Universität Stuttgart

INSTITUT FÜR KOMMUNIKATIONSNETZE UND RECHNERSYSTEME
Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn

Interner Bericht / Internal Report **Nº 56**

Titel / Title

A Scalable Architecture for Flexible High-Speed Packet Classification

Verfasser / Author(s)

Simon Hauger, Sascha Junghans, Martin Köhn, Detlef Sass

Datum / Date 21.12.2006

Umfang / Size 9 Seiten / Pages

Quelle / Source

Schlüsselworte / Keywords Classification; Architecture; Broadband; High Speed; Hardware; Implementation; Programmable Logic; Reconfiguration; System Design; Table-Look-Up Method; MAN; WAN

Beitrag der Arbeit / Achievement

An architecture of a packet classification unit that is highly flexible and adaptable to new or changing protocol definitions and classification rules is presented and realization aspects and scalability issues of the proposed architecture are discussed.

Kurzfassung / Abstract

Inspection of a packet with respect to certain criteria and subsequent mapping of the packet to an according category is an essential task in any packet processing system. This is commonly referred to as packet classification. For such packet processing systems new or changing requirements are coming up continuously as networks and network services are rapidly evolving. So the architecture of a packet classification unit needs to be capable of processing packets at high speeds in the network nodes. Furthermore it needs to offer a high flexibility and adaptability with respect to new or changing protocol definitions and classification rules. Also, in order to meet future requirements the scalability of packet classification architectures is very important.
We present an architecture for hardware-based packet classification, that is highly adaptable to new or changing protocol definitions and classification rules. The architecture is easily scalable with respect to both processing speed and the number of classification rules. Further, we describe our realization of the proposed classification architecture integrated into an FPGA based 1 Gbps high precision network measurement system. Also, we show that our architecture is applicable for line rates of at least 10 Gbps on current FPGAs. Finally, a discussion of scalability issues of the proposed architecture and performance and resource utilization measures are presented.

A Scalable Architecture for Flexible High-Speed Packet Classification

Simon Hauger
Institute of Communication Networks and Computer Engineering
University of Stuttgart
Germany
simon.hauger@ikr.uni-stuttgart.de

Sascha Junghans
Institute of Communication Networks and Computer Engineering
University of Stuttgart
Germany
sascha.junghans@ikr.uni-stuttgart.de

Martin Köhn
Institute of Communication Networks and Computer Engineering
University of Stuttgart
Germany
martin.koehn@ikr.uni-stuttgart.de

Detlef Sass
Institute of Communication Networks and Computer Engineering
University of Stuttgart
Germany
detlef.sass@ikr.uni-stuttgart.de

ABSTRACT

Inspection of a packet with respect to certain criteria and subsequent mapping of the packet to an according category is an essential task in any packet processing system. This is commonly referred to as packet classification. For such packet processing systems new or changing requirements are coming up continuously as networks and network services are rapidly evolving. So the architecture of a packet classification unit needs to be capable of processing packets at high speeds in the network nodes. Furthermore it needs to offer a high flexibility and adaptability with respect to new or changing protocol definitions and classification rules. Also, in order to meet future requirements the scalability of packet classification architectures is very important.

In this paper we present an architecture for hardware-based packet classification, that is highly adaptable to new or changing protocol definitions and classification rules. The architecture is easily scalable with respect to both processing speed and the number of classification rules. Further, we describe our realization of the proposed classification architecture that we integrated into an FPGA based 1 Gbps high precision network measurement system. Also, we show that our architecture is applicable for line rates of at least 10 Gbps on current FPGAs. Finally, we thoroughly discuss scalability issues of the proposed architecture and present performance and resource utilization measures of FPGA-based realizations.

1. INTRODUCTION

Packet classification is the process of mapping packets to certain categories and is needed for most network processing systems. For this task, first for each packet the protocol stack has to be decoded and only then specific protocol fields can be evaluated by pattern matching based rules. Packet classification is one of the major tasks of modern packet processing systems imposing challenging requirements, especially at the edge of high-speed transport networks. Examples for packet classification are the assignment of packets to MPLS paths, flow identification, classification for QoS and scheduling purposes, observation of firewall rules and detection of intrusions.

The above application areas of packet classification span a wide range of the classification criteria regarding quantity and complexity. This ranges from classification according to few bits in a packet up to complex and dependent expressions of numerous criteria.

In modern packet processing systems, packet classification needs to be performed at full line speed with deterministic processing delays to avoid loss, additional delay jitter and additional packet buffers. These systems with link speeds up to 10 Gbps or more introduce hard constraints to classification speed and it therefore still remains an open networking problem [4].

Further, the lifetime of the node hardware—especially in core networks—is usually in the range of several years or even a decade. During this time, protocol definitions may change and new protocols are developed and introduced. For applications like intrusion detection and others that require deep packet inspection it is absolutely necessary to weave the changes into its rule set for an error-free operation. So, beside the high data rates to be processed, a high flexibility and easy adaptation of a packet classification module to new protocols and changing classification rules is an important requirement.

To achieve such a high flexibility and adaptability, the concept of microprogramming can be applied. Microprogramming is a concept of enabling the desired flexibility into hardwired logic circuits. It has been widely used in CISC microprocessors and is now being rediscovered for the use in network processing systems [11], [8].

In this paper we present an architecture for a packet classification module suitable for high speed packet processing systems which can automatically decode protocol stacks consisting of a configurable set of protocols. This architecture allows the flexible adaptation of classification rules by storing them in content addressable memories. Furthermore, by deploying the concept of microprogramming and storing this code in random access memories new network protocols can be implemented on the fly into the classifier. This usage of microprogramming together with content addressable memories lead to deterministic processing times. We also discuss major scalability aspects and show numerical results of exemplary fittings to current FPGAs. Finally, we show how the classification architecture has been applied to a high-performance measurement platform for network traffic [7].

The rest of the paper is structured as follows: Section 2 reviews the process of packet classification and suitable platforms for classification systems. Section 3 introduces our architecture for a microprogrammed classification module. In Section 4, the scalability of the architecture with respect to the number of classification rules as well as throughput are discussed. The realization aspects and implementation results are shown in Section 5. Finally, the paper closes with conclusions and an outlook to future work.

2. CLASSIFICATION MODULES

In literature several architectures as well as realizations for hardware-based packet classifiers have been proposed and discussed. In the following we will review these approaches with respect to general concepts and implementation aspects.

2.1 Classification Review

Packet classification is widely used in many different applications. Generally, packet classification inspects packets with respect to certain criteria and maps them to specific categories [2]. These categories can be interface identifiers for switching or routing decisions, classes of service for QoS support or packet filtering rules in a firewall or in an intrusion-detection system.

In different application fields, classification is often called differently. In the case of examining the layers two to four to make switching or routing decisions it is often called *lookup*. In the case of examining higher layer protocols the expression *deep packet classification* is used [6]. Very high line rates limit the number of classification rules, while systems with moderate line rates can support high numbers of classification rules.

The rules can be represented by multi-dimensional trees which must be traversed during the classification process or they can be ordered in a grid of tries. For the ordering hash values can be used with lookup tables during the identification process of the rule with the highest priority [4].

Other approaches separate the process of packet classification into two parts [10]: the assessment of single fields in the packets and the deviation of classifications results by the analysis of the assessment results. The first assessment is done on behalf of different *classification criteria* often called *single field matching*, the latter by the combination of several to *classification rules* which determine a *classification result*. As classification is not limited to certain protocol fields but could also be used by regarding arbitrary data in the payload of the packet, therefore we call the first assessment rules in a more general way *criteria* and not conditions for single field matching.

The classification criteria can be defined on behalf of different characteristics. They can refer to exact values like a specific port number or protocol type or they can cover ranges of values like target or source IP addresses for the identification of IP subnets or even more complex criteria. Some criteria check for values of certain bits in a distinct protocol header like the syn bit of TCP which need the introduction of "care" and "don't care" flags for each bit for an easy evaluation of parts of a packet field. This behavior can be described with so-called ternary logic and allows the efficient and compact definition of classification criteria. If a classification system can not cover ranges by a native comparison function, they can be covered by using ternary logic. A specific range must then be represented by several criteria.

The result of the single field matching phase can be represented by a large vector with a bit set for each matching criteria. This representation is often called *lucent bit vector (BV)* [5]. In [1] a compression for the processing on processor based platforms of the BV is introduced and called *aggregated bit vector*.

Many classification systems limit the classification on the analysis of the 5-tuple (source address – destination address – protocol type – source port – destination port). A flexible classification module should not be limited to currently existing protocols but should be extensible to newly introduced protocols. Communication protocol stacks have in common, that each protocol must have two fields indicating the type of its payload and the length of its protocol header or the protocol has static definitions for one or both fields. This allows the generalization of classification to any hierarchical data stream fulfilling these requirements. With this generalization a classification system needs not to be limited to specific protocol layers as long as this hierarchical data stream is fully contained in a single packet. For segmented data streams like the segmentation of IP payload, the classification engine would need a large state memory for continuing the classification on following packets.

2.2 Implementation Aspects

For the implementation of packet processing systems several basic methods and technologies can be used. General-purpose processors provide the flexibility for achieving highly adaptable systems but can not fulfill the high processing requirements and especially the high I/O bandwidth required in classification systems for high speed transport networks. Network processors (NPs) are devices which are designed for high I/O throughput and provide dedicated functions

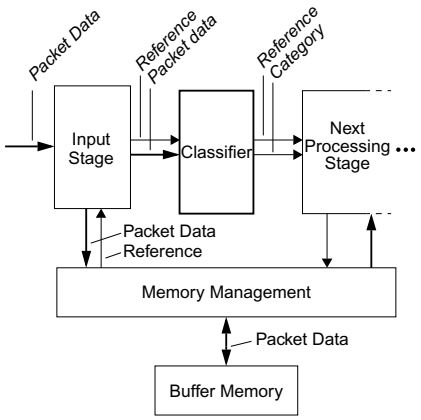

Figure 1: Typical environment for a classification module

for packet processing tasks. They can be programmed with specialized programming languages which allow the efficient usage of all NP components [2]. As NPs usually have a dedicated lookup engine as a monolithic block, the programmer can not influence the lookup process in detail.

Field programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs) allow more specialized logic designs for the processing modules. While FPGAs provide a very high flexibility because the logic design can be updated in the system, ASICs are faster and if the number of produced devices is high enough also cheaper. As the designer can develop system modules on different abstraction layers and make them very specific, both device types—FPGAs and ASICs—are very well suited to achieve the required performance. For the analysis of these BVs different methods depending on the used technology are known. Often, content addressable memories (CAM) sometimes with ternary logic (TCAM) are used in classification modules. In [9], BV analysis is realized on an FPGA-based platform using TCAMs.

Classification modules can process each packet in different ways. If the packet data is sent to the classifier as a data stream the classifier often scans the whole packet and identifies all matching classification criteria. Others receive a reference to the packet location in the buffer memory and read out the memory at those addresses which are important for their rules.

The constraints concerning processing delays for the scheduling modules are very strict for systems with high data rates as all packets must be classified at line speed. This is why we decided to design a system which does not handle references to buffer addresses which can lead to long memory access times. Instead, it scans a copy of the packets on the fly without time costly memory accesses.

Figure 1 shows a typical environment for a classification module with on the fly processing in a logic design.

The packet data enters the system via an input stage. This input stage forwards the data to the memory management

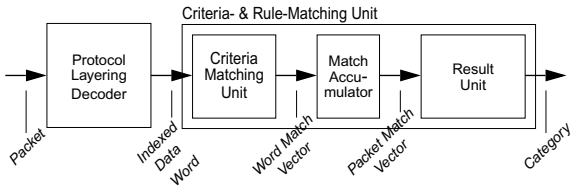

Figure 2: Basic structure of the packet classifier

to store the data into the buffer memory. The memory management returns a unique reference to the stored data. Upon this the input stage forwards a copy of the packet data as well as the corresponding reference to the classifier. The copied packet is scanned by the classifier. The classifier forwards the reference and the identified category and if needed part of the packet data to the next packet processing stage. Following stages, processing the determined category, use the reference to access the packet data in memory. This approach requires strict processing times for each packet but with these processing times, modules with high throughput can be assured.

3. CONCEPT AND ARCHITECTURE

In communication systems, the user data is encapsulated in several different protocols before it is transmitted. As packet classification is performed usually on matching packet header fields to certain values, a direct access to each protocol header and with this to the header fields is rather important. Protocols have either specified a fixed header length or a variable length, so the absolute position of a protocol field is not always fixed within a packet. Thus, it is almost not possible to perform the pattern matching without decapsulation layer by layer or at least decoding the boundaries of each protocol header.

Therefore, packet classifiers require an awareness of protocols and its layering within the packets. Many approaches in literature do not describe how they perform this task. For our classification module, we have realized such a functionality.

In the following sections we describe the concept, main architecture and the functionality of the units of our classification module. We present the most challenging block, the microprogrammable Protocol Layering Decoder, in more detail, which is responsible for the protocol awareness of our packet classifier. We show the high flexibility and adaptability to arbitrary new protocols or protocol extensions. Finally, we describe the Criteria-&Rule-Matching Unit and its realization aspects in order to achieve an efficient realization and in order to scale the number of classification rules and criteria.

3.1 Concept

In our packet classification module, the determination of the classification category is based on classification rules composed of one or multiple criteria, where each criteria corresponds to a single field matching expression. A criteria represents a searched pattern, mostly a protocol header field, at a specified position within a packet. This position is not an absolute position but relative to the start of each protocol.

This means, that our classifier is based on three main functionalities: protocol layering awareness, criteria matching and identification of the resulting category.

Considering a received packet at the classifier, the data in the packet can be distinguished into payload and protocol data. The protocol data represents the hierarchical encapsulation of the payload according to the protocol layers, by adding headers and/or trailers to the payload. The structure of the protocol headers are defined in the corresponding protocol specifications.

These protocol specifications have in common that each protocol must specify the length of its header as well as the protocol which is encapsulated in it. For both, either dedicated header fields are used or they are defined in the protocol specification. These two information are sufficient to decode all packet headers for the classification process. So, we consider a protocol header to be such a structural description and the rest of the packet, which can not be described in the above sense, to be the payload.

So, the protocol awareness provides the knowledge at which positions in a packet the different contained protocol headers are located. This enables the handling of protocols with variable header length as well as of tunneled protocols without any modification of the rule set. Once a protocol can be described according to the above definition of a protocol header, the module is able to detect arbitrarily ordered encapsulations of the known protocols within a packet.

In our classifier, the protocol fields are addressed relatively to the start of the protocol the field belongs to. This makes definition of criteria easy and compact. The packet data is processed word by word. All criteria are applied to each data word in parallel.

In parallel to that, all matched criteria are accumulated in a packet match vector, also known as the lucent bit vector, until the entire packet is processed. Based on this, the highest prioritized matching category is selected. Criteria can be shared by multiple rules.

3.2 Main Architecture

The main architecture of our classification module consists of two major blocks the Protocol Layering Decoder and the Criteria-&Rule-Matching Unit incorporating the above mentioned three functionalities. The first functionality, the protocol awareness, is contained in the first block and the latter two in the second block. These blocks consist of several functional units forming a pipeline structure, as depicted in Figure 2. Each packet is continuously processed in multiple bytes wide data words. We describe these functional blocks in the following.

The main task of the first unit, the *Protocol Layering Decoder*, is interpreting the relevant fields (e.g., header length field) of each protocol in order to label each data word with a distinct index. This index states the current protocol and the relative position of this data word within that protocol layer (e.g., word 2 within protocol UDP). So, this unit owns the knowledge about all the different protocol layers, incoming packets might consist of. It is completely micro-

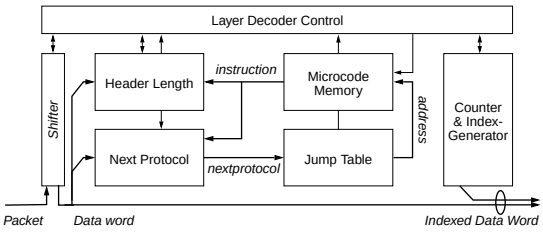

Figure 3: Basic structure of the Protocol Layering Decoder

programmable, so that the classification unit can easily be adapted to changes in protocol definitions or even new protocols. This unit is described in more detail in the next section.

The indexed data words are forwarded from the Protocol Layering Decoder to the *Criteria Matching Unit*. Here each data word is compared to a data set consisting of all criteria that are needed for the classification rules. Because of the appended index of each word only those criteria that are relevant for the current protocol and the current position within that protocol can lead to a match. The result of this comparison is output in the form of a word match vector, consisting of zeros at those positions where no match was found and ones at those positions where a match was found at the according positions in the data set.

The word match vectors of each data word of a packet are combined in the *Match Accumulator* unit. This unit basically combines the word match vector of each data word with all previous match vectors of this packet by a bit-by-bit logical OR. When all words of a packet have been processed, the packet match vector shows the set of all criteria this packet complies with.

The *Result Unit* uses this packet match vector to come to a decision and assigns the respective category to the processed packet. This is done by comparing the fulfilled criteria with all classification rules, and classifying the packet according to the classification rule with the highest priority that is adhered to.

3.3 Protocol Layering Decoder

In order to achieve the required flexibility to current and future protocols our classifier contains a microprogrammable Protocol Layering Decoder.

The Decoder utilizes the fact that all protocols must contain a field specifying the protocol contained in its payload, unless this is statically defined. Further, all protocols have either a fixed header length or contain a field in their protocol header defining their header length.

The structure of the Protocol Layering Decoder is depicted in Figure 3. The *Microcode Memory* contains microcoded instructions for all supported protocols. The microcoded instructions contain information about the position of the header length field or a fixed length as well as the position of the field specifying the next protocol.

These instructions are interpreted by two functional units aptly named *Header Length* unit and *Next Protocol* unit. These units extract, process and buffer the respective fields of the incoming packet data words. The header length is used to determine when the microcoded instructions for the following protocol have to be loaded. Also, it is used to compute the shift value for the *Shifter* which segments and aligns the incoming packet such that each protocol header starts in a new data word. This reduces the complexity in following functional units.

The extracted header field, containing the information about the next protocol layer, is forwarded to a *Jump Table*, where the address of the corresponding microcode instruction in the Microcode Memory is determined.

Finally, the *Counter- & Index Generator* keeps track of the position of the current data word within the current protocol. This information is used in the Header Length and Next Protocol units in order to extract the correct byte fields of the protocol header. Furthermore the Counter- & Index Generator uses this information to label each data word with an index stating the protocol this data word is part of and the word position within this protocol. This index is used in the following Criteria Matching Unit as described in the previous section. The *Layer Decoder Control* consists of a finite state machine controlling the cooperation of all units.

As the Microcode Memory as well as the Jump Table are completely user-programmable this Protocol Layering Decoder is adaptable to arbitrary new or changed protocol definitions. There are no constraints about the structure of the protocol headers except for mandatory fields specifying the following protocol in the encapsulation of the packet and the current header length unless those are fixed values.

When realizing this decoder on an FPGA or ASIC, the Jump Table can be implementing using binary content-addressable memory (CAM), looking for exact matches when looking for the address for the next protocol instructions.

3.4 Criteria-&Rule-Matching Unit

The Criteria-&Rule-Matching Unit is composed of the Criteria Matching Unit, the Match Accumulator and the Result Unit. Both the Criteria Matching Unit and the Result Unit mainly consist of a CAM, called Criteria CAM and Rule CAM, respectively. These two units use ternary CAMs (TCAMs) that allow the usage of a third value 'X' meaning "don't care" besides the logical '1' and '0'. With this third state irrelevant bit fields within a data word can be masked out. This is an efficient way to check for different criteria or to find a final category in those units. The Match Accumulator is mainly a large register which is called Accumulator Buffer. The principle realization of the Criteria-&Rule-Matching Unit is depicted in Figure 4.

The Criteria-&Rule-Matching Unit gets the indexed data word of a packet from the Protocol Layering Decoder. This is applied to the Criteria CAM, the entries of which consist of the index and the actual check pattern. It produces the word match vector indicating which CAM entries match to this. Then, the word match vector is combined by a bit-

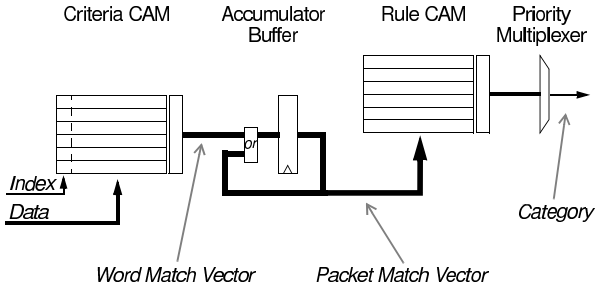

Figure 4: Principle realization of the Criteria-&Rule-Matching Unit

by-bit logical OR with all previous matches stored in the Accumulator Buffer.

When all words of a packet have been processed the Accumulator Buffer shows the set of all criteria this packet complies with. This is used to come to a category by applying it to the Rule CAM, showing all matched rules. Each entry in the Rule CAM corresponds to one rule and is the required accumulated packet match vector to be fulfilled for this rule. The priority multiplexer selects from all matched rules that rule with the highest priority.

In order to scale the classifier for high data rates and large numbers of criteria and rules, a critical aspect is the width of the CAMs. However with increasing the width of a CAM, its maximum possible operating frequency decreases.

A simple method to circumvent this problem of too wide CAMs is to split the wide CAM into several less wide CAMs. Each smaller CAM checks its part of the input vector with its corresponding part of the original CAM data set. After that, each result bit of each row of the smaller CAM is then combined by a logical AND with the corresponding output bits of all other CAMs. This method is also described in [11] and can be applied when scaling the Criteria CAM and the Rule CAM to support wider input vectors.

But often criteria check only few bits in a data word and do not depend on long bit strings. So large parts of the rows in the criteria CAMs are mostly filled with "Don't Care" bits. If in this case the CAM is split into several smaller CAMs with a following AND stage, often entire rows are filled with "Don't Care" bits and can thus be avoided.

This is exploited in the concept shown in Figure 5. Each check pattern entry in the Criteria CAM, i.e., the CAM entry without the index, is segmented into smaller sub-words and distributed to the smaller CAMs. The entries of these smaller CAMs consist then again of replicated index and the sub-word of the check pattern. Now, the succeeding AND stage is omitted. The same is applied to the processing data word which is also segmented into sub-words and the replicated index is pre-pended. These new small data words are fed to the smaller CAMs in the order as the data word is segmented into sub-words.

This approach has multiple advantages. The size of each check pattern in the smaller CAMs remains still small, en-

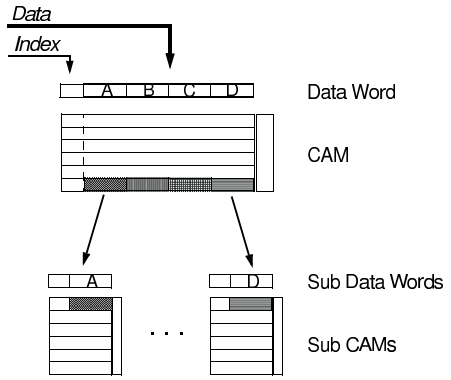

Figure 5: Reduction of Criteria CAM size

abling a fine granular definition of the criteria, which can be reused by other rules. Also, the overall resource consumption is reduced, because the sum of all smaller criteria entries is smaller than the sum of the large CAM entries due to the more specific definition possibilities by the smaller criteria.

4. SCALABILITY

Network data rates will continue to increase so the processing rates in network nodes have to speed up in the future as well. Consequently it is important to examine the presented classification architecture with respect to its capability to support higher data rate processing. This will be done in the last section of this chapter.

Also, more and more network services emerge that have to be provided at line rate in the network nodes. This means more rules to check within classification units and consequently also more criteria to check in the Criteria Matching Unit. The scalability of our classification architecture concerning these issues is discussed in the following two sections.

4.1 Number of Rules

Increasing the number of rules N the classifier can support, only affects the Result Unit of the proposed architecture. Here all three logical stages, namely the Rule-CAMs, the logical AND gates and finally the priority multiplexer have to be regarded.

The Rule CAMs output a bit vector indicating which rule is fulfilled given the criteria looked at in this CAM. The depth of the Rule CAMs corresponds to the number of supported rules N. Thus with increasing the number of rules also the CAMs become deeper. In order not to get too deep CAMs, possibly negatively affecting the routing capability on an FPGA and thus the classifier's throughput, each CAM can be divided into several smaller CAMs each only generating a bit vector for a subset of all rules. So this stage is scalable without deteriorating latency and throughput, and the resource utilization is linearly proportional to the number of rules.

In the next logical stage, each logical AND conjunction combines for each rule the corresponding output bits of the preceding Rule CAMs. When increasing the number of supported rules N within the architecture, only the number of such parallel AND gates increases linearly with the number of rules. Thus throughput and latency of this stage are independent of the number of rules, too, and the resource utilization scales with $O(N)$.

The terminal priority multiplexer selects of all matching rules that rule with the highest priority. When scaling up the number of possible rules checked within the classification architecture, this multiplexer also has to become larger. Given N rules the number of its input ports scales with $O(N)$, increasing its resource utilization also only linearly. In order to achieve a high throughput several priority multiplexers can be cascaded with a register at each output, thus forming a pipelined prioritizing multiplexer. By that the number of supported rules of the packet classification architecture is easily scalable without sacrificing throughput and only slightly increasing the latency of the system by introducing a pipelined priority multiplexer. The resource utilization increases linearly with the number of rules.

The scalability statements above, obviously do not consider additional minor routing delays or additional minor resource increases due to physical constraints on an FPGA or ASIC.

4.2 Number of Criteria

The number of supported single field criteria K within the classification system not only affects the Criteria Matching Unit. But, as with the number of criteria also the width of the input word match vector enlarges, also the Accumulator and the Result Unit have to be modified. More specifically, the depth of the Criteria CAMs, the width of the Accumulator Buffer, and the width of the Rule CAMs depend on the number of supported criteria.

Firstly, the number of criteria is commensurate to the depth of all Criteria CAMs together. To support a higher number of single field criteria either the depth of each Criteria CAM can be increased proportionally, or—as with the Rule CAMs before—they can be subdivided into several parallel Criteria CAMs each checking only for a subset of the criteria corresponding to its input. Again, the resource utilization scales with $O(K)$, neither affecting throughput nor latency of the system.

Also, the accumulating buffer size is linearly proportional to the overall number of criteria, thus only linearly affecting the resources.

With an increased number of criteria K, the Rule CAMs also need to operate on wider criteria vectors. However increasing the width of the CAMs, increases their combinatorial delay and thus the maximum throughput. So it is better to use pipelined CAMs or, more easily, several CAMs that work on smaller subsets of the criteria, and the subsequent AND stage combines the results for each rule, possibly in a pipelined fashion. By this the throughput is not affected by the number of criteria and the system latency is only increased by the depth of the AND conjunction pipeline. The resource utilization scales linearly $O(K)$ with the number of criteria K.

Again, additional minor routing delays or resource increases due to physical constraints are not regarded here.

4.3 Throughput

The throughput of a packet processing system can be characterized by the data rate, measured in bits per second, and the packet rate, measured in packets per second.

Internally the data rate of the classification module is limited by the clock rate f_{clk} multiplied by the word width W. The Criteria-&Rule-Matching Unit does not change this data rate due to its pipelined structure processing one data word each clock cycle. The Protocol Layering Decoder sometimes delays the processing of data, however. Each time a boundary between two adjacent protocol headers lies within a data word, the Protocol Layering Decoder shifts the data back to such an extent that the new header starts aligned to the next word boundary. The Ethernet header, for example, has a length of 14 Bytes, so with a word width of 16 Bytes the data word containing the Ethernet header also contains 2 bytes of the next protocol layer, e.g. of the Internet Protocol (IP). In the next clock cycle, the Layering Decoder shifts the data back by two bytes, so that the next header, here the IP header, begins aligned with the beginning of the next data word.

The maximum internal data rate D_{int} of the classification module therefore computes to the maximum internal data rate $f_{\text{clk}} \cdot W$ multiplied by the quotient of the volumes of input data and output data of the Protocol Layering Decoder:

$$D_{\text{int}} = f_{\text{clk}} \cdot W \cdot \frac{\sum_{i=1}^{N_{\text{hdr}}} H_i + R}{\sum_{i=1}^{N_{\text{hdr}}} \left\lceil \frac{H_i}{W} \right\rceil W + \left\lceil \frac{R}{W} \right\rceil W}$$

Here, the input data volume is the sum of the sizes H_i of all N_{hdr} protocol headers and the size of the remaining packet R. The output data volume is larger, as each protocol header size is rounded up to the next multiple of the word size. The same applies for the remaining packet size.

When computing the external data rate that can be applied to the classifier, i.e. the external line rate, also the minimum guard time between succeeding packets (e.g., the Ethernet inter-framing gap) has to be considered. The external data rate therefore increases by the factor $\frac{\sum_{i=1}^{N_{\text{hdr}}} H_i + R + G}{\sum_{i=1}^{N_{\text{hdr}}} H_i + R}$, here, G is the number of bytes equivalent to the minimum guard time between two packets (e.g. for Ethernet links $G = 12$ bytes). So the external maximum data rate D_{ext} applicable to the classifier is:

$$D_{\text{ext}} = f_{\text{clk}} \cdot W \cdot \frac{\sum_{i=1}^{N_{\text{hdr}}} H_i + R + G}{\sum_{i=1}^{N_{\text{hdr}}} \left\lceil \frac{H_i}{W} \right\rceil W + \left\lceil \frac{R}{W} \right\rceil W}$$

Both, the internal and the external data rate increase if the Protocol Layering Decoder is modified in a way that only the protocol headers are processed, starting with the next packet as soon as all relevant headers of the current packet are processed. With this modification of the Protocol Layering Decoder the external throughput computes to:

$$D_{\text{ext,mod}} = f_{\text{clk}} \cdot W \cdot \frac{\sum_{i=1}^{N_{\text{hdr}}} H_i + R + G}{\sum_{i=1}^{N_{\text{hdr}}} \left\lceil \frac{H_i}{W} \right\rceil W}$$

The maximum packet rate P_{ext} of the classification module is the data rate divided by the size of the regarded packets (including guard time):

$$P_{\text{ext}} = \frac{f_{\text{clk}}}{\sum_{i=1}^{N_{\text{hdr}}} \left\lceil \frac{H_i}{W} \right\rceil + \left\lceil \frac{R}{W} \right\rceil}$$

With the formulas above it can be seen, that in order to increase the throughput of the classification module two approaches or a combination of them can be considered. One approach is to increase the clock rate f_{clk} of the system. To be able to do this either an ASIC solution or future high performance FPGAs have to be used. The other approach to speed up the throughput of the system is to increase the word width W and thus the number of bytes processed in each clock cycle. This affects both the Protocol Layering Decoder and the Criteria-&Rule-Matching Unit of the classifier.

However, increasing the word width of the Layer Decoder to much more than 16 Bytes is not reasonable, as this also increases the overhead of redundant bytes at the end of each protocol layer and thus also negatively affects the throughput. To circumvent this drawback, the Layer Decoder could be changed in a way that it can process more than one protocol within one data word and thus within one clock period. However this would dramatically increase the complexity of this unit, thus pushing up the resource utilization and decreasing the maximum possible clock frequency.

Another way to increase the number of bytes processed in each clock cycle, might be to replicate the entire Layer Decoder Unit several times and to distribute the incoming packets between those units. This would entail the need of buffers in front of and after the Protocol Layering Decoder, that have to be carefully dimensioned. Also packet ordering had to be considered in this case.

The back part of the presented architecture, the Criteria-&Rule-Matching Unit can be scaled to a higher throughput per clock cycle in both of the above mentioned ways. The word width of the parallel CAMs might be increased or the complete classification part might be replicated as the layer decoder above. By increasing the word width of the parallel CAMs, even more CAMs have to work in parallel, thus not negatively affecting their possible maximum clock rate.

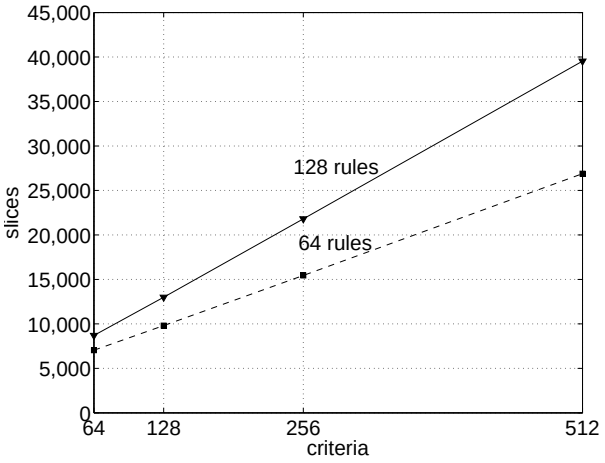

Figure 6: Resource utilization with different numbers of criteria and 64 rules (dashed) and 128 rules (solid)

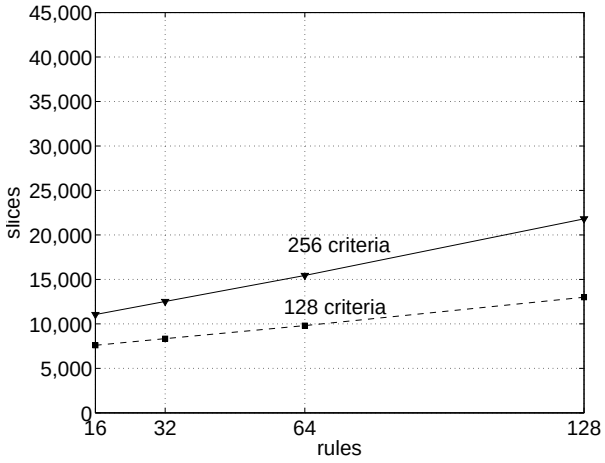

Figure 7: Resource utilization with different numbers of rules and 128 criteria (dashed) and 256 criteria (solid)

5. APPLICATION ON FPGA

The presented classification module has been implemented in VHDL and synthesized for configuration of FPGAs. We developed the VHDL design as generic as possible in order to easily implement differently scaled classification modules. Further, this allows us to synthesize the design for both Altera and Xilinx FPGAs.

In this section, we first present results of case studies in order to quantify the resource consumption of our classifier when used as a module in an FPGA as well as to validate the complexity estimations presented in the last section. We consider scenarios with different numbers of criteria and different numbers of rules. Then, we show the successful application of the classification module in our high-performance Internet measurement platform which uses the classifier for filtering purposes.

5.1 Implementation Results

In this section, we present results of numerical case studies. For this, we synthesized, mapped, placed and routed a multitude of variants of our implemented classifier module for the configuration of a Xilinx Virtex IV FPGA.

In order to achieve a high throughput the protocol layering decoder unit works with a word width of 128 bits in all tested classification modules. The 128 bit wide output is separated into four 32 bit wide sub-words that are processed in parallel in the subsequent criteria CAMs.

We vary the number of criteria between 64 and 512. This is realized by increasing the number of CAM entries checked in parallel in the four criteria CAMs. The accumulated match bit vector with a width equal to the number of checked criteria is then further processed by several rule CAMs followed by the AND stage and a priority multiplexer. Here, we use a range of 16 to 256 for the number of possible rules of the test-wise realized classifiers.

The different classification unit variants were synthesized for a XILINX Virtex IV FX100-11 and LX200-11 using Precision Synthesis from Mentor Graphics for synthesizing the net lists and the ISE software from Xilinx for the Place and Route. While for almost all configurations the results are equivalent for both FPGAs, the large classifiers with 512 criteria and 256 rules only fit into the LX200-11 variant of Virtex-IV.

Figure 6 depicts the resource utilization versus the number of criteria ranging from 64 up to 512. We plot the results for classification units with 64 and 128 rules. The number of occupied "slices" (basic logical elements) within the FPGA is increasing almost linearly with the number of criteria. This supports the predication of linear scalability of our classifier with respect to the number of criteria as stated in chapter 4.

In Figure 7, we show the resource utilization versus the number of rules ranging between 16 and 128 rules. We plot the results for classification units supporting 128 or 256 criteria. Again, the linear scalability of the classification architecture referring to the number of supported rules, as claimed in chapter 4, is demonstrated.

Furthermore, our synthesis experiments have also shown that currently the longest combinatorial path within the classification unit which limits the maximum clock frequency lies always within the protocol layering decoder. As this unit is independent of the number of rules and criteria, all realized classification units have similar performance results with respect to throughput. All mentioned variants support a clock rate of approximately 60 MHz. For a word width of 128 bits this corresponds to a data rate of 7.68 Gbps after the protocol layer decoder. For a mean packet length of 200 Byte including Ethernet, IP and TCP Headers as well as considering the Ethernet inter-framing gap, a throughput of approximately 20 Gbps can be achieved.

5.2 Prototype

The classification module is successfully applied in a high-performance Internet measurement platform, the I^{2}MP [7]. The measurement platform is built up on the UHP (Universal Hardware Platform) [3] of the authors' institute. This platform provides a rather old FPGA from Altera (APEX20K 400CB652-C7) and several communication interfaces, like electrical and optical Gigabit Ethernet interfaces.

This measurement platform has been designed to record selected areas of the packet headers. As this selection must be done based on the content of the packet, we first classify all incoming packets using this classification module. Despite the rather old FPGA technology used for this application, it is still possible to perform all packet processing tasks, including the classification, at full Gigabit Ethernet line speed.

Within the measurement platform, the classification module is operated at a clock frequency of 42 MHz and a word width of 32 bits. With this clock frequency classification can be performed of up to 2.2 million minimum-sized Ethernet packets in one second. This performance is achieved with only using few resources even on this outdated FPGA. A classification module with 32 classification rules using a total of up to 64 criteria only uses 10% of the logic elements and 5% of the available memory bits of the used FPGA.

Unfortunately modern Altera FPGAs like Stratix II and Cyclone II do not support the effective realization of TCAMs within the FPGA, so higher throughput classifiers can not be implemented on Altera FPGAs.

6. CONCLUSIONS

In this paper we presented a scalable architecture for a high speed classification module. It enables a network node to scan data packets on the fly with very low latency at line speed. With the microprogrammable Protocol Layering Decoder, the module is easily adaptable to new protocols and services on any layer during operation. This flexible classifier was realized on an FPGA-based platform as part of a traffic measurement system providing important traces for traffic characterization. The module needs only few FPGA resources without critical timing performance in the Gigabit Ethernet environment. Also, case studies have shown that the architecture is scalable up to data rates of 10 Gbps with only minor modifications.

In our future work selected modules will be doubled and operated in parallel in order to increase the data rate further. Also, an additional scan module shall be designed which allows the identification of signatures at arbitrary positions in the packets. This module will support the easy identification of higher layer services like peer to peer traffic or typical patterns for intrusion-detection systems.

7. ACKNOWLEDGMENTS

The authors would like to thank Damir Ferenci and Arthur Mutter for their contributions to the development, implementation and test of the I^{2}MP measurement platform containing the classification module.

8. REFERENCES

[1] F. Baboescu and G. Varghese. Scalable packet classification. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 199–210, New York, NY, USA, 2001. ACM Press.

[2] D. E. Comer. *Network Systems Design using Network Processors*. Prentice Hall International, 1. edition, 2006.

[3] Institute of Communication Networks and Computer Engineering. The Universal Hardware Platform (UHP), 2005. `http://www.ikr.uni-stuttgart.de/Content/UHP/`.

[4] M. Kounavis, A. Kumar, R. Yavatkar, and H. Vin. Line rate packet classification and scheduling. In *Proceedings of the Tutorial at Symposion on Architectures for Networking and Communications Systems (ANCS)*, Princeton, New Jersey, USA, Oct. 2005.

[5] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 203–214, New York, NY, USA, 1998. ACM Press.

[6] P. C. Lekkas. *Network Processors: Architectures, Protocols and Platforms*. McGraw-Hill, 2003.

[7] D. Sass and S. Junghans. I2MP - an architecture for hardware supported high-precision traffic measurement. In *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB)*, Nrnberg, Mar. 2006.

[8] C. Semeria. Implementing a flexible hardware-based router for the new ip infrastructure. JuniperNetworks Inc., White Paper, August 2005.

[9] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245, New York, NY, USA, 2005. ACM Press.

[10] J. van Lunteren and T. Engbersen. Fast and scalable packet classification. *Selected Areas in Communications, IEEE Journal on*, 21(4):560–571, 2003.

[11] S. Vassiliadis, S. Wong, and S. Cotofana. Microcode processing: Positioning and directions. *Micro, IEEE*, 23(4):21–30, 2003.