agreements Technical Overview [online]. 2004. URI: `http://www.cisco.com/en/US/technologies/tk648/tk362/tk920/technologies_brief0900aecd801e3602.ppt`.

[45] Compuware Gomez Network Performance Monitoring [online]. URI: `http://www.compuware.com/application-performance-management/gomez-network-performance-monitoring.html` [Retrieved on August 14, 2011].

[46] S. Niccolini, M. Molina, F. Raspall, and S. Tartarelli. Design and implementation of a One Way Delay passive measurement system. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, volume 1, 2004.

[47] F. Fatemipour and M. Yaghmaee. Design and Implementation of a Monitoring System Based on IPFIX Protocol. *Telecommunications, 2007. AICT 2007. The Third Advanced International Conference on*, May 2007.

[48] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese. Every Microsecond Counts: Tracking Fine-Grain Latencies with a Lossy Difference Aggregator. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM '09), Barcelona, Spain*, Aug. 2009.

[49] H. Khlifi and J.-C. Grégoire. Low-complexity offline and online clock skew estimation and removal. *Computer Networks*, 2006.

[50] IEEE1588-2008: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control System, 2008.

[51] Y. Somchit. Performance evaluation of NTPv4 when SPIK state cannot step frequency. In *International Conference on Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON)*, May 2010.

[52] C.-Y. Hong, C.-C. Lin, and M. Caesar. Clockscalpel: Understanding root causes of Internet clock synchronization inaccuracy. In *Passive and Active Measurement conference (PAM)*, 2011.

[53] V. Paxson. On Calibrating Measurements of Packet Transit Times. *SIGMETRICS Performance Evaluation Review*, 1998.

[54] S. Moon, P. Skelly, and D. Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, March 1999.

[55] L. Zhang, Z. Liu, and C. Honghui Xia. Clock synchronization algorithms for network measurements. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, 2002.

[56] O. Gurewitz, I. Cidon, and M. Sidi. One-way delay estimation using network-wide measurements. *IEEE/ACM Transactions on Networking*, 2006.

[57] R. Jain and S. Routhier. Packet Trains–Measurements and a New Model for Computer Network Traffic. *Selected Areas in Communications, IEEE Journal on*, Sep. 1986.

[58] K. Claffy, H.-W. Braun, and G. Polyzos. A Parameterizable Methodology for Internet Traffic Flow Profiling. *Selected Areas in Communications, IEEE Journal on*, Oct. 1995.

[59] S. Amante, B. Carpenter, and S. Jiang. Update to the IPv6 flow label specification. Internet-Draft draft-carpenter-6man-flow-update-04, Internet Engineering Task Force, September 2010. Work in progress. URI: `http://www.ietf.org/internet-drafts/draft-carpenter-6man-flow-update-04.txt`.

[60] B. Claise. IPFIX mailing list archive, E-Mail: Flow in IPFIX -> not only IP Flows [online]. September 2010. URI: `http://www.ietf.org/mail-archive/web/ipfix/current/msg05565.html` [Retrieved on October 4, 2010].

[61] Internet Accounting (ACCT) WG - Charter [online]. URI: `https://datatracker.ietf.org/wg/acct/charter/` [Retrieved on October 12, 2010].

[62] Realtime Traffic Flow Measurement (RTFM) WG - Charter [online]. URI: `http://datatracker.ietf.org/wg/rtfm/charter/` [Retrieved on October 12, 2010].

[63] sFlow.org forum website [online]. URI: `http://sflow.org/` [Retrieved on October 5, 2010].

[64] sFlow version 5 [online]. URI: `http://sflow.org/sflow_version_5.txt` [Retrieved on October 5, 2010].

[65] S. Löffler. Verwendung von Flows zur Analyse und Messung von Internet-Verkehr. Diploma thesis (in German), 1997.

[66] Release Notes for Cisco 7000 Family for Cisco IOS 11.1 CC [online]. URI: https://www.cisco.com/en/US/docs/ios/11_1/release/notes/rn111cc.html [Retrieved on October 12, 2010].

[67] Netflow services and applications. White Paper, 1999. No longer available under old URL, only via web archive. URI: http://web.archive.org/web/20050524045255/www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.pdf [Retrieved on October 12, 2010].

[68] E. Jasinska. sFlow - I can feel your traffic. In *23rd Chaos Communication Congress*, 2006. URI: http://events.ccc.de/congress/2006/Fahrplan/attachments/1137-sFlowPaper.pdf.

[69] Cisco Systems. *Cisco IOS NetFlow Command Reference*, 2010. URI: http://www.cisco.com/en/US/docs/ios/netflow/command/reference/nf_cr_book.pdf [Retrieved on October 19, 2010].

[70] Internet Assigned Numbers Authority (IANA) – Port Numbers [online]. URI: http://www.iana.org/assignments/port-numbers [Retrieved on October 20, 2010].

[71] Cisco Systems. *Configuring NetFlow and NetFlow Data Export*, 2010. URI: http://www.cisco.com/en/US/docs/ios/netflow/configuration/guide/cfg_nflow_data_expt.pdf [Retrieved on October 19, 2010].

[72] Netflow packet Version 5 (V5) [online]. URI: http://netflow.caligare.com/netflow_v5.htm [Retrieved on October 20, 2010].

[73] Cisco IOS Flexible NetFlow Technology [online]. URI: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/ps6965/product_data_sheet0900aecd804b590b.pdf [Retrieved on June 23, 2011].

[74] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart. Peeling Away Timing Error in NetFlow Data. In *Passive and Active Measurement conference (PAM)*, Atlanta, Georgia, USA, March 2011.

[75] G. Münz, B. Claise, and P. Aitken. Configuration Data Model for IPFIX and PSAMP. Internet-Draft draft-ietf-ipfix-configuration-model-08, Internet Engineering Task Force, October 2010. Work in progress. URI: http://www.ietf.org/internet-drafts/draft-ietf-ipfix-configuration-model-08.txt.

[76] L. Deri. nProbe: an Open Source NetFlow Probe for Gigabit Networks. In *TERENA Networking Conference*, 2003.

[77] C. Inacio and B. Trammell. YAF: Yet Another Flowmeter. In *Proceedings of the 24th Large Installation System Administration Conference (LISA 2010)*, San Jose, California, USA, November 2010.

[78] invea-tech FlowMon [online]. URI: http://www.invea-tech.com/products-and-services/flowmon/flowmon-overview [Retrieved on July 3, 2011].

[79] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2004. ACM.

[80] N. G. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. *IEEE/ACM Transactions on Networking*, 2001.

[81] S. Leinen. Flow-Based Traffic Analysis at SWITCH. In *Workshop on Passive and Active Measurements (PAM 2001)*, Apr. 2001.

[82] Plixer scrutinzer website [online]. URI: http://www.plixer.com/products/netflow-sflow/scrutinizer-netflow-sflow.php [Retrieved on March 1, 2011].

[83] Arbor Peakflow: IP Traffic Flow Monitoring System [online]. URI: http://www.arbornetworks.com/index.php?option=com_content&task=view&id=1465&Itemid=692 [Retrieved on March 1, 2011].

[84] Lancope StealthWatch system website [online]. URI: http://www.lancope.com/products/StealthWatch-System/ [Retrieved on March 1, 2011].

[85] CA NetQoS ReporterAnalyzer Product Brief [online]. URI: http://www.ca.com/~/media/Files/productbriefs/netqos_reporteranalyzer_ps_228203.pdf [Retrieved on March 1, 2011].

[86] IsarFlow network monitoring solution [online]. URI: http://isarflow.com/ [Retrieved on March 1, 2011].

[87] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview on IP Flow-based Intrusion Detection. *Communications Surveys & Tutorials, IEEE*, 2009.

[88] F. Mansmann. *Visual Analysis of Network Traffic : Interactive Monitoring, Detection, and Interpretation of Security Threats*. PhD thesis, Universität Konstanz, 2008.

[89] G. Münz. *Traffic Anomaly Detection and Cause Identification Using Flow-Level Measurements*. PhD thesis, Technische Universität München, 2010.

[90] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. A Methodology for Studying Persistency Aspects of Internet Flows. *ACM SIGCOMM Computer Communication Review*, 2005.

[91] T. Limmer and F. Dressler. Flow-based TCP Connection Analysis. In *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*, 2009.

[92] A. Dhamdhere, L. Breslau, N. Duffield, C. Ee, A. Gerber, C. Lund, and S. Sen. FlowRoute: Inferring Forwarding Table Updates Using Passive Flow-level Measurements. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, New York, NY, USA, 2010. ACM.

[93] D. Schatzmann, W. Mühlbauer, T. Spyropoulos, and X. Dimitropoulos. Digging into HTTPS: Flow-Based Classification of Webmail Traffic. In *IMC '10: Proceedings of the 10th Internet measurement conference*, Melbourne, Australia, November 2010.

[94] D. Rossi and S. Valenti. Fine-grained traffic classification with Netflow data. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, New York, USA, 2010.

[95] Y. Gu, L. Breslau, N. Duffield, and S. Sen. On Passive One-Way Loss Measurements Using Sampled Flow Statistics. In *IEEE INFOCOM 2009*, April 2009.

[96] M. Lee, N. Duffield, and R. R. Kompella. Two Samples are Enough: Opportunistic Flow-level Latency Estimation using NetFlow. In *INFOCOM'10: Proceedings of the 29th conference on Information communications*, INFOCOM'10, 2010.

[97] S. Leinen. Floma: Pointers and software [online]. URI: `http://www.switch.ch/network/projects/completed/TF-NGN/floma/software.html` [Retrieved on September 28, 2010].

[98] A. Bourges. IsarFlow, NetFlow from a tool-vendor's perspective..., October 2009. URI: `http://emanics.org/component/option,com_remository/Itemid,97/func,download/id,210/chk,bc7b06be94ca6b73a20500a6a15b11ba/fname,isarnet.pdf/`.

[99] flow-tools - Tool set for working with NetFlow data, website [online]. URI: `http://www.splintered.net/sw/flow-tools/` [Retrieved on February 13, 2011].

[100] SiLK, the System for Internet-Level Knowledge [online]. URI: `http://tools.netsa.cert.org/silk/` [Retrieved on March 1, 2011].

[101] NFDUMP - tools that collect and process netflow data, website [online]. URI: `http://nfdump.sourceforge.net/` [Retrieved on February 13, 2011].

[102] NfSen, web based front end for the nfdump netflow tools, website [online]. URI: `http://sourceforge.net/projects/nfsen/` [Retrieved on February 13, 2011].

[103] A. Kobayashi, B. Claise, G. Münz, and K. Ishibashi. IPFIX Mediation: Framework. Internet-Draft draft-ietf-ipfix-mediators-framework-09, Internet Engineering Task Force, October 2010. Work in progress. URI: http://www.ietf.org/internet-drafts/draft-ietf-ipfix-mediators-framework-09.txt.

[104] B. Trammell, E. Boschi, A. Wagner, and B. Claise. Exporting Aggregated Flow Data using the IP Flow Information Export(IPFIX) Protocol. Internet-Draft draft-trammell-ipfix-a9n-02, Internet Engineering Task Force, February 2011. Work in progress. URI: http://www.ietf.org/internet-drafts/draft-trammell-ipfix-a9n-02.txt.

[105] B. Claise. Specification of the Protocol for IPFIX Mediations. Internet-Draft draft-claise-ipfix-mediation-protocol-03, Internet Engineering Task Force, February 2011. Work in progress. URI: http://www.ietf.org/internet-drafts/draft-claise-ipfix-mediation-protocol-03.txt.

[106] S. Anderson, S. Niccolini, and D. Hogrefe. SIPFIX: A Scheme for Distributed SIP Monitoring. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, 2009.

[107] R. Hofstede, A. Sperotto, T. Fioreze, and A. Pras. The Network Data Handling War: MySQL vs. NfDump. In *Proceedings of the 16th EUNICE/IFIP WG 6.6 Workshop (EUNICE 2010)*, 2010.

[108] T. Dübendorfer, A. Wagner, and B. Plattner. A Framework for Real-Time Worm Attack Detection and Backbone Monitoring. In *IWCIP '05: Proceedings of the First IEEE International Workshop on Critical Infrastructure Protection*, Washington, DC, USA, 2005.

[109] V. Marinov and J. Schönwälder. Design of a Stream-Based IP Flow Record Query Language. In *DSOM '09: Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Venice, 2009.

[110] S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[111] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[112] A. Margara and G. Cugola. Processing Flows of Information: From Data Stream to Complex Event Processing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, New York, NY, USA, 2011.

[113] J. Kang, J. Naughton, and S. Viglas. Evaluating Window Joins over Unbounded Streams. In *Proceedings. 19th International Conference on Data Engineering, 2003.*, 2003.

[114] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing Memory Requirements for Queries over Continuous Data Streams. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, New York, NY, USA, 2002. ACM.

[115] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, June 2006.

[116] U. Srivastava and J. Widom. Memory-Limited Execution of Windowed Stream Joins. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, 2004.

[117] J. Wu, K.-L. Tan, and Y. Zhou. Window-Oblivious Join: A Data-Driven Memory Management Scheme for Stream Join. In *Scientific and Statistical Database Management, International Conference on*, 2007.

[118] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, New York, 2003.

[119] T. Plagemann, V. Goebel, A. Bergamini, G. Tolu, G. Urvoy-Keller, E. W. Biersack, and A. Bergamini. Using Data Stream Management Systems for Traffic Analysis. In *Passive and Active Measurement Workshop 2004 (PAM)*, 2004.

[120] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR 2003, First Biennial Conference on Innovative Data Systems Research*, Jan. 2003.

[121] D. Apiletti, E. Baralis, T. Cerquitelli, and V. D'Elia. Characterizing network traffic by means of the NetMine framework. *Computer Networks*, 2009.

[122] RIPE Atlas [online]. URI: `http://atlas.ripe.net/` [Retrieved on November 14, 2011].

[123] T. Zseby, S. Zander, and G. Carle. Evaluation of Building Blocks for Passive One-way-delay Measurements. In *Workshop on Passive and Active Measurements (PAM 2001)*, April 2001.

[124] P. Arlos. *On the Quality of Computer Network Measurements*. PhD thesis, Blekinge Institute of Technology, 2005.

[125] P. Arlos and M. Fiedler. A Method to Estimate the Timestamp Accuracy of Measurement Hardware and Software Tools. In *Passive and Active Network Measurement Conference (PAM 2007)*, 2007.

[126] L. R. Rabiner, M. J. Cheng, A. E. Rosenberg, and C. A. McGonegal. A Comparative Performance Study of Several Pitch Detection Algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1976.

[127] Hping - Active Network Security Tool, website [online]. URI: `http://www.hping.org/` [Retrieved on February 15, 2011].

[128] L. Wenwei, Z. Dafang, Y. Jinmin, and X. Gaogang. On evaluating the differences of TCP and ICMP in network measurement. *Computer Communications*, 2007.

[129] M. Horneffer. Assessing Internet Performance Metrics Using Large-Scale TCP SYN-based Measurements. In *Proceedings of Passive and Active Measurement Workshop (PAM'00)*, 2000.

[130] IKR Simulation Library. http://www.ikr.uni-stuttgart.de/IKRSimLib/.

[131] The Linux Foundation: netem [online]. URI: `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem` [Retrieved on November 13, 2011].

[DIN 1319-1] DIN 1319-1, Grundlagen der Meßtechnik, Teil 1: Grundbegriffe. (in German), January 1995.

[RFC 768] J. Postel. User Datagram Protocol. RFC 768, IETF, August 1980.

[RFC 791] J. Postel. Internet Protocol. RFC 791, IETF, September 1981.

[RFC 792] J. Postel. Internet Control Message Protocol. RFC 792, IETF, September 1981.

[RFC 793] J. Postel. Transmission Control Protocol. RFC 793, IETF, September 1981.

[RFC 1035] P. V. Mockapetris. Domain names – implementation and specification. RFC 1035, IETF, November 1987.

[RFC 1272] C. Mills, D. Hirsh, and G. R. Ruth. Internet Accounting: Background. RFC 1272, IETF, November 1991.

[RFC 1305] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, IETF, March 1992.

[RFC 1363] C. Partridge. A Proposed Flow Specification. RFC 1363, IETF, September 1992.

[RFC 1589] D. Mills. A Kernel Model for Precision Timekeeping. RFC 1589, IETF, March 1994.

[RFC 1633] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, IETF, June 1994.

[RFC 1812] F. Baker. Requirements for IP Version 4 Routers. RFC 1812, IETF, June 1995.

[RFC 2123] N. Brownlee. Traffic Flow Measurement: Experiences with NeTraMet. RFC 2123, IETF, March 1997.

[RFC 2205] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, IETF, September 1997.

[RFC 2330] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP Performance Metrics. RFC 2330, IETF, May 1998.

[RFC 2460] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, IETF, December 1998.

[RFC 2474] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, December 1998.

[RFC 2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.

[RFC 2679] G. Almes, S. Kalidindi, and M. Zekauskas. A One-way Delay Metric for IPPM. RFC 2679, IETF, September 1999.

[RFC 2681] G. Almes, S. Kalidindi, and M. Zekauskas. A Round-trip Delay Metric for IPPM. RFC 2681, IETF, September 1999.

[RFC 2720] N. Brownlee. Traffic Flow Measurement: Meter MIB. RFC 2720, IETF, October 1999.

[RFC 2721] N. Brownlee. RTFM: Applicability Statement. RFC 2721, IETF, October 1999.

[RFC 2722] N. Brownlee, C. Mills, and G. Ruth. Traffic Flow Measurement: Architecture. RFC 2722, IETF, October 1999.

[RFC 2723] N. Brownlee. SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups. RFC 2723, IETF, October 1999.

[RFC 2783] J. Mogul, D. Mills, J. Brittenson, J. Stone, and U. Windl. Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0. RFC 2783, IETF, March 2000.

[RFC 2819] S. Waldbusser. Remote Network Monitoring Management Information Base. RFC 2819, IETF, May 2000.

[RFC 2991] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991, IETF, November 2000.

[RFC 2992] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, IETF, November 2000.

[RFC 3031] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, IETF, January 2001.

[RFC 3168] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, IETF, September 2001.

[RFC 3176] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, IETF, September 2001.

[RFC 3234] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234, IETF, February 2002.

[RFC 3410] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410, IETF, December 2002.

[RFC 3654] H. Khosravi and T. Anderson. Requirements for Separation of IP Control and Forwarding. RFC 3654, IETF, November 2003.

[RFC 3697] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering. IPv6 Flow Label Specification. RFC 3697, IETF, March 2004.

[RFC 3758] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758, IETF, May 2004.

[RFC 3917] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, IETF, October 2004.

[RFC 3954] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, IETF, October 2004.

[RFC 3955] S. Leinen. Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX). RFC 3955, IETF, October 2004.

[RFC 4271] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, IETF, January 2006.

[RFC 4443] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, IETF, March 2006.

[RFC 4656] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas. A One-way Active Measurement Protocol (OWAMP). RFC 4656, IETF, September 2006.

[RFC 4960] R. Stewart. Stream Control Transmission Protocol. RFC 4960, IETF, September 2007.

[RFC 5101] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, IETF, January 2008.

[RFC 5102] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer. Information Model for IP Flow Information Export. RFC 5102, IETF, January 2008.

[RFC 5103] B. Trammell and E. Boschi. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103, IETF, January 2008.

[RFC 5357] K. Hedayat, R. Krzanowski, A. Morton, K. Yum, and J. Babiarz. A Two-Way Active Measurement Protocol (TWAMP). RFC 5357, IETF, October 2008.

[RFC 5424] R. Gerhards. The Syslog Protocol. RFC 5424, IETF, March 2009.

[RFC 5470] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for IP Flow Information Export. RFC 5470, IETF, March 2009.

[RFC 5472] T. Zseby, E. Boschi, N. Brownlee, and B. Claise. IP Flow Information Export (IPFIX) Applicability. RFC 5472, IETF, March 2009.

[RFC 5475] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. Sampling and Filtering Techniques for IP Packet Selection. RFC 5475, IETF, March 2009.

[RFC 5476] B. Claise, A. Johnson, and J. Quittek. Packet Sampling (PSAMP) Protocol Specifications. RFC 5476, IETF, March 2009.

[RFC 5477] T. Dietz, B. Claise, P. Aitken, F. Dressler, and G. Carle. Information Model for Packet Sampling Exports. RFC 5477, IETF, March 2009.

[RFC 5905] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, IETF, June 2010.

[RFC 5982] A. Kobayashi and B. Claise. IP Flow Information Export (IPFIX) Mediation: Problem Statement. RFC 5982, IETF, August 2010.

[Y.1540] ITU-T. Internet protocol data communication service - IP packet transfer and availability performance parameters. Rec. Y.1540, ITU-T, December 2002.

[Y.1543] ITU-T. Measurements in IP networks for inter-domain performance assessment. Rec. Y.1543, ITU-T, November 2007.

Acknowledgements

This thesis completes my six year stage at the Institute of Communication Networks and Computer Engineering (IKR) of the University of Stuttgart, Germany. I enjoyed the broad expertise at the institute and the opportunity of being involved in several challenging activities. This includes national and international projects as well as collaboration with students and colleagues, where I could gain experience in project work and a deep understanding of computer architecture and network technology. Although the time at the institute was sometimes hard, I was highly motivated by the technical and scientific challenges as well as by the good working atmosphere and fun with colleagues.

I would like to thank Prof. Paul Kühn for the opportunity to join the IKR, his confidence in my work, and the supervision of the thesis. Moreover, I would like to thank Prof. Andreas Kirstädter for reviewing the thesis, and Prof. Georg Carle for reviewing the thesis, his interest in the topic and open-minded discussions during my visit at TU München.

I am very thankful to all colleagues at the IKR for discussing topics of this thesis and in general for providing a friendly and constructive atmosphere. In particular, I want to thank the members of the former services and security group and its successor, the fixed network group: Christian Hauser, Martin Neubauer, Andreas Reifert, Marc Barisch, Joachim Scharf, Arthur Mutter, Mirja Kühlewind, Sebastian Meier, David Wagner, Frank Feller, Domenic Teuchert, and Ulrich Gemkow. Special thanks to Thomas Werthmann, Magnus Proebster, Christian Blankenhorn, Sebastian Meier, Frank Feller, Joachim Scharf, Marc Barisch, and Ulrich Gemkow, who reviewed drafts of this thesis. During my time at the institute I enjoyed a lot working in different projects, different research areas, and always found interested colleagues for more general discussions. In this context, I want to thank my office mates Wolfgang Payer and Christian Müller, as well as Sebastian Kiesel, Michael Scharf, Matthias Kaschub, Simon Hauger, Marc Necker, Detlef Sass, and Martin Köhn. Finally, I would like to thank Ulrich Gemkow for the freedom, critical feedback, and support for all activities at the IKR, not only in the context of this thesis.

Several students worked in the research area of this thesis and contributed with discussions, parts of software, or interesting questions during their Diploma, Master, Bachelor or other kind of projects. Especially, I want to thank Sebastian Scholz, Tim Kleefass, Mohammed Ali Seblani, Markus Kling, and Emilio Sánchez de Rojas.

The topic of this thesis originated in a project with Robert Bosch GmbH. I want to thank everybody who was involved in this project for interesting insights and discussions. Especially I want to thank Anja Moog-Lölkes and Dirk Ernst, who supported the recording of measurement

data and gave feedback on the results. While I developed the ideas of this thesis, I got valuable and motivating feedback from the network monitoring community. In particular, I want to thank Benoit Claise, Paul Aitken, Aiko Pras, and Brian Trammell for their feedback around NMRG workshops, as well as Simon Leinen and Dominik Schatzmann for the joined work in flow monitoring. It was great to meet and learn from various people in national and international projects, such as Scalenet, Daidalos and a bilateral projects with Deutsche Telekom AG. I would like to thank all of them, especially Gerhard Schröder, Stefan Wahl, Joao Girao. Jean-Michel Combes, Carsten Schmoll, and Jürgen Jähnert.

Finally, I want to thank my family. Many thanks to my wife Simone for her patience and motivation, especially in the final stage of writing this thesis. I want to thank my parents who formed and supported my technical and interested spirit from my early days, and who gave me the opportunity to decide for my studies and supported this academic way at the university.