Increasing Packet Sizes to Mitigate Performance Issues in High-Speed Packet Networks

Frank Feller, Joachim Scharf
Institute of Communication Networks and Computer Engineering (IKR),
Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
Email: {frank.feller, joachim.scharf}@ikr.uni-stuttgart.de

Abstract

Emerging packet-switched transport networks face a continuous growth of link bitrates. Without counter-measures, this translates into a proportional increase of the packet rate and may turn packet processing in network nodes into a bottleneck. In this paper, we investigate one approach to reduce the processing load for a given line rate: increasing the maximum size of packets exchanged between end systems. We first present the fundamental mechanisms along with side effects of increased packet sizes. Then we discuss the dependence of the achievable packet rate reduction on application and protocol stack. Based on a traffic model from the year 2000 and current measurements, we finally evaluate the reduction for web traffic and the Internet protocol stack. Results show that today's web traffic enables higher packet rate reductions than traffic of 2000. A fourfold packet size increase now allows to cut the packet rate by two thirds.

1 Introduction

New applications and services are driving a continuous growth of the bandwith demand in transport networks. Efforts to increase link bitrates are therefore undertaken in research and development, currently targeting 100 Gbps in the case of Ethernet [1]. Since packetized data has grown predominant in transport networks in recent years, we are witnessing a shift from traditional *synchronous time division multiplex* (STDM) systems to packet-switching architectures such as carrier grade Ethernet solutions.

Without adaptation of the packet format, an increase of the line rate directly translates into an increase of the packet rate. This in turn raises the requirements on packet processing network nodes. Accordingly, high-end network nodes designed for the upcoming transmission speeds in transport networks will be expensive, if the processing requirements can be fulfilled at all. Means to reduce the packet processing load relative to the line speed are therefore desirable.

Two complementary approaches allow for such processing load reduction. First, we can minimize the effort spent on an individual packet by simplifying protocol mechanisms. According options are mentioned in [2]. Second, we can increase the amount of data transported per processed packet. There are three variants of doing so:

- Aggregation of packets following the same path through the transport network. This reduces the load on core nodes which only have to process one header for the aggregate [3].
- Increasing the minimum size of packets transported between end systems. This allows to reduce the worst-case packet rate at the expense of a waste of transmission capacity. Small data units are padded with void data.
- Increasing the maximum size of packets transported between end systems. Thereby, the average packet rate is reduced. However, the worst-case packet rate remains unaffected.

Figure 1 visualizes the effect of these variants. In this paper, we focus on the last approach, the increase of the maximum packet size. The packet rate reduction achieved by this means depends on two factors: the application, which defines the size of transmitted data blocks, and the protocols in charge of enabling the communication. We discuss these influences and investigate their effect by the example of web traffic and the Internet protocol stack.

The remainder of this paper is structured as follows. We present current packet size limits along with principle benefits and drawbacks of their increase in Section 2. Sections 3 and 4 then discuss the influence of protocols and describe web traffic as the exemplary application in our analysis, respectively. We provide results in Section 5 and conclude this paper in Section 6.

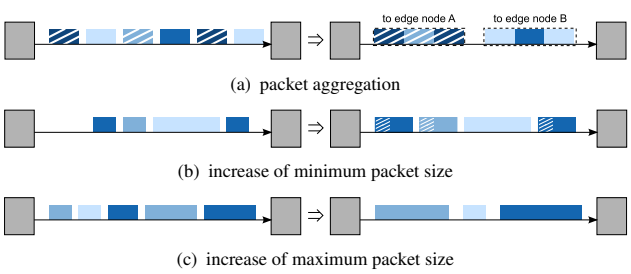

Figure 1 Approaches to reduce the packet rate

2 Packet Sizes

When a packet is forwarded through a network, all nodes on its path have to be able to handle it, with respect to both its format and its size. Therefore, most packets observed in today's networks still obey technological limitations and design choices dating back several decades. Ethernet frames, as defined in the early 1980s, are able to carry up to 1500 bytes of payload data. For compatibility reasons, this limitation has been retained in all subsequent Ethernet standards. Since most hosts connected to the Internet are locally attached to Ethernet networks, 1500 bytes has become the de-facto maximum packet size. For the *Internet Protocol* (IP), RFC879 from 1983 cites a packet size of 576 bytes each IP host has to be able to receive [4]. Conversely, no host shall send larger packets unless it has knowledge that the recipient is able to accept them. Protocol implementations not performing such a negotiation still limit packet sizes to 576 bytes. Together with pure signaling packets of about 40 bytes, these two limits make up the well-known tri-modal distribution of IP packet sizes [5].
These packet size limitations, however, seem anachronistic given the development of both computing power and memory sizes over the last quarter century. Home computers now support multimedia applications handling files of sizes unthought of a few years ago. Networking technology has kept up with this development. For instance, the IEEE working group 802.3ba [1] is currently addressing 40 Gbps and 100 Gbps transmission for Ethernet, up from 10 Mbps in the early 1980s. Accordingly, the transmission time of a 1500 byte frame is being reduced from 1.2 ms to 12 ns. This raises the question whether the currently observed packet size limitations are still reasonable.
Parts of the deployed protocols and network equipment do actually support larger packets. Already in version 4 of IP (IPv4), the packet size is only limited by the maximum value of the 16 bit length field, i. e. to 65,535 bytes. With Gigabit Ethernet, equipment vendors started to support so-called *Jumbo Frames*, i. e. non-standard Ethernet frames of generally up to 9000 bytes. They were motivated by the inability of contemporary end systems to exploit the available link bitrate and are used in closed scenarios like data centers. The authors of [6] describe a case where the activation of Jumbo Frames brought down CPU usage for network transmission from 100 % to 55 % while increasing the throughput by 50 % on 300 MHz Sun servers.
Although not always feasible, backward compatibility to the equipment base installed in the field is a major requirement for network standards and protocols. Therefore, changes of the packet size are difficult to achieve. This, however, should not rule out the study of potential benefits of larger packets.

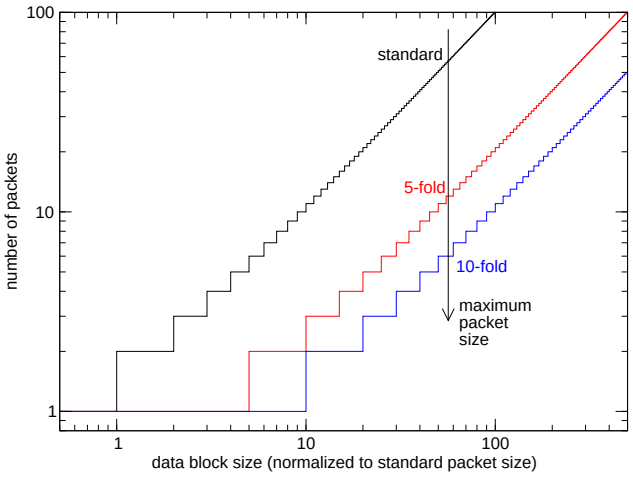

Figure 2 Dependency of the number of packets on the packet size and the transferred data block size

2.1 Benefits of Packet Size Increase

The basic use of a network is to assure data transport between two or several end points on behalf and request of some application. This application determines the characteristics of the data blocks it exchanges, including their size. Since the packets, i. e. the data units transported through the network, are generally limited in size, application data blocks need to be segmented into pieces fitting into the packets.
The number of packets required for the transfer of one data block depends on the data block size and the maximum packet size. The larger the maximum packet size, the less packets are required. However, if only small data blocks are exchanged, higher packet size limits cannot provide any benefit. Figure 2 visualizes these effects. It plots the number of packets over the data block size for different maximum packet sizes. For small data blocks, the already low number of standard-size packets limits the packet count reduction obtained from increased packet sizes. The theoretical reduction by factors corresponding to the packet size increase, i. e. five or ten respectively, can only materialize for large data blocks.
While the focus of this paper is on packet rate reduction, we have so far only considered the number of packets for a certain data transfer. If a given amount of data is transferred in a time interval, however, the reduction of the number of packets directly translates into a proportional reduction of the average packet rate.

2.2 Issues and Limitations

Besides the practical problem of backward compatibility when introducing extended packet sizes into an existing network infrastructure, further basic issues have been brought up in the discussion about Jumbo Frames. They are summarized e. g. in [7].
One set of issues revolves around timing behavior. Basically, the time required for the transmission of a packet

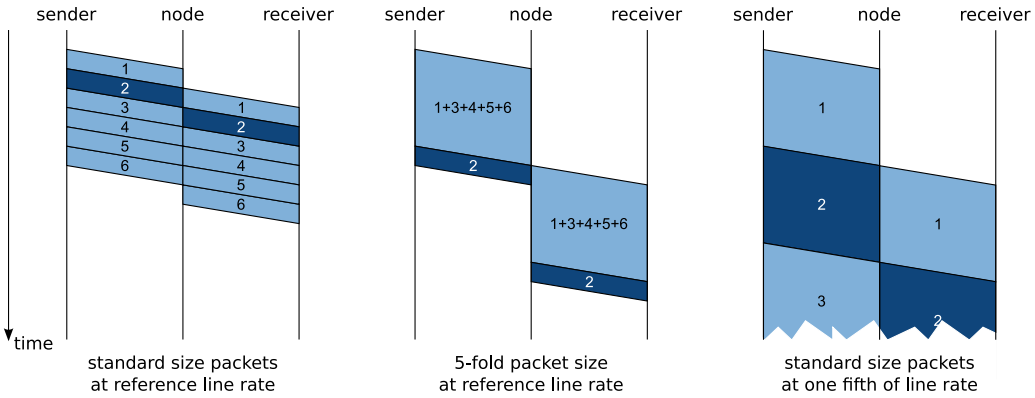

Figure 3 Impact of packet size and line rate on transfer times

over a link grows with the packet size. Network nodes following the wide-spread *store and forward* principle only relay a packet to the next link after having received it completely. Therefore, data transported in large packets experiences longer transmission delays than data segmented into smaller packets. Figure 3 details this effect by the example of the transmission of two data blocks (shaded lightly and darkly) from a sender over one intermediate node to a receiver under various conditions. On the left, small packets are used, and the intermediate node can already forward the first packet while receiving the second one. In the middle, large packets are used. Due to the need to receive the entire packet before forwarding it, the transmission time grows.

While applications exchanging large blocks of data may tolerate such delays, the effect is particularly problematic for real-time applications competing with bulk data transfers for transmission resources. For illustration, the darker packet in Figure 3 shall belong to a real-time application and be queued for transfer while the first bulk data packet is sent. Since it has higher priority, it is transferred immediately after this packet in the left-hand-side case. If large packet sizes are authorized and preemption is not possible, the real-time packet has to wait for the completion of the bulk transfer and suffers a considerably longer delay (Figure 3, center). The delay variation experienced by both bulk data and real-time applications is similarly subject to adverse effects from packet size enlargement.

The delay penalties, however, have to be considered in the context of the line rate. As shown on the right hand side of Figure 3, the transmission of small packets on an accordingly slower link takes as much time as the transmission of large packets on the fast one. Consequently, delay issues are negligible when the packet size increase coincides with an increase of the link rate. Conversely, we have to watch out for the timing performance on bottleneck links. Since the packet size increase has an end-to-end scope, access links are likely bottlenecks. Nevertheless, the general trend of bandwidth growth is also present in access networks, promising to mitigate the issue.

Bit errors potentially corrupting packets during transmission are another concern. They necessitate error detection and correction, which is often deployed in layer 2. Such mechanisms need adaption to larger packets for two reasons. First, larger data units run a higher risk of being corrupted. Second, detection and correction capabilities of error protection codes deteriorate with increasing amounts of secured data. For instance, the cyclic redundancy code protecting Ethernet frames loses its ability to securely detect three bit errors when applied to more than 11,450 bytes of data. In addition, if packet corruption triggers retransmission, larger packet sizes increase the amount of data to be resent.

Finally, the maximum packet sizes supported throughout a network may vary. Such heterogeneous environments require mechanisms to cope with the diversity without generally resorting to the smallest common packet size. On the one hand, the maximum supported packet size could be determined for each path by means of appropriate protocol mechanisms. On the other hand, segmentation and reassembly functions could be provided within the network in order to adapt packet sizes at the edges of sections allowing only smaller packets.

While the previously discussed limitations have to be taken into account for an overall evaluation of the approach of increasing packet sizes, they need to be considered in relation to the potential benefits. In the remainder of this paper, we focus on the latter.

3 Protocol Functions

The transport of data through a network requires a set of mechanisms which are implemented by different protocols. For instance, the segmentation of data blocks into packets described in Section 2.1 is actually performed by a protocol instance. Further protocol mechanisms also influence the packet rate. In the first subsection, we describe the general functionality of a protocol layer. We then discuss the real-

ization of these mechanisms in the Internet protocol stack in Section 3.2 and finally outline the assumptions underlying our evaluation in Section 3.3.

3.1 Protocol Layer Mechanisms

In a layered network architecture, functionality is distributed among several protocol layers. While each layer actually fulfills distinct tasks, we are able to abstract from these functions in a general network model [8]. A protocol layer is then characterized by a set of generic mechanisms. Figure 4 shows the basic functionality. Layer N+1 hands a *Protocol Data Unit* (PDU) containing both signaling information and payload data from higher layers to layer N. Layer N may process this data, which is now called *Service Data Unit* (SDU). It then complements the SDU with *Protocol Control Information* (PCI), i. e. signaling data, in order to form a layer N PDU. This PDU is finally handed down to layer N–1 for transmission.

In addition to such PDUs carrying data from higher layers, layer N may create pure signaling PDUs (i. e. PDUs only containing PCI) if required by a protocol mechanism. Besides, protocol layers providing reliable communication may have to resend PDUs in response to packet losses.

The size of the PDUs a protocol layer can hand down to lower layers may be limited. If the (N+1)-PDU, together with the N-PCI, exceeds the maximum size of N-PDUs layer N–1 accepts, segmentation is required. Layer N then has to segment the N-SDU, such that the resulting pieces (together with the N-PCI) respect the size limitation of layer N–1. Figure 5 depicts this mechanism. The decrease of the number of N-PDUs by increasing their maximum size is visualized in Figures 6a and 6b. Figure 6c shows how small (N+1)-PDUs prevent the decrease.

3.2 Internet Protocol Stack

Since web traffic is carried over the Internet protocol stack, we need to consider the according protocols from the networking layer up to the application layer in more detail. For our analysis, we can reduce *Medium Access Control* (MAC) layer functionality to providing transport of data containers of a certain size. Further MAC protocol mechanisms can thus be disregarded. Since the application layer protocol is addressed in Section 4, the remaining relevant protocols are the *Internet Protocol* (IP, [9]) and the *Transmission Control Protocol* (TCP, [10]).

Both TCP and IP add PCI to the payload data. Headers of each of these protocols range between 20 bytes and 60 bytes in size, depending on optional fields. Another commonality of these protocols is the ability to segment SDUs, though with different focus. IP has to respect MAC layer frame size limitations and accordingly fragments larger packets if required. TCP, in contrast, has been designed for the transmission of continuous data streams, i. e. the SDUs TCP receives may be very large. Hence, the segmentation of data into pieces of the *Maximum Segment Size* (MSS) is part of the basic TCP functions. Since segmentation on several protocol layers is inefficient, the MSS is generally chosen such that the *Maximum Transmission Unit* (MTU), i. e. the MSS plus TCP and IP headers, does not exceed the MAC layer frame size.

A problem arises when the frame size limit varies along the transmission path and thus the appropriate MTU is not known to the sender. In 1990, the IETF defined a *Path MTU Discovery* mechanism [11]. It relies on sending unsegmentable packets and reducing their size in response to error messages. However, this mechanism does not work reliably as error messages are often filtered.

Further features provided by TCP include flow control and reliable communication. Thereto, the protocol uses connections requiring three signaling packets for setup and four packets for teardown. Essentially, TCP flow control limits the number of packets sent in a time window. Therefore, the use of larger packets likely turns its behavior more aggressive. A detailed study of this implication is up to future work.

Reliable data transmission is achieved by retransmission of

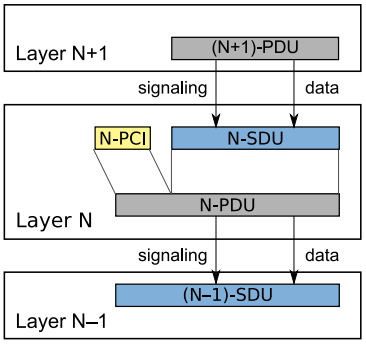

Figure 4 Layer network model

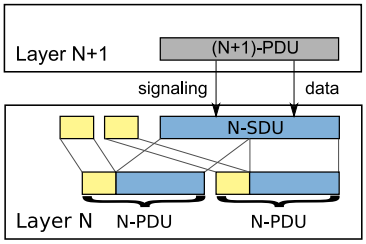

Figure 5 Segmentation

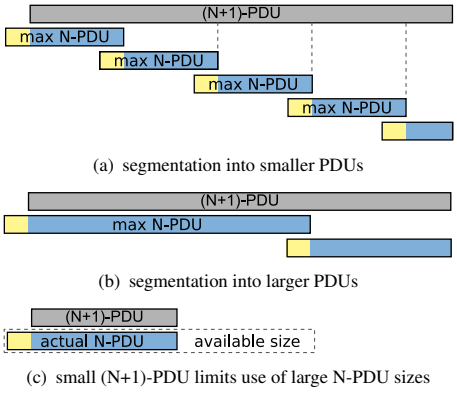

Figure 6 Effects of different PDU sizes

lost packets. In order to detect packet losses, TCP acknowledges the reception of packets to the sender. This is either accomplished by means of separate signaling packets, or by including signaling information into data packets. The latter, referred to as *piggybacking*, requires data to be transmitted to the initial sender at the time the acknowledgment has to be sent. Therefore, separate *acknowledgment packets* (ACKs) are frequent. Figure 7 presents TCP signaling in an exemplary transaction. Depending on the TCP implementation, an ACK may be sent for one or several received packets. For our analysis, however, the important observation is that the number of ACKs is roughly proportional to the number of data packets.

3.3 Assumptions for Evaluation

For our analysis, we modeled the influence of TCP and IP as follows. The headers of both protocols combined make up 40 bytes, i. e. there are no optional fields. TCP connections incur an overhead of seven packets, three for setup, four for teardown. For each block of transmitted payload data, we send an ACK for the first data packet, and afterwards one for every second data packet. All signaling is done by extra packets, i. e. there is no piggybacking. Finally, we focus on the error-free case, i. e. no packet losses and retransmissions occur.

4 Web Traffic Description

As pointed out in Section 2.1, the size of the data blocks exchanged by the application limits the packet rate reduction obtained through packet size increase. Therefore, we introduce the principle mechanisms of the *Hypertext Transfer Protocol* (HTTP, [12]), the application layer protocol carrying web traffic, in the following. We then discuss the two data sources underlying our evaluation: a statistical model of the *static web* of the 20th century in Section 4.2, and our own measurements of today's web traffic in Section 4.3.

4.1 HTTP Features

Initially, the web consisted of static *Hypertext Markup Language* (HTML, [13]) documents interconnected by *hyperlinks*. Each HTML document described the text and structure of a web page, and the hyperlinks served to navigate between such pages. An HTML document could reference files defining objects embedded in the page, e. g. images. For increased flexibility, this mechanism has been extended to compose web pages of several HTML documents.

The basic purpose of the HTTP protocol is the retrieval of HTML documents and other web objects. It thereto centers on a *request–response* model: A client, e. g. a web browser, sends an HTTP request specifying the desired object by means of a *Uniform Resource Locator* (URL) to the (web) server. Besides basic requests for some resource, further request types exist. For instance, post requests allow to upload data entered in a web form. The HTTP response sent by the server contains a status code and, in case of a successful request, the demanded object. Otherwise, the status code specifies the error. Figure 7 outlines a successful HTTP transaction embedded in a TCP connection.

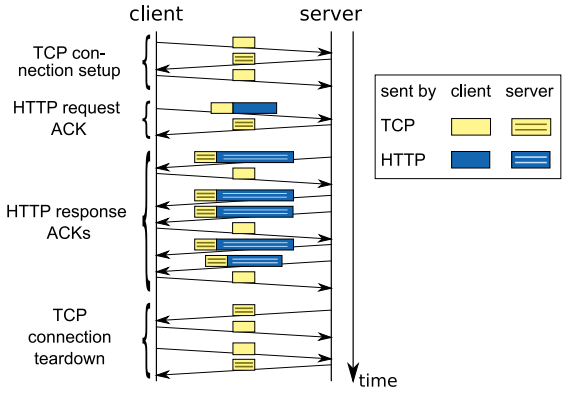

Figure 7 Exemplary HTTP transaction over TCP

Since web pages often contain considerable numbers of embedded objects, their retrieval requires many HTTP transactions. Establishing a TCP connection for each request is not desired due to setup delays and initially low data rates. Therefore, HTTP's *connection keep-alive* option allows client and server to negotiate the reuse of an established connection for subsequent transactions. Connections are finally torn down when a timer expires, after a certain number of transactions, or when the client would otherwise exceed a maximum number of active connections. The mechanism is widely used today. Nonetheless, clients do establish small numbers of parallel connections to the same server to speed up retrieval.

Request *pipelining* is a further optimization of HTTP. It allows a client to send a number of requests back to back. The server then returns the responses in a single block, too. Compared to connection keep-alive where a further request is only sent after the completion of the previous transaction, pipelining reduces the impact of propagation delays. Unlike the former, it is of little practical importance since unsupported by wide-spread browser implementations or deactivated by default. We therefore disregard pipelining in our analysis.

Caching proxy servers located at the gateway connecting a local area network to the Internet are a means of reducing HTTP traffic. They receive all HTTP requests from the local clients and initially forward them to the respective original web servers. By caching the responses while relaying them back to the requesting clients, the proxies are able to directly answer subsequent requests for the same objects. Thereby, they shorten response times for the clients and reduce Internet traffic. The ability of proxies to log information on the served requests is of interest for our study.

Running HTTP over *Transport Layer Security* (TLS, [14])

protected connections enables the exchange of sensitive information. This mechanism, referred to as *HTTP Secured* (HTTPS), relies on a trust relation between the client/user and the web server – and on end-to-end encryption. A proxy therefore has to relay HTTPS packets between client and server without processing or inspecting them. Thereto, it has to establish a forwarding relation, which is initiated by the client. The proxy is thus unable to log details on the data exchange within the HTTPS connection.
In recent years, the web itself has been evolving from the *static web* outlined above to more dynamic mechanisms like *Asynchronous Javascript And XML* (AJAX, [15]). Previously, the web browser retrieved the main HTML document upon user interaction and afterwards all embedded objects. With AJAX, in contrast, the browser only transfers the frame HTML document and a piece of Javascript, which issues subsequent HTTP requests. The script code can dynamically determine the required objects and continue with transfers in the background after the currently required parts of the page are retrieved. Thereby, it may provide for better responsiveness of web sites and improve user experience. The deployment of AJAX has changed HTTP traffic characteristics [16]. Another recent observation is that objects embedded in one web page have to be retrieved from a number of different servers.

4.2 *Static Web* Model

We based the first part of our evaluation on an extensive statistical model of web traffic providing distribution functions for both timing and transfer size metrics [17]. It has been derived from packet level traces collected in November 2000.
The information essential for our analysis is the size of HTTP requests and responses, along with the number of TCP connections to determine the overhead. According to the model, almost every request fits into a single 1500 byte packet. For simplicity, we thus assume that requests are made up of only one packet regardless of the allowed packet size. The average number of response packets has been computed from the response size distribution under the assumption that all but the last packet fully exploit the maximum packet size.
Unfortunately, the model does not give any indication about TCP connections. We therefore assume that all embedded objects are transferred within the same TCP connection as the main document. Therewith, we obtain an average of 5.29 transactions per connection.
Due to its age, the traffic model bases on measurements of the former static web. Thus, it is unclear to what extend it still covers today's traffic patterns. We therefore performed new measurements of HTTP traffic and derived its characteristics.

4.3 Proxy Log File Evaluation

Basically, we are interested in a description of web traffic occurring between clients and web servers. Directly measuring at a client or server, however, would limit both the amount of retrievable data and its generalizability. More representative data could be extracted from packet-level traces collected within the network. Though, the reconstruction of HTTP transactions from such traces is a complex task. Proxy servers handling requests from different clients to different web servers see a similar diversity of traffic and are able to log summaries of HTTP transactions. We therefore referred to information provided by proxies.
We obtained anonymized log files collected at a dormitory network during two weeks in January/February 2009. After removing entries referring to erroneous, non-HTTP, or HTTPS connection requests, we computed the average number of response packets from the HTTP response sizes provided in the log file. Since the request sizes are not part of the log, we again assumed a single packet per request.
In principle, the logged information allows to trace TCP connections. Being established between clients and the proxy server, however, these connections do not describe normal client–server behavior. For instance, objects from different origin servers may be transferred through one connection. We therefore estimated the number of TCP connections by means of a time-out based heuristic. We assumed that all successive requests issued from one client to one server are served within the same connection. This connection times out when no further request is sent in a specified period of time. Thus, we do not allow parallel connections between one client–server pair.
We applied two different time-out values. The first one, 3 s, shall limit connections to transfers of single web pages and thus permit a comparison with the statistical model where we made the same assumption. The second one, 120 s, is a realistic value derived from observation of actual web servers and browsers. It allows TCP connections to span several web page transfers.

5 Results

Based on the HTTP response size distributions of the statistical traffic model and the proxy log files according to Sections 4.2 and 4.3 respectively, we first computed the average number of packets required to transmit the HTTP response depending on the MTU. Under the assumptions stated in these sections and in Section 3.3, we therefrom deduced the average number of packets needed for the entire HTTP transaction in three scenarios: the static web of the year 2000, and today's web with 3 s and 120 s time-outs. In addition to the response packets, the transaction thus comprises the HTTP request along with TCP connection signaling and ACKs. Technically, we therewith compute packet counts. The reduction ratios we obtain, however, equally apply to packet rates (cf. Section 2.1).

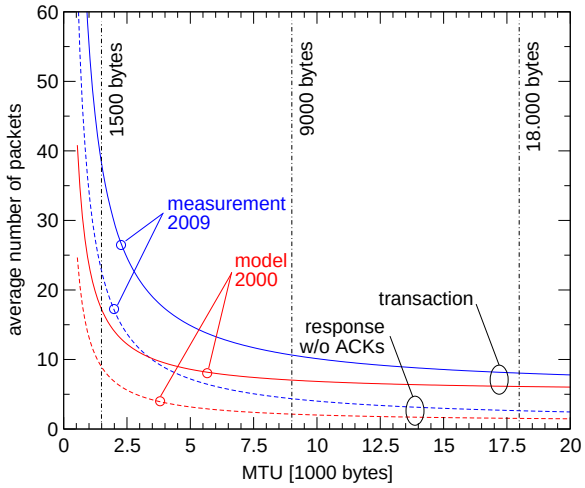

Figure 8 Packets required for HTTP transaction resp. HTTP response with TCP connections only spanning one page transfer, over MTU size

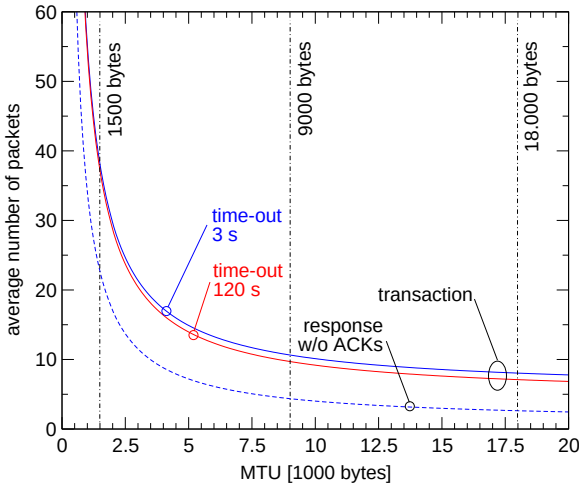

Figure 9 Packets required for HTTP transaction resp. HTTP response according to proxy log measurements, over MTU size

Figure 8 compares the results for the statistical model with the measurements for the 3 s time-out. It plots the average number of packets required for HTTP responses and complete transactions over the MTU. All curves exhibit a saturation effect: As more and more requests can be served by a single response packet, the packet count reduction with increasing MTU is limited. While the responses actually converge towards one packet, the transaction curves approach higher limits due to the MTU-independent overhead of HTTP requests and TCP connection signaling. This overhead is slightly more important for the proxy measurements where on average, only 4.39 requests were served per TCP connection, compared to 5.29 for the statistical model.

The response sizes show a more significant difference between the static web of the year 2000 and today's traffic. The average size grew from 12,074 bytes in 2000 to 32,198 bytes today. This results in considerably more moderate-sized packets being required to transfer an HTTP response of 2009, which in turn enables higher packet count reduction ratios. An increase of the MTU from state-of-the-art 1500 bytes to Jumbo Frame inspired 9000 bytes and further to 18,000 bytes yields response packet reductions of 80.7 % and 88.4 % respectively for today's traffic. For the static web model, we obtain somewhat smaller values, namely 76.0 % and 82.9 %. The MTU-independent part of the packet overhead limits the gain for transactions. The cited MTU increases only yield packet count reductions of 72.1 % and 78.9 % respectively for HTTP transactions of 2009. For the web traffic model of 2000, we obtain 59.1 % and 64.4 %. The adverse effect of the signaling overhead on the packet count reduction is thus significant, but not dramatic.

Since an MTU increase has negative side effects (cf. Section 2.2), arbitrarily large packets are generally not desirable. MTU sizes rather have to be subject to a trade-off between the packet rate reduction and timing or packet corruption penalties. The characteristics of the packet counts in Figure 8 give first indications of reasonable target MTU sizes, since moderate packet size increases already account for the lion's share of packet reduction. For the statistical static web model, an MTU of 4000 bytes seems to be reasonable, yielding a 44.8 % reduction of transaction packets over a 1500 byte MTU. For the larger web objects of 2009, an MTU of 6000 bytes appears appropriate and reduces transaction packets by 65.1 %.

The plots in Figure 9 detail the findings of the proxy log analysis. They contrast the packet count curves for HTTP responses and transactions assuming a 3 s time-out (already present in Figure 8) with the results for a connection time-out of 120 s. Since the time-out only influences the number of connections and thus the mean signaling overhead per transaction, the response curves are identical. The difference in the number of packets per HTTP transaction is not dramatic, either. Although the average number of transactions per TCP connections increases to 10.7 with 120 s time-outs, the per-transaction overhead is thereby only reduced by approximately one packet. For the reasonable target MTU size of 6000 bytes, we thus obtain a packet count reduction of 66.8 % compared to an MTU of 1500 bytes.

6 Conclusion

In this paper, we first discussed benefits and drawbacks of an increase of the maximum packet size. We then detailed the impact of protocol mechanisms on the packet rate reduction achievable by this means, for both a general architecture and the Internet protocol stack. The description

of web traffic and the HTTP protocol finally prepared the ground for the results of our analysis of the packet count – and therewith the packet rate – reduction for this application.

For current web traffic, a moderate increase of the MTU by a factor of four, from widely used 1500 bytes to 6000 bytes, would reduce the packet rate by 66.8 %. Further important packet rate reductions would require a disproportionate increase of the MTU and thus aggravate issues like timing performance and packet loss probability. The reduction is bounded at 87.5 % obtained for unlimited packet sizes.

The feasible packet rate reduction grows with the size of transferred data blocks. The comparison of the web traffic model of the year 2000 with current measurements exhibits a trend of increasing web object sizes. We can thus expect moderately higher gains for future web traffic. Significant packet rate reductions are more likely to be achieved for future, e. g. multimedia, applications transferring large blocks of data.

Further research is required in two areas in order to comprehensively assess the approach to reduce the packet rate by increasing packet sizes. First, the investigation has to be extended to a range of additional applications including file transfer, peer-to-peer file sharing, and video streaming. This will finally allow to determine the gross benefit for a realistic traffic mix. Second, side effects of larger packets like timing penalties and cross-layer interactions need to be studied and quantified. Combined, these studies will enable an overall evaluation of the approach. In addition, the migration to extended packet sizes is an open issue, since all network devices including end systems as well as applications need to be adapted to support large packets.

Packet rate reductions of several orders of magnitude, however, will hardly be achieved by solely increasing the MTU. To this end, aggregation of packets at the edge of the transport network is more promising. The concatenation of packets (including signaling messages) from several applications and end systems has larger potential to reduce the number of transmitted data units than the enlargement of packets of individual application instances. The higher packet rate reduction, however, comes at the cost of complex network nodes performing aggregation and disaggregation at the network's edge.

Acknowledgments

The authors would like to thank Martin Köhn for valuable discussions.

The work presented in this paper was partly funded within the 100GET project 100G ARP by the German Bundesministerium für Bildung und Forschung under contract No. 01BP0768.

References

[1] IEEE Computer Society. P802.3ba: IEEE Standard for Local and Metropolitan Area Networks– Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Amendment: Media Access Control Parameters, Physical Layers and Management Parameters for 40 Gb/s and 100 Gb/s Operation, 2008.

[2] S. Hauger, T. Wild, A. Mutter, A. Kirstädter, K. Karras, R. Ohlendorf, F. Feller, and J. Scharf. Packet processing at 100 Gbps and beyond—challenges and perspectives. In *Proceedings of the 10. ITG Symposium on Photonic Networks*, Leipzig, May 2009.

[3] A. Mutter, M. Köhn, and M. Sund. A generic 10 Gbps assembly edge node and testbed for frame switching networks. In *Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom2009)*, 2009. (Accepted for publication).

[4] J. Postel. TCP maximum segment size and related topics. RFC 879, IETF, November 1983.

[5] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S.C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *Network, IEEE*, 17(6):6–16, Nov.-Dec. 2003.

[6] Alteon Networks. Extended frame sizes for next generation Ethernets. Technical report, Alteon Networks, 1999.

[7] Chelsio Communications. Ethernet Jumbo Frames – the good, the bad, the ugly. Technical report, Chelsio Communications.

[8] ITU. Rec. X.200: Information technology – Open Systems Interconnection – Basic Reference Model: The basic model, 1994.

[9] J. Postel. Internet Protocol. RFC 791, IETF, September 1981.

[10] J. Postel. Transmission Control Protocol. RFC 793, IETF, September 1981.

[11] J. C. Mogul and S. E. Deering. Path MTU discovery. RFC 1191, IETF, November 1990.

[12] 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.

[13] D. Connolly and L. Masinter. The 'text/html' Media Type. RFC 2854, IETF, June 2000.

[14] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF, August 2008.

[15] N. C. Zakas, J. McPeak, and J. Fawcett. *Professional AJAX*. Wiley, 2006.

[16] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann. *The New Web: Characterizing AJAX Traffic*, volume 4979/2008 of *Lecture Notes in Computer Science*, pages 31–40. Springer Berlin / Heidelberg, 2008.

[17] J. W. Färber. *Modellierung von IP-basiertem Paketverkehr ausgewählter interaktiver Dienste*. Dissertation, University of Stuttgart, Stuttgart, 2007.