FPGA-Assisted DPI Systems: 100 Gbit/s and Beyond

Péter Orosz, *Member, IEEE*, Tamás Tóthfalusi, and Pál Varga, *Member, IEEE*

***Abstract*—Carrying out Deep Packet Inspection (DPI) in aggregated network connections remains a continuous requirement even though the line rate reaches and exceeds 100 Gbit/s. The increasing packet-arrival rate necessitates efficient solutions for on-the-fly packet parsing, packet classification, and distribution for parallelized, software-based payload inspection.**

Inspection complexity and real-time processing are competing requirements. The deep analysis capabilities of software-based approaches can be enhanced by hardware-based support on time-critical packet parsing and classification. Moreover, some payload inspection tasks can be carried out in hardware as well, further reducing the resources spent on software-based solutions.

This paper aims at presenting the state-of-the-art and describing a set of best practices in FPGA-based packet processing, which can be applied for DPI-related tasks at 100 Gbit/s and beyond. Accordingly, we provide an architectural view of the DPI systems throughout the paper.

Besides summarizing the limitations of hardware- and software-based solutions for the three processing phases within a DPI system (packet parsing, packet classification and payload inspection), this paper reveals the possible trade-offs for choosing the different technical approaches. These limitations include operating frequency, bus size, available memory, on-chip physical resources for hardware-based implementations, and CPU time for software-based solutions.

***Index Terms*—Deep Packet Inspection (DPI), packet parsing, packet classification, payload inspection, FPGA, 100 Gbit/s.**

I. Introduction

DEEP Packet Inspection (DPI) is a network traffic analysis method that is used for various purposes such as traffic classification, intrusion detection, virus and spam filtering, protocol misbehavior detection, or resource management. Accordingly, the application fields of DPI are network monitoring and management, Intrusion Detection Systems (IDS), on-the-fly Quality of Service evaluation, traffic engineering, and many more [1][2][3][4]. Within a DPI system (from an architectural viewpoint), the analysis process can be divided up to three phases, which are evolved into substantive research areas in the recent years: *packet parsing*, *packet classification* and *payload inspection,* respectively. Throughout this paper we assign the payload inspection process (i.e., the most significant part of the DPI system) to the transport protocol payload, focusing on the protocol headers of the upper layers (i.e., session, presentation and application layers).

Péter Orosz, Tamás Tóthfalusi and Pál Varga are with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economincs, 1117 Budapest, Hungary
(email: orosz.peter, tothfalusi.tamas, pvarga@tmit.bme.hu)

Since on-line DPI requires large amount of processing power in order to analyze high volume network traffic in real time, network bandwidth beyond 10 Gbit/s brings novel challenges for software-based packet inspection solutions [3]. In a 100 Gbit/s core network, for example, packets should be processed on-line at a rate of 1.5×10^8 packets per second, in the worst case.

The requirement for hardware assistance of any DPI phase is therefore emerging with the evolution of the core networks. Accordingly, some reasonable questions arise: i) How hardware acceleration can assist the different phases of the deep packet inspection process? ii) What are the benefits and drawbacks implementing some parts of the packet processing in hardware? iii) What are the hardware constraints for accelerating the payload inspection phase?

Getting advanced Field Programmable Gate Arrays (FPGAs) [5] Network Processors (NPs) [6], or Graphics Processing Units (GPUs) [7] involved in the DPI process opens the way for resource intensive tasks to be accelerated and therefore enables software-based packet inspection to step up in the performance scale. Although NPs [8] and GPUs [9] are capable to speed up some well-defined tasks of the DPI process, we are focusing on the acceleration features of the FPGA technology. Due to its hardware-level reconfiguration property, an FPGA can adapt to the specific requirements of the traffic inspection process. When real-time operation is a requirement (e.g., in an intrusion detection and prevention system), a further advantage of the FPGA technology is the ability to build an implementation that processes incoming traffic with non-varying latency.

In this survey paper, we investigate the FPGA acceleration of the DPI phases and present the state-of-the-art considering 100 Gbit/s networking and beyond. We demonstrate the benefits and the drawbacks of stepping towards FPGA-based acceleration for each DPI phase. We argue that the hybrid hardware- and software-based architecture can eliminate the capacity bottleneck of the entirely software-based DPI solutions. In contrast to previous DPI related surveys [1][10], our aim is to consider the entire DPI process – from a system architectural perspective – as a whole, including all of the three consecutive phases, i.e., *packet parsing, packet classification, and payload inspection*. We pay special attention to challenges arising at state-of-the-art, high-speed networks.

Nevertheless, we also intend to show some interesting alternative research proposals based on the GPU technology.

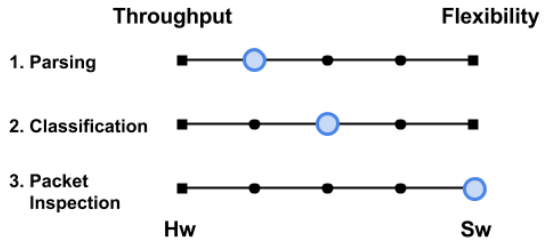


Figure 1. Design trade-off at the phases of the DPI process

High performance packet parsing and packet classification are separated research areas. Therefore, related works often present methods and solutions focusing only to a well-defined task of the DPI process without taking into consideration the interaction between phases. Instead, we show tradeoffs between FPGA-based hardware acceleration and CPU-based processing, throughout the three abovementioned phases of the inspection process. Moreover, we present hardware related design factors and best practices for both 100 and 400 Gbit/s operations.

In order to distinguish aims and actual depth of processing, let us cover the DPI problem domain with three operational phases (see Figure 1). This divisioning scheme of the DPI process as well as the structure of our survey corresponds to the individual research areas as well as a system architectural viewpoint. First, packet parsing is required to identify the different fields of the protocol messages, as well as to reveal protocol multi-encapsulation (e.g., IP-in-IP, Ethernet over MPLS). Second, packet classification covers the analysis of well-defined protocol fields, according to a set of filtering rules. Third, the ultimate DPI-challenge is payload inspection. Based on the result of the online inspection, each individual TCP/UDP flow can be assigned to an application class or to a dedicated application. The primary question related to all DPI phases is the design trade-off between throughput and flexibility (see Figure 1), which determines the requirement for hardware acceleration to achieve high throughput.

In the payload inspection DPI phase, we have to discuss two major methods that are newly used for various purposes in the DPI domain. These are bloom filters and neural networks. This survey on hardware-assisted DPI methods could not be considered complete without them.

The remainder of the paper is organized as follows. In Section II we define the major DPI-related terms and methods. Section III introduces packet parsing, the first phase of the DPI process. We review the basic operational concepts of the FPGA-based packet parsing methods as well as the design trade-offs between throughput and protocol flexibility. Packet classification methods and implementational issues will be discussed in Section IV. We compare existing solutions and show the state-of-the-art in both packet parsing and packet classification. The third DPI phase, i.e., payload inspection will be overviewed in Section V. We also reviewed scientific works for two common methods related to various tasks within the inspection process: bloom filters and neural networks. Finally, Section VI concludes the paper. While we present related works for the consecutive phases of the DPI processing chain, we also aim to show best practices as well as open issues in terms of performance and flexibility. In order to put greater focus on the open challenges, we summarize "lessons learned" at the end of each section, beside the final conclusions of the paper.

The general overview of the paper is the following:

- Definitions
  - Traffic Classification
  - Statistical Classification of the Traffic
  - Deep Packet Inspection
  - DPI Process
- Phase I – Packet Parsing
  - Limitation of Software-based Parsing: Time
  - Benefits of a Hybrid HW/SW Approach
  - A Survey on HW-based Packet Parsing
  - Lessons Learned
- Phase II – Packet Classification
  - Limitation of Software-based Classification
  - Benefits of a Hybrid HW/SW Approach
  - A Survey on HW-based Packet Classification
  - Lessons Learned
- Phase III – Payload Inspection
  - Challenges and Solutions for on-the-fly payload Inspection
  - Why decoding Higher Layers in not easy in Hardware?
  - Hardware-assisted Payload Inspection
  - Bloom Filters in a DPI system
  - Neural Networks in a DPI system
  - Lessons Learned

II. Definitions

1. Traffic Classification

The automated process of categorizing computer network traffic according to various parameters into a number of traffic classes is called traffic classification. The two main uses of traffic classification are related to security [11] and quality. The security-related uses support the fight against malicious attacks or intrusions related to network segments, endpoints or users. The quality-related uses aim to provide quality guarantees to different applications, since their traffic have different QoS criteria when traversing the network.

a. Flow-based Classification Methods

The input of the traffic classification method can be the mere packets, or some digested information about the series of packets related to the traffic between the source and the destination. The latter is called flow information, which is identified by some basic attributes, such as the 5-tuple: source and destination IP addresses, transport protocol ports (source and destination), and the type of the transport protocol. The flow information can be provided by any active networking node (e.g., switch or router), as well as passive monitoring equipment.

b. Packet-based Classification Methods

The traffic classification method is called packet-based when the input of the traffic classification are the pure packets, or part of the packets (e.g., some headers).

2. Statistical Classification of the Traffic

There are statistical methods of traffic classification that may take the flows or the packets as input. Instead of checking whether the input information is matching some rule (e.g., 5-tuple or payload fingerprint), statistical methods use other, seemingly hidden information for traffic classification. This information comes from statistical properties of the series of packets and flows. These properties include packet timings (e.g., interarrival time, gap time, or even round-trip delay), packet length, flow timing (e.g., duration, interarrival jitter within the flow), or flow length (e.g., TCP segment size), among others.

3. Deep Packet Inspection

In order to make a decision on a packet under analysis (e.g., for traffic classification or filtering purposes), the packet header or the payload (or both) can also go under investigation. As their name suggest, Deep Packet Inspection methods go deeper than the packet header, and inspect the payload. Some DPI methods merely focus on the payload, and others analyze the headers and the payload, as well. In this paper, we follow this latter, wider meaning of DPI. Furthermore, while some sources state that DPI is used for detecting or preventing intrusions, we suggest to avoid narrowing its meaning this far.

a. Software-based DPI

Software-based DPI methods take the packets and analyze them merely through software, run on CPU-based algorithms.

b. Hardware-assisted DPI

For pre-processing, software-based methods can be assisted by algorithms optimized for hardware. The assisting hardware can be anything but a CPU: either FPGA, GPU or Network Processor.

4. DPI process

a. Packet Parsing

The aim of packet parsing is to identify the various protocol headers encapsulated in the packet header, and extract meaningful information from them. These methods recognize the protocol header structure, multiple encapsulations, and are able handle the so-called shim header embedding.

b. Packet Classification and Packet Filtering

Packet classification and packet filtering are similar terms, although the first is predominantly used in network management (specifically in routing and switching), whereas the latter is mostly used in security management. The algorithms used in packet classification aim for an output of multiple choices, e.g., which output port(s) should we direct the packet to. On the other hand, the algorithms of packet filtering have a binary choice output: is that packet interesting (for further processing), or not. Another decision can be whether to drop the packet or not.

c. Payload Inspection

The deepest analysis of the DPI process happens during payload inspection. Depending on the desired precision of the output, the complexity of the inspection can range from looking for a typical pattern in independent packets to the stateful analysis of messages, considering protocol-specific dynamics, and payload signatures, as well.

III. First Phase: Packet Parsing

The first phase of DPI process parses the protocol headers up to the transport layer for each incoming packet. Packet parsing methods recognize the protocol header structure, and also handle the shim header embedding (i.e., MPLS labels). Typically, the Ethertype field or the standard 5-tuple, which is often a common part of a packet classification filter rule, is extracted at this phase [12]. Accordingly, the main output of the parsing process is a pre-defined subset of header fields, so called *n*-tuple or protocol metadata.

The parser engine operates on a parse graph, which is a predefined aggregation of the valid header structures and defines state transitions and an internal control structure. A typical example of the parse graph is presented in Figure 2. The parse graph determines the possible steps of the packet analyzer process. Today's state-of-the-art parsing engines [12][13] handle a large scale of encapsulation structures (i.e., implement a complex parse graph) and extract the appropriate protocol fields (*n*-tuples) from multi-encapsulated packets that are commonly traversing core carrier networks.

The packet parsing process requires up-to-date knowledge of the protocols appearing in the network messages. This does not only mean standardized protocol parameters, but current proprietary implementations, and protocol dynamics, as well. These parameters change relatively rarely, but when they do, it should be noticed, and relevant modifications should be applied in the DPI algorithms.

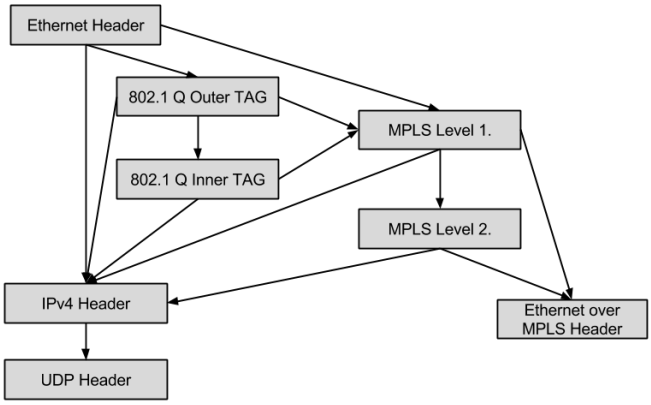


Figure 2. An example of a possible parse graph

Adaptation of the parser engine to new protocols is a critical requirement in today's networks. Therefore, updating or

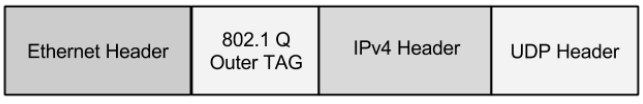
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/COMST.2018.2876196, IEEE Communications Surveys & Tutorials

reconfiguring the parse graph should be a regular task for network operators when a new service appears in the network. Based on the method of reconfiguration, parsing engines commonly implement one of the following design principles:

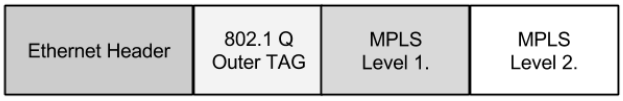
a) Parse graph and extracted n-tuple are predefined in compilation time,
b) Parse graph and extracted n-tuple are reconfigurable in run time with service interruption,
c) Parse graph and extracted n-tuple are reconfigurable in run time without service interruption.

In this survey, we do not focus on the naive approach that merely handles a fixed header structure and without allowing alternative cases, i.e., the parse graph is actually a single path. A predefined parser (a) may support alternative header structures while it is still a closed-design, in which header field extraction always results in a pre-defined n-tuple. The fields of the n-tuple are fixed at compilation time. In contrast, a reconfigurable parser (b), (c) has a flexible architecture that enables *add*, *delete* and *modify* operations on the parse graph and therefore it is expandable with new fields and protocols. Nevertheless, the extracted fields are fixed. To overcome this limitation, a parser programming language often belongs to a reconfigurable design, which is object-oriented in many cases to enable a convenient way of creating the header structure definition graph [12][13][14]. The third parser type (c) is an extended version of the second one (b), where, besides the parse graph, the subset of the extracted metadata is also reconfigurable. New header fields – which were previously not supported by the engine – can be added.

For extracting protocol metadata, the packet's header sequence needs to be decoded. To identify the subsequent header, the parser typically interprets the next header field of the current header. However, there are slim headers, which contain no information about the ensuing protocol (e.g., MPLS header contains only a Bottom of Stack bit for this purpose, while the encapsulated protocol can be arbitrary, such as, another MPLS, IPv4/IPv6, or even Ethernet over MPLS). Figure 3.a represents a typical packet structure, where each header contains information about the next header. In contrast, Figures 3.b-3.d show similar header sequence up to the first MPLS header, where the parsing process tests multiple paths and should decide on the right one. These structures also appear in practice.

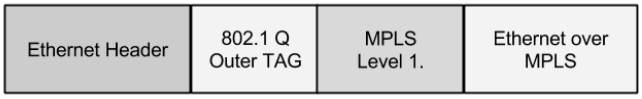


a)



b)

c)



d)

Figure 3.a-d Examples for header structures

In these cases, recognition of the subsequent header eventuates in more complex steps within the parse graph. To find the appropriate header, executing a simplified pattern matching algorithm on the next header is a common practice. We note here that there are protocols requiring multiple fields to be involved to the pattern matching process for a definite result.

1. Limitation of Software-based Parsing: Time

The main disadvantage of software-based processing methods against hardware-based ones is related to *execution time*. This is partially caused by the pure existence of the initial, minimal, constant delay of fetching the packets from the interface card to be analyzed by the processor. Besides this, there is a variation in processing delay. It depends on how the processing software is scheduled to actually get it through the operating systems' control. Depending on the depth of the analysis, processing time may differ from packet to packet. An extensive generic survey on the existing high-performance software-based packet processing frameworks is presented by Moreno *et al.* [15].

The software-based parsing and classifier algorithms are working with packets that are previously stored in the main memory by the operating system. Since the kernel-level packet processing involves polling-based reception and multiple packet queues within the processing path, the availability of the received packet for the user space parsing application is neither real-time nor guaranteed. The availability of the line-rate operation faces serious bottlenecks. Furthermore, a software solution cannot guarantee lossless packet reception, since it is executed on shared hardware resources and its CPU time is controlled by the operating system's task scheduler. If a CPU core reaches the maximum load, packets may get dropped, depending on the packet queue length. In contrast, a hardware solution is designed for dedicated purposes and tasks.

2. Benefits of a Hybrid HW/SW Approach

Software-based solutions usually have the advantage of utilizing general, considerably cheap CPUs and memory units. These resources allow deep analysis of the packets and flows.

In-depth, dynamic analysis with flow-state or protocol-state methods (see Section V for details) is more efficient with a software-based approach. The deeper the analysis goes (and the more synthesis may be carried out), the more time it

requires to come to the conclusion and provide a result. This is against the on-the-fly requirement.

It is a relatively new approach to implement some part of the DPI process in an FPGA device [16]. This is because parser and classifier tasks fit well on dedicated hardware elements of an FPGA chip [17]. However, working with circuit-defined constraints and fixed number of logic elements is a challenging engineering task.

3. A Survey on HW-based Packet Parsing

a. Requirements of a High-throughput Parser

In comparison to packet processing at 10 Gbit/s, data rates at 100 Gbit/s and beyond imply higher demand for the FPGA internal design and resources. This practically means higher operating frequency and a significantly larger data path width, typically 320-bits or 512-bits. For the target throughput, the effective frequency and data path width combination depends on the design of the Media Access Control (MAC) module. There are two commonly implemented output modes: segmented and non-segmented. In segmented mode, each word can contain portions of multiple data frames, while non-segmented mode restricts frames to start only at the first byte of each word.

In the available 100 Gbit/s Ethernet MAC IP cores [18][19][20][21], the frequency of the segmented operation is typically 312.5 MHz, combined with a 320-bit data path. Although this seems to be an implementation-specific constraint, it is wise to respect it during system design. However, there is a lower frequency mode for the target throughput that is 225 MHz with a data path of 512-bit. In non-segmented mode, the commonly applied frequency (in MAC IP cores) is 312.5 MHz and the data path is also 512-bit wide. Considering best practice, we can assume a 320-bit or 512-bit wide data path as a general choice of implementation. In order to maintain a uniform data path and a single clock domain within the FPGA, all successive pipeline stages (e.g., packet parser and packet classifier) should adapt to these design constraints.

The parser engine must treat incoming packets at line rate. Since a minimum sized Ethernet frame fits into one 512-bit word, a 100 Gbit/s parser engine must accept a new incoming packet in every clock cycle, in the worst-case scenario. This implies processing of 1.5×10^8 packets per second.

Stepping up from 100 Gbit/s, 400 Gbit/s will be the upcoming target bandwidth for next generation core networks according to the IEEE 802.3bs-2017 standard [22] approved by the IEEE Standards Board on December 6, 2017. Today, there are commercially available single data path 400 Gbit/s Ethernet PHY/MAC IP cores for FPGA architectures [23][24]. Unfortunately, some of their major operational parameters are not covered yet and therefore they can only be estimated from lower bandwidth designs.

Before starting the planning phase, the primary question should be: how can the throughput of a 100 Gbit/s design be further increased? For parameter evaluation, the main factor is the core frequency, which is limited to a few hundred MegaHertz in an FPGA device. The maximum frequency of the chip is typically not more than 400 MHz even in a high-end device. This is determined by the divided FIFO and cascadable Block RAM layout, besides the extension logic and the internal signal path. Still, how can we increase the performance of a 100 Gbit/s design up to 400 Gbit/s? Extending the frequency further is not reasonable in this case (see the next section), hence the data path width should be considered to be widened. Taking 312.5 MHz or lower frequency as a reasonable choice, data path must be at least 2048-bit wide for the target throughput.

b. Limitations of Hardware-based Parsing

Depending on the requirements, the hardware-assisted DPI may have drawbacks. The primary bottleneck is the maximum available frequency, as mentioned in the last section.

The second performance issue is also related to the size of the mapped design: the physical space required within the circuit. A hardware device has a given number of logical elements, which sets a limit to the size, functionality and complexity of the implementation. The output of the *Place and Route* operation – that results in the physical layout of the physical elements and the allocated routes between them inside the FPGA – depends on the latency. The performance of an implementation can be enhanced by using a higher capacity chip, but the physical resource set always remains a limiting factor. In contrast to the typical software development process, FPGA programmers need a lot of time and effort to reduce the size of their implementations, or to map it to another type of FPGAs.

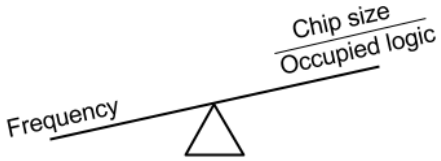


Figure 4. Latency and other requirements necessitate FPGA design trade-offs for high-performance packet processing

The number of logical units and the maximum available frequency together dictate the operational constraints for a given algorithm and its implementation. These properties restrict the operational complexity of FPGA-based DPI methods (see Figure 4). All of the memory and control elements for packet header parsing rules, for the packet classification rules, and for the payload inspection have to fit into the FPGA.

In order to detect complex header structures, the parsing algorithm must handle a lot of cases and branches. However, the number of protocol combinations is not arbitrary, because of the logical complexity and the physical resource limitations.

The multiple instantiation and the parallel processing are also constrained by the size of the chip. For example, if we have a hardware implementation, which is capable of 10 Gbit/s of throughput, a parallel 4-instance configuration can perform at 40 Gbit/s. However, the number of occupied logical units shows non-linear increment, and the longer internal routes reduce the maximum achievable frequency [30][34].

Following this example, a 100 Gbit/s design is much more complex, and we have to instantiate ten copies of this module. In addition, the algorithm has to distribute the data path, synchronize the instances, and control the modules. In contrast to the 10-instance module, it is a better choice to use a one-module approach, which is able to achieve 100 Gbit/s of throughput with one single instance. Having only a single instance makes controlling simpler, requires less hardware resources, and eliminates the need of synchronization logic between the instances. Considering the specified internal resources (e.g., number of logical and memory blocks) of the state-of-the-art FPGAs (e.g., Xilinx UltraScale), using multiple instances of a 100 Gbit/s-capable module, a throughput of 400 Gbit/s, or even 1 Tbit/s can be reasonable to achieve.

Nevertheless, it is not trivial to implement a working 100 Gbit/s design in practice, because of the presented constraints. The packet parsing algorithm should co-operate with an FPGA implemented Media Access Control (MAC) module that, at this speed, occupies a large portion of the chip, alone. As the number of occupied logical elements increases, finding electrical routes with the required latency constraint becomes more and more challenging.

In the next sections, we give an overview related scientific works focusing on FPGA implementations. To classify the packet parsing methods, we differentiated the solutions based on throughput, since design best practices differ under and above 100 Gbit/s of data rate.

c. *Hardware Accelerated Solutions under 100 Gbit/s*

Kobiersky *et al.* [12] proposed a packet header analysis and field extraction architecture, which is able to process network traffic at 20 Gbit/s on a Xilinx Virtex-5 FPGA. Packet processing implementation is generated from a standard XML protocol scheme. The parse graph is expandable with new network protocols. Operational frequency is in the range of 115 MHz and 165 MHz, depending on the input data width.

Kozanitis *et al.* [25] proposed a design called Kangaroo system, which is a flexible packet parsing solution supporting 40 Gbit/s throughput. The implementation uses a look ahead technique to parse several protocol headers in one step. Offsets are calculated using Content Addressable Memory (CAM). The design was prototyped on the NetFPGA-1G board [26], which contains a Virtex-2 Pro chip from Xilinx Inc. The achievable operational frequency, after the *place and route* phase, was 70 MHz. However, it can be implemented in a standard 400 MHz ASIC, in which the architecture supports 40 Gbit/s throughput. This solution extracts two 32-bit fields for six protocols.

Gibb *et al.* [27] proposed an abstract model for packet parsers, which can be operated in a 10 Gbit/s switch as a multi-instance design. The model enables an implementation on reconfigurable computing devices, i.e., FPGAs. The abstract design contains three main stages: header identification, field extraction and field buffer. The control structure of the first two submodules is based on a state machine, in which steps are reconfigurable through Ternary Content Addressable Memory (TCAM). Authors generated 500 different parser engines from the elaborated model with different input parameters. Then, results have been compared to each other. The study revealed that the occupied chip area depends on the field buffer and the memory size.

Duan *et al.* [28] designed a NetFPGA-10G prototype for presenting a self-adaptive programming mechanism, namely SAP. The system offers reconfigurable parsing, packet processing and adaptive control features for the network data plane. The parser module uses RAM to get the offset information for the various protocols, and uses a TCAM-based solution for the field matching mechanism. To perform a programmable processing engine, a look up module binds the match results with the actions.

While the presented works are optimized for 10 Gbit/s or 40 Gbit/s line rates, state machines and packet buffering are not viable options for higher throughput systems. In the next subsection, we present related works of higher performance packet parsers.

d. *100/400 Gbit/s Design Trade-offs and Best Practices*

Since the common width of the data path in a 100 Gbit/s MAC module is equal to 64 bytes, i.e., the minimum size of an Ethernet frame, the worst case signifies a new packet in every clock cycle. The requirement for such high throughput rules out parser engines based on store-and-forward (e.g., packet buffering, state machines) architecture. To achieve the required core frequency, simple hardware operations must be used, where the logic is able to operate on new incoming data in every clock cycle. This design implies pipeline architecture, which is the best practice in systems with a throughput above 40 Gbit/s [28][30][33][34].

However, analyzing a complex parse graph with high throughput often demands an extremely large pipeline structure, in which latency is directly proportional to the number of operations.

The best practice of the hardware-assisted packet parsing is based on a multi-stage (pipeline) design [29][31][33], which suits well to the physical architecture of the FPGA chip (see Figure 5).

Let P denote the pipeline and let P_i be the i^{th} stage within P where $i = 0, 1, \ldots n$, and n is the number of pipeline stages.

Each P_i has its own well-defined task. These stages are not complex, and thus are capable of operating at a relatively high frequency. In terms of I/O interfacing, the FPGA device fits very well into the physical route of the packet, continuing the data path as a pipeline inside the chip. There is a trade-off between operational frequency and data path width. The required throughput (packet rate) can be assured with multiple data path width and frequency combinations.

Let $O_j(P_i)$ denote the j^{th} operation in P_i, and $j = 0, 1, \ldots m_i$, where m_i is the number of operations in P_i.

Let $f_{op\_max}(O_j(P_i))$ denote the maximum frequency of the $O_j(P_i)$ operation.

Then $f_{Pi\_max}(min(f_{op\_max}(O_j(P_i))))$ is the maximum achievable frequency for P_i.

Let $k_i \in R^+$, where k_i is the weight for the routability of P_i within the chip. Then $f_{Pi} = k_i \times f_{Pi\_max}$ is the achievable

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/COMST.2018.2876196, IEEE Communications Surveys & Tutorials

maximum frequency for P_i provided by the *place and route* process. Then $f_e = min(f_{Pi})$ is the achievable maximum operational frequency of the pipeline architecture.

Evolving the architecture of P is depending on the function of the parser. Each P_i can be classified into one of the following three classes, depending on its task. Let $C_{Pi} \in \{C1, C2, C3\}$ denote the complexity of P_i.

- *C1*: P_i operates on a simple, well defined task, working on elementary operations. Parsing of a protocol header is performed by multiple consecutive P_i.
- *C2*: P_i operates on one full header of the protocol structure.
- *C3*: P_i operates on multiple headers within the protocol structure.

P can be also classified depending on the chain of the architecture:

- linear: P has a series of consecutive P_i stages.
- non-linear: P_i stages can operate in parallel relative to each other.

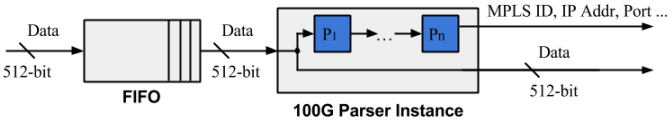


Figure 5. Pipeline-based 100 Gbit/s protocol parser architecture in FPGA

The complexity of the protocol header structure is the main design challenge during the planning process to achieve the highest f_e. As an example, dynamic IPv4 header is advisable to parse with multiple P_i stages. If the header sizes in the protocol structure are not variable, the complexity of the stages does not set a bottleneck for operating f_e frequency. The main design goal for a P pipeline architecture is to define optimal C_{Pi} classes for each stage in order to achieve the highest frequency on a given FPGA device.

Handling packet data inside the parser engine, which eliminates the need for packet buffers, is a drawback of the pipelining technique in terms of resource utilization. This internal data path can occupy significant chip area, depending on the implementation mode and the target hardware type. As an example, Virtex-7 FPGA family is currently a popular choice for high performance parser engine implementations, where basic elements within the Configurable Logic Blocks (CLBs) are D-type flip-flops, multiplexers and shift registers, grouped in slices. When the internal packet data path is built from D-type flip-flops, it needs 64 slices for each 1 clock cycle long parsing step. According to our experimental simulations with a medium category Virtex-7 chip, a parser with near 0.5 µs latency therefore requires 79360 D-type flip-flops just for the internal travelling of the related packets, which sums up in 9920 slices. This means that a single data path can occupy nearly 15% of a medium category Virtex-7 chip. For 400 Gbit/s of throughput, data path should be four times larger than in the 100 Gbit/s case. Calculating with 2048-bit, the number of required slices increases up to approx. 4×10^4.

e. *Hardware Accelerated Solutions for 100 Gbit/s and above*

Attig and Brebner [13] introduced a high-level packet parsing description language. Codes written in this language can be compiled to a high-performance FPGA device. The generated architecture allows run time programmability during the field extraction operation, as well. The solution supports 400 Gbit/s line rate parsing on a single Virtex-7 FPGA, with a data path wider than 512-bit. The language supports an optional header structure, which enables decoding new network protocols.

Pus *et al.* [29] found latency to be a critical performance factor in high throughput systems. However, due to the high frequency constraint, a heavily pipelined design cannot be optimized for latency or chip area. Accordingly, in this work authors describe a hand-optimized parser, which is able to process data at rates higher than 100 Gbit/s. The architecture relies on uniform pipeline stages, which are optimized for dedicated tasks. A classic 5-tuple prototype was implemented on a Xilinx Virtex-7 870HT chip.

Brebner and Jiang [14] proposed a high-level object-oriented language, namely PX, which is designed for packet processing engine specification. The language covers the hardware designer's tasks and lets the user deal with the main targets. The compiler generates a pipeline architecture with live re-programming property. Their paper gives an overview of the main generated functions: a parser and a classifier engine. These functions are demonstrated through a 100 Gbit/s OpenFlow implementation, in which the parsing engine extracts 12 OpenFlow fields from four headers. The architecture uses 512-bit data path for 100 Gbit/s throughput.

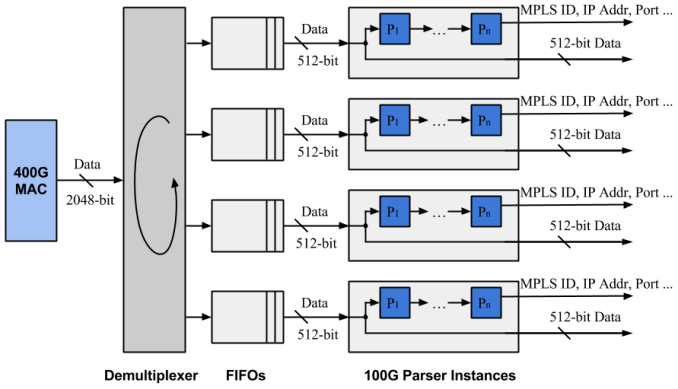


Figure 6. Multiple instance parser engine

Benácek *et al.* [30][31] presented a high-level parse graph description language, namely P4. The compact solution contains a code generator part, which converts the P4 graph into synthesizable VHDL code. Test results show that the generated VHDL implementation is able to process traffic at 100 Gbit/s speed on a Virtex-7 FPGA device. P4 defines five basic language tools to determine header format, state machine, extracted headers, actions, and control flow. The generated implementation applies a pipeline architecture with

512-bit wide data bus.

Hu *et al.* [32] proposed an SDN-based architecture. The solution uses the P4 abstract language for the forwarding plane to resolve the scalability problem space.

Li *et al.* [33] introduced a programmable parser, namely P5, which is able to operate at 100 Gbit/s speed on 128 bytes packets. The solution uses a circular pipeline for parsing (lower than 200 μs latency), where each header is parsed in a loop. An action RAM is used to obtain the parsing information, which fields are needed to be stored. The prototype is demonstrated to operate at 200 MHz on an Altera FPGA-based platform, namely NetMagic (which only has 8 x 1 GE interfaces).

Pus *et al.* [34] proposed a pipelined packet parser architecture for over 100 Gbit/s operation throughput. The authors offer 2048-bit wide data path for over 100 Gbit/s. Each pipeline stage uses extra input information about the protocol stack (e.g., header type, header offset, partially aligned start, last byte of the packet). The hand optimized prototype is able to process more minimum sized packets in a 2048-bit datapath. Test results show that 400 Gbit/s throughput is achievable in a Virtex-7 FPGA.

Da Silva *et al.* [35] used P4 for describing the parser graph, and achieved a low-latency implementation when compared to [27] and [30]. The authors used high level synthesis, their RTL code is generated from the C++ description using Xilinx Vivado HLS and synthesized with Xilinx Vivado. They found that to achieve 100 Gbit/s throughput, their optimum design trade off is at 320 bits data bus length with 312.5 MHz clock frequency with a Virtex-7 FPGA

Orosz *et al.* [47] created a parse graph of 14 elements, covering the typical combinations of L2-L4 protocol encapsulations. This solution supports GTP (GPRS tunneling protocol), a typical element of mobile core networks, as well. The base platform – called C-GEP – is an actual Virtex-6 FPGA-based, 100 Gbit/s-capable network node. The design trade-off found by the authors in this architecture is 312.5 MHz operating frequency and 512 bits long data bus. Among its various applications, C-GEP is used as a high-speed network monitoring equipment [17].

f. Lessons Learned in Packet Parsing

In contrast to a design operating at 100 Gbit/s, a 400 Gbit/s architecture must handle wider data path (reasonably more than 2048-bit). There are two modes of MAC modules: segmented, when one word can contain more than one Ethernet frame; and non-segmented, when each Ethernet frame starts at the beginning of a 2048-bit word. Although non-segmented mode is simpler to implement, for frames that are shorter than the word-size, the difference must be filled out with padding bits. At 400 Gbit/s, it is sane to choose the operation mode of the MAC module to be segmented. This is due to the minimum size of an Ethernet frame. In the case of non-segmented processing, the incoming 2048-bit word may contain a minimum sized packet (512 bits) and therefore the number of padding bits will be 1536. This is a significant waste of performance, and needs large packet buffers inside the MAC module, which is not a reasonable option. If we assume segmented operation, multiple packets can arrive concurrently in one 2048-bit word. This scenario implies parallel packet parsing in every clock cycle, in worst case. According to the principles of the architecture, a multi-engine parser seems to be the best practice with 100 Gbit/s instances.

Table I provides an overview of the surveyed HW accelerated parsing solutions. Since these are based on FPGA, all the solutions support reconfigurable parse graphs in theory; however, there could be various reasons why some papers do not mention this. Similarly, various protocols may have been implemented besides those mentioned in the papers, we only list the ones actually appear there. The notes of Table I are supposed to point out the novelties of the paper.

In order to reach or even exceed a processing throughput of 100 Gbit/s, high performance parser engines implement an internal data path that is 320-bit wide, at least. As we highlight related works for 100 Gbit/s packet parsing, we conclude that due to the relatively large data path width (comparing to the 64-bit data length for 10 Gbit/s) and the significantly higher operational frequency (over 195.3 MHz) a high-performance FPGA chip, e.g., Xilinx Virtex 6/7 or Altera Stratix, is an essential requirement for these implementations.

Another factor for resource allocation is the design principle itself: single-instance (Figure 5) versus multi-instance (Figure 6) engine. For 100 Gbit/s of performance, a single-instance parser seems to be a reasonable design, since core frequency related to 512-bit data per cycle can meet the throughput requirement. However, in case of a complex parse graph, this design may not be able to operate at 312.5 MHz and therefore should be substituted with multiple lower-throughput instances. While this engine operates at a lower frequency, with softer implementation constraints, the appropriate scheduling algorithm performing synchronization between instances claims extra logic.

As the result of the parsing inspection phase we get a set of protocol metadata directly acquired from the captured packets. In terms of functionality, configurational flexibility of the parse graph is important, in order to support the successive DPI-phase (i.e., the classification engine) with the aptitude to adapt to a wide scale of network configurations. Actually, it is one of the key features of the parser engine to enable the modification of the parse graph to update protocol field layout or inserting new networking protocols.

For further reading, we suggest [13][25][31][34][47] as fundamental literature in the field of hardware-accelerated packet parsing.

Table I – An overview of FPGA accelerated parsing solutions

Reference	Throughput	Chip Class	Data Bus	Operational Frequency	Demonstrated Protocols	Notes
Kobiersky et al. [12]	20 Gbit/s	Virtex-5	128 bits	115MHz & 165MHz	Ethernet, VLAN, MPLS, IPv4, IPv6, UDP, TCP	Parse graph is extendable.
Kangaroo system [25]	40 Gbit/s	ASIC	192 bits	400 MHz	generic	Look ahead tech. for parsing; uses CAM.
Gibb et al. [27]	10 Gbit/s	generic FPGA	640 bits	172.2 MHz - 184.1 MHz	various L2-L4 protocols, dozens when encapsulated	Very detailed parse graphs; uses TCAM.
Duan et al. [28] eval. with NetFPGA-10G	10 Gbit/s	Virtex-5	256 bits	375 MHz	Ethernet, VLAN, MPLS, IP, TCP	Self-adaptive programming mechanism; uses TCAM-based field-matching.
Attig and Brebner [13]	100 Gbit/s / "400 Gbit/s" simulated	Virtex-7	2048 bits	274 MHz - 335 MHz	Ethernet, VLAN, MPLS, IPv4, IPv6, UDP, TCP, RTP	Device utilization stays below 25% for 100Gbit/s. Specific description language allows for decoding new protocols.
Pus et al. [29]	> 100 Gbit/s	Virtex-7	2048 bits	195.3 MHz	Ethernet, VLAN, MPLS, IPv4, IPv6, UDP, TCP	Heavily pipelined design with manually optimized parser.
Brebner and Jiang [14]	100 Gbit/s	Virtex-7	512 bits	274 MHz - 335 MHz	Ethernet, VLAN, MPLS, IPv4, IPv6, UDP, TCP, OpenFlow	SDN-capable design with a four-stage pipeline, utilizing TCAM.
Benácek et al. [30][31]	> 100 Gbit/s	Virtex-7	512 bits	195.3 MHz	Ethernet, (2x) VLAN, (2x) MPLS, IPv4, IPv6, UDP, TCP, ICMP(+v6)	Uses P4: Parse-graph description language – generating pipelined parser design; manual optimization.
Hu et al. [32]	N/A	N/A	N/A	N/A	generic	Uses P4; generated SDN-capable design.
Li et al. [33]	"100 Gbit/s" simulated	Altera	N/A	200 MHz	Ethernet, VLAN, IPv4/6, TCP, UDP, (NV)GRE	P5: programmable parser with packet-level parallel processing; predictive parsing.
Pus et al. [34]	> 100 Gbit/s capable of 400 Gbit/s	Virtex-7	2048 bits	195.3 MHz	Ethernet, (2x) VLAN, (2x) MPLS, IPv4, IPv6, UDP, TCP	Modular parser design, nine-stage pipeline, manual optimization.
da Silva et al. [35]	> 100 Gbit/s	Virtex-7	320 bits	312.5 MHz	Ethernet, (2x) VLAN, (2x) MPLS, IPv4, IPv6, UDP, TCP, ICMP(+v6)	Uses P4; advanced parser, achieves lower latency when compared to [27] and [30].
Orosz et al. (C-GEP) [47]	100 Gbit/s	Virtex-6	512 bits	312.5 MHz	Ethernet, (2x) VLAN, (2x) MPLS, IPv4, IPv6, UDP, TCP, GTP	Parse graph with 14 elements, covering typical encapsulation combinations.

IV. Second Phase: Packet Classification

As terminologies, *packet filtering* and *packet classification* show differences in terms of the output they provide. From the viewpoint of the inspection process, the main purpose of *packet filtering* is to separate the interesting traffic from the non-interesting one. Accordingly, filtering is a common technique in network security management. In contrast, a *packet classification* process determines a class of traffic or an output interface for the incoming packet that is common in network management (see Section III.1 for exact definitions).

However, considering the packet processing pipeline in a DPI system, both methods (i.e., filtering and classification) are nestled between the packet parsing and payload inspection stages.

In this section we are focusing on packet classification based on filter-rules. This selection method enables an implicit traffic classification. This means that the packet classification phase can support parallel payload inspection modules assigned to different traffic classes with different analysis features. The filter rules are working with protocol fields extracted by the parser stage. With the evolution of the Internet, new protocols are continuously appearing on the network links. The requirement for high numbers of user-defined filter rules is also a major design challenge. With the emergence of QoS-sensitive applications (i.e., Voice over IP, video conferencing, IPTV, video on demand, etc.), next-generation firewalls, intrusion detection and intrusion prevention systems (IDS/IPS), the filter-based classification method has become a common practice. These systems, in extreme cases, need to classify the incoming data packets at line-rate. This research field is well investigated, and thus, there are already several proposals for high performance traffic filtering as well as traffic classification. Within a networking device, this rule-based processing must support line-rate throughput. To achieve this goal, there is a need of

efficient parsing algorithms to handle extremely high packet rates, and to classify packets based on the extracted fields.

In recent years more than a hundred designs were published to satisfy the categorization demands with novel approaches. These are mainly 10 Gbit/s or 40 Gbit/s throughput capable works, most of them are based on classic solutions [36][37][38]. Nevertheless, 100+ Gbit/s designs only appeared in the last few years.

The classification phase of the DPI process expects information about the incoming packet and about the base of the classification. The former is a set of the extracted header fields, coming from the packet parsing stage, which determines the complexity of the lookup engine. The latter is a set of classifying rules. The structure of a rule is constructed from protocol fields using logical operators. Among the matching rules, the highest priority match gives the result of the classification engine. Classical solutions implement 5-tuple methods, where the incoming packets are categorized based on *Source IP*, *Destination IP*, *Source Port*, *Destination Port* and Transport *Protocol*. There are a few high-performance works, which are operating on more than 5-tuple. Nevertheless, with the emergence of the Software Defined Networking (SDN) technology and the OpenFlow standard, handling OpenFlow-like tuples is an important research area. Hsieh *et al.* [39], for example, present a GPU-based classifier dedicated for OpenFlow-based SDN environment, which supports 15-tuple ruleset with 100K entries operating at 163 Mpps. However, the widespread traffic filtering solutions are based on TCAM (Ternary Content Addressable Memory). Nowadays, TCAM is expensive and not scalable in contrast of an SRAM-based system, and the power consumption is also a design drawback. Nevertheless, more algorithms were proposed for increasing the performance of a TCAM-based design [40][41][42][43][44], but the maximum bandwidth is not higher than 100 Gbit/s. Because of the limitations, new groups of classification engines were published based on novel internal architectures.

To increase the operational throughput, applying parallel algorithms is an effective design principle. A general method for parallelization is splitting the header information, where the search engines perform parallel and independent searches. There are two high-throughput capable design principles:

- *Decomposition based algorithms*
  The incoming header information is split into sub-vectors, where each vector is compared to an appropriate ruleset, independently. The results of the independent searches are aggregated in later steps for getting the final result.
- *Tree-based algorithms*
  The header information is compressed in trees for the search engines, where node calculation and update information calculation are often offline processes.

There are proposals focusing on the optimization of the rule set in order to save storage space or decreasing the time required for classification. Hager *et al.* [45] introduced a rule set transformation strategy, namely Minflate, which uses two steps: rule set minimization and decision tree encoding. The first step generates the optimized form of the original rules in a sub-second preprocessing time for up to 5000 rules. The authors implemented the solution in C++ and used *iptables* and *tcpreplay* in the test environment. Test results show that Minflate outperforms the existing software-based packet filters.

Zheng *et al.* [46] proposed a prefix length-based clustering method. The solution uses a partitioning technique to cluster the rules based on the prefix length into a quad-tree. Test results show that a C++ implementation of the algorithm can save 37% of time comparing to a simple area-based quad-tree method, in the case of a 100k ACL rule set.

In the next sections, we focus only on high-performance designs, which can operate at a data rate above 100 Gbit/s.

1. Limitation of Software-based Classification

The main limitation of a software-based classifier is similar to that in software-based parsing, described in Section III.1 “Limitation of Software-based Parsing: Time”.

Examining a general software operation, the basic scheduler-idea inhibits the parallel operation of rule-set update and classification. The scheduler always has to stop the sequential rule fitting process for more clock cycles, while the rule update is done, or just serve other system or user processes.

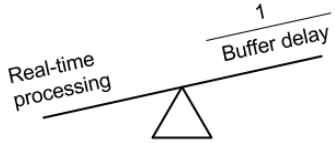


Figure 7. Software-based packet processing trade-offs

Software solutions always operate on previously stored (buffered) packets, which, beyond 10 Gbps or by using more than 5 tuples, cannot guarantee real-time analysis. Figure 7 aims to visualize this trade-off. Using general memory elements is also a performance bottleneck. DDR memories are effective in large block read/write processes, instead of working on small sized data. In contrast with a dedicated hardware-based solution (e.g., TCAM, BRAM), general purpose memories cannot serve the consumer in every clock cycle, although it is a requirement in high-speed classification.

2. Benefits of a Hybrid HW/SW Approach

The overall latency of the packet processing path in the hardware is determined at clock cycle level and therefore the clock-based operation of the implementation can be tested through simulations.

The programmable pipeline structure can be a further advantage. Notably, the programmable logical units enable us to work with very large data path within the hardware. This method ensures lossless packet processing, since each pipeline stage is able to process a new data packet in every clock cycle [47]. A single-chip pipeline architecture is able to extract the packet header fields in real-time, even if the header structure is not known in advance. Each pipeline stage requires a fixed number of clock cycles to process the packet independently of

its header structure.

In this way, packet parsing and classification methods work with constant delays. The pipeline structure allows an uninterrupted, line-rate operation. Within the pipeline, consecutive stages detect the header structure and extract the required field values. In addition to the real-time processing, a hardware device can perform very accurate packet timestamping based on a local clock.

After the field extraction, the tuples can be stored in the appropriate memory (i.e., block RAM, distributed RAM) within the chip. In this case, the reordering problem also has to be addressed. Inside the data path, the extracted field-set must be synchronized with the packet it belongs to. Delaying the extracted metadata and passing it in-sync with the packet is a good solution.

A hardware-based traffic filtering algorithm has many advantages. Let R denote the ruleset in a classification engine (Figure 8). Let R_l denote the l^{th} rule in R, and $l = 0,1, ...,$ where s is the number of rules in R. The incoming data is classified in real time by the engine, based on the ruleset R. Typically R_l includes multiple conditions, which conditions belong to the extracted packet header data [48][49].

A traffic filtering method examines these fields through the rule conditions, and gives one result at its output. A hardware device can support this method in several ways [50][51][52], in contrast of the software solutions.

The header analysis is a well parameterizable process. Many conditions can be tested in parallel, even within one clock cycle. These results can also be AND-ed in the next cycle, which is a straightforward hardware operation. Parallel examination of the rule entries can reduce the resource requirements of the module [51]. This way, a very fast classification algorithm can be implemented, which provides the result within a few clock cycles.

Another advantage of an FPGA chip is the relatively high internal storage capacity. A bigger device contains more than 10 Mbit on-chip memory, which enables storing the R rule set along the logic elements. The operation frequency of the internal memory is appropriate to a well-designed method.

The dual-port mode enables a run-time rule update [53]. Internal memories can store even a few thousands of classification rules [54], eliminating the need of an intensive memory read/write software function, which can be a bottleneck of the software throughput. The ordered list of the rules can be implemented through pipeline steps, memory addresses, or dedicated logics.

Many of the papers that present proposals for traffic filtering [52][53][54] do not investigate the process of filter rule update, which can be also a bottleneck for the operational frequency.

There are two types of rule update algorithms. In the standard method, the incoming data packets are routed to another path, which bypasses or drops the packet, depending on the operation mode. The packets are not classified during the update process, analysis systems need post-processing from storage elements. Software solutions are typically based on this standard method. Alternatively, if rule update is a frequent event, it is a feasible option to implement a backup table and using it during the update process.

In contrast, a hardware device is able to do the run-time update operation, while the next incoming packet is investigated according to the new rule set.

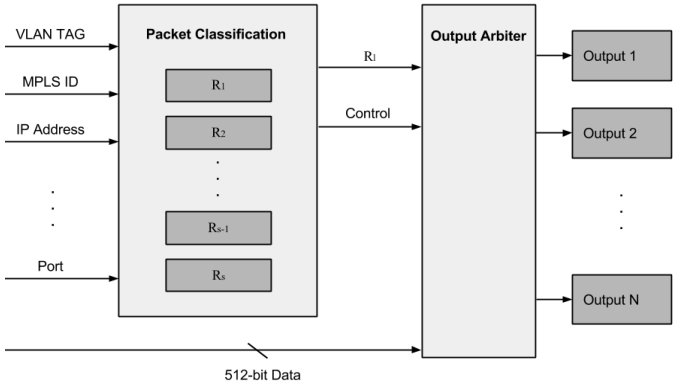


Figure 8. FPGA-accelerated traffic filtering architecture for 100 Gbit/s

The rule order can be implemented through pipeline steps, memory addresses, or logical planning.

Fiessler *et al.* [55] presented a hybrid classifier solution that supports a software-based classifier with a resource efficient FPGA circuit. The test setup is based on a NetFPGA SUME board [26], which is used to accelerate the netfilter/iptables firewall. The pipeline architecture is able to operate at 180 MHz frequency and applies a 512-bit wide data bus. The authors combined a fast rule set update mechanism with two approaches. MPFC [56] is used for a specialized matcher, and StrideBV [53] is used as a generic matcher function. They built the idea into a hybrid, FPGA and Linux firewall-based classification system, namely HyPaFilter [57]. The key challenge, which has to be solved in a hybrid system was the priority management between the software and hardware defined rules, in parallel. While the maximum operational throughput is about 85 Gbit/s theoretically, the authors aimed a 40 Gbit/s throughput capable hybrid design.

The authors further developed the hybrid solution [58], and the demonstrated system achieves thirty times larger throughput than a pure software-only packet processor.

3. *A Survey on HW-based Traffic Classification*

a. *Requirements of a High-throughput Classifier*

Besides the described high-speed packet processing requirements in Section III.3.a "Requirements of a high-throughput parser", a classification engine has claimed for store and search tasks.

The basic requirement is to handle hundreds or thousands of rules, where the ruleset is based on minimum 5-tuple. Towards the SDN technology, more than 5-tuple (e.g., 10 or 12) is the base of a rule fitting task [59].

Since in high-speed networking every clock cycle can contain one or more new, minimum sized Ethernet frame (e.g., at 400 Gbit/s), the classifier engine must be able to handle new packet or packets in every clock cycle. Next to the parallelization and multi-instance designs, the frequency-related requirements need to be achieved, as well.

The ruleset needs to be modifiable and extensible, on-the-fly. This is due to the ever-changing nature of network properties and operator needs, as well as the appearance of new protocols in the network traffic. A filtering module must be reconfigurable in real-time, without the risk of inconsistent decisions on a running analysis.

Furthermore, the examined header-combinations should also be variable. Deleting or adding new parameters and header fields are requirements of a complex classifier.

b. Limitations of HW-based Classification

The limitations of a hardware-based classification engine are discussed already in Section III.3.b, "Limitations of HW-based Parsing".

c. Classic High-performance Solutions

Gamage *et al.* [52] proposed a 5-tuple based, high throughput capable TCAM architecture. The algorithm is based on parallel operation with multiple TCAM chips for simultaneous traffic filtering. The proposed architecture consists of two stages: a Distributor, with a small cache to store the popular rules, and a TCAM chip. If a packet arrives, the Distributor unit looks up in the cache. If there is a match, the action of the fitting rule is appeared as output result, and the second stage is skipped. In the other case the Distributor sends a search request signal to the appropriate TCAM chip. A connection resolver (CR) unit handles the incoming requests and the TCAM resource sharing. The design was simulated with 10^4 rules, operating at 100 MHz core frequency. The Distributor and CR unit was synthesized on a Virtex-2 Pro FPGA. With appropriate design parameters, the algorithm can achieve 200 Gbit/s throughput, operating.

Jiang *et al.* [60] proposed a parallel lookup engine, named Field-Split Bit Vector (FSBV), in which the packet header fields are split into 1-bit subfields. The lookup engine performs independent searches on each subfield. Each parallel search results in a 2N wide vector (N is the number of rules), and each bit of the result vector represents a rule. A bit is set to '1', if the appropriate rule fits on the subfield, otherwise set to '0'. The result of the bitwise AND operation gives the fitting rules on the actual header fields. This algorithm was implemented on a single Virtex-5 FPGA chip, in which the design principle is a 5-tuple classification architecture. The implementation uses 7 pipeline stages with Block RAM memory resource. The main design advantage – beside achieving 100 Gbit/s throughput – is that memory occupation is a linear function of the ruleset size.

Another bit-vector based classification algorithm, StrideBV, was proposed by Ganegedara *et al.* [53], in which the maximum achievable throughput is 400 Gbit/s, operating with four 100 Gbit/s bandwidth capable engines. Although this is a 5-tuple design, the scheme can be expanded for more-tuple systems. The proposed method is based on the FSBV [60] algorithm. The FPGA implementation operates with parallel pipelines. The novel design principle in contrast with the FSBV algorithm is the *k*-bit subfield splitting instead of the 1-bit examination. Working from the output of the StrideBV engine, a Pipelined Priority Encoder module results in the final match from an *N* bit-vector, where *N* is the number of rules. This is shown by Figure 9. Each pipeline stage results in an *N*-bit vector, which are AND-ed with the output of the previous stage. The design was implemented on a Xilinx Virtex-6 FPGA with 512 ruleset size limitation.

d. Enhanced Designs, based on Classic Algorithms

Sanny *et al.* [61] compared the design of a TCAM scheme and a StrideBV [53] solution. The FPGA implementation of the TCAM design also handles *don't care* bits. Each method was implemented on a state-of-the-art Virtex-7 FPGA chip, to summarize and examine the performance of the algorithms. During the research, the ruleset size was limited to the range of 32-2048. The StrideBV algorithm – depicted by Figure 9 – was also tested in two different designs. This raises an implementation related question: "Shall a high throughput system build from distributed RAM resource, or Block RAM based algorithms are also efficient?" Summarizing the results, the StrideBV implementation achieves higher operation capacity than the TCAM design. More precisely, the StrideBV version based on distributed RAM was 6 times better, and the Block RAM based StrideBV algorithm was 4 times better than TCAM. However, the TCAM design has *O(1)* lookup time. The implemented StrideBV algorithm based on DRAM logic can achieve more than 100 Gbit/s throughput, working with 2048 ruleset size.

Qi *et al.* [54] proposed a memory-efficient traffic filtering algorithm. The design was based on a previous work, namely HyperSplit [36], which is a tree-based binary search algorithm. The model uses heuristics to select the splitting point between classification decisions. This method can be easily implemented in hardware.

The algorithm is optimized for two things: to reduce the pipeline stages and to control the memory usage (Block RAM allocation). There are two practical optimization processes implemented here. Firstly, the tree-height-reduction process, which merges a non-leaf node with its children into a single node. Secondly, the leaf-pushing algorithm, which limits the number of nodes in each stage, and supports on-the-fly rule update. The FPGA implementation was carried out in a Virtex-6 FPGA chip, and can achieve more than 100 Gbit/s of throughput combined with 5×10^4 ruleset size support. The 10^4 ruleset size can operate at 115.4 MHz.

Kennedy and Wang [62] proposed a parallel, high throughput capable classification architecture based on a decision tree. The work modifies and improves the performance of the classic HyperCuts [38] solution to optimize it for hardware implementation. First of all, the cutting scheme was modified to reduce the memory access by placing the leaf nodes closer to the root node. In addition, the rule storage method was also improved by reducing the number of bits and storing the actual rule in the leaf node, instead of storing the pointer to the rule. Although the implemented architecture is a classic 5-tuple based system, it can support IPv6 header fields with some modification.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/COMST.2018.2876196, IEEE Communications Surveys & Tutorials

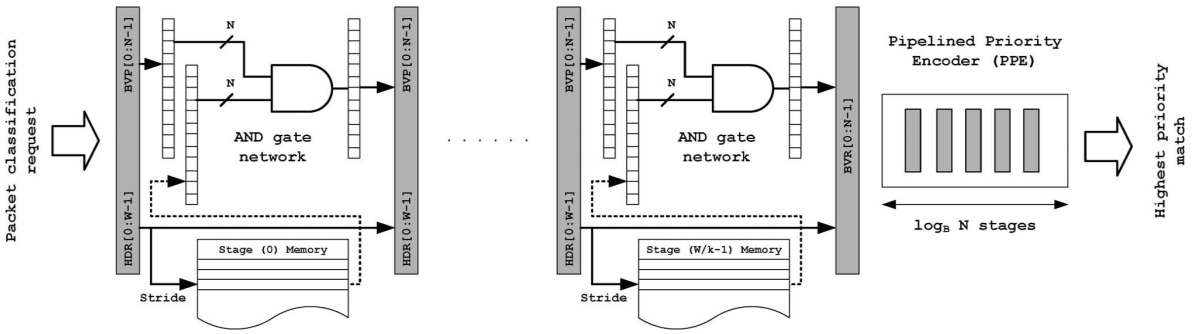


Figure 9. StrideBV algorithm [49]

The proposed solution was based on multiple engines, and was implemented on Cyclone III and Stratix III FPGAs from Altera Corporation, and can achieve even 138.56 Gbit/s throughput (e.g., 433 Mpps) combined with 8×10^4 rule set size. In a case of a Stratix III FPGA, it can provide pretty low power consumption.

Wee and Pak [63] presented a HyperCuts [38] based classification engine. The algorithm achieves 30% higher performance in memory access than HyperCuts by reducing the height of the cutting decision tree. The method applies a complicated pointer table (pivot-based cutting) for the table size, which leads to the memory access reduction.

Ganegedara *et al.* [53] originally proposed a bit-vector based classification engine, called StrideBV (shown by Figure 9). Furthermore, they presented an architectural comparison [64] between a serial and a parallel bit vector-based solution. To compare the approaches in practice, the designs were implemented in a Virtex-7 2000T FPGA. The serial architecture, including pipeline steps, has more latency, but it can operate on higher throughput. In contrast the parallel architecture has less latency regarding the classification steps, but the maximum achievable core frequency is also lower. The memory consumption of the algorithm does not depend on the architecture. The proposed serial or parallel architectures are both able to operate above 100 Gbit/s.

Qu *et al.* [65] proposed a classifier algorithm with fast update feature, based on the DBV (Dynamic Bit Vector) approach. The pipeline architecture contains a V-matrix, which stores the valid bit of the rules, and a B-matrix, which stores the bit-vectors. The V-Matrix is used only at the first pipeline, whereas the later stages compare only the header fields and the B-Vector. At the end of the pipeline, there is a priority encoder module. The V-matrix grants fast (e.g. 1k updates/second) rule update process. The design was implemented on a Virtex-6 FPGA, and was able to operate with 120 Gbit/s of throughput. With a rule set size of 512, a distRAM based architecture can operate at 337 MHz, in contrast of the BRAM based solution, in which the maximum frequency is 235 MHz.

Khatami *et al.* [66] proposed a decision tree-based, fast updatable architecture. The multi-pipeline design uses LUTs instead of memory blocks, and also uses the partial reconfiguration property of a Virtex-6 FPGA device. The architecture is divided into N decision trees, which is mapped onto a parallel, multi-pipeline engine. Each piece of header information is propagated through all the N pipelines in parallel, where each stage operates on one header field. The architecture is able to operate at 120 Gbit/s data rate, in a case of 10^4 real-life ruleset size.

Pus *et al.* [67] proposed an FPGA based classification algorithm, which operates together with an off-chip SRAM memory. Despite of the external memory access, the architecture is able to operate at 100 Gbit/s of data rate. The proposed work uses perfect hashing for searching in constant time. To achieve this, the perfect hash table is stored in the off-chip memory. The classification engine is based on three steps. The first step calculates the longest prefix match on each field. After that the results are mapped to the rule set, using perfect hashing function for fast and constant search. If the header fields do not match any rule, the output of the hash function can result in an invalid rule number. Therefore, the final step checks the packet against the resulted rule. The algorithm is a standard 5-tuple classification method, but scales well with more external memories.

Yang *et al.* [68] proposed a novel traffic filtering algorithm, namely D^2BS (Dynamic Discrete Bit Selection), which is based on a previous work, DBS (Discrete Bit Selection). D^2BS can work effectively at for various types of platforms (e.g., network processors, FPGAs). The algorithm has two phases: a preparation and a classification phase. Assume that R is a rule set, which consists of r rules. It can be observed, that some bits can effectively split R into subsets. These bits are called the *E-Bits*, and are selected in the preparation phase. To filter the *E-Bits* from the incoming packets, a masking vector (M-Vector) is also generated in this phase. The filtered bits are concatenated as a bitstring, which are used to index the D-Table, a dynamic index table. This D-Table is generated in the preparation phase, in which each cell contains a pointer to a memory block (S-Block). S-Blocks contain the sub-partitions of the original R ruleset, where each rule is ordered based on its priority. The classification phase operates in the incoming packet, utilizing the tools mentioned above. The FPGA based implementation is capable of running at 135 Gbit/s, in a case of 10^4 ruleset size.

Hager *et al.* [69] presented a fast rule-set modification algorithm, namely SFL (Small, Fast, Lazy), which is a platform independent solution. The authors use update buffers and linked list techniques for the rule set modification. A software implementation is used to evaluate the SFL approach on existing classification algorithms (e.g. HyperSplit [36]). Results show that SFL increases the responsiveness of a fast classification method, and the packet processing performance is significantly better than a non-SFL solution.

Nakahara *et al.* [70] introduced a DSP block-based FPGA solution for parallel packet classification. The implementation

throughput is 640 Gbit/s for a minimum Ethernet frame on a Virtex-7 VC707 evaluation board, calculating with 600 MHz clock frequency. The first stage of the classifier uses a modified HiCuts [71] solution for rule partitioning, and the system converts each field groups into DSP blocks. The authors implemented a small CAM-based update mechanism, where the CAM is also built from FPGA logic.

e. Towards more than 5-tuple Classification

Orosz *et al.* [47] proposed a 14-tuple parser/classifier implemented and verified within the 100 Gbit/s network monitoring platform, C-GEP [16]. It uses an optimized version of Field-Split Bit Vector (FSBV) [60], which is a parallel, lookup-based filtering method that uses subfields from the original header information. This optimized version of FSBV supports the operating frequency of 312.5 MHz, instead of the original one, designed for 167 MHz. While the parse graph is only reconfigurable in design time in order to keep the lossless nature of the solution, the classification rules are run-time reconfigurable.

Sun *et al.* [72] proposed a parallel architecture for intermediate result aggregation in decomposition-based designs. The idea was implemented on a Xilinx Virtex-6 FPGA chip, where the resource occupation and throughput were analyzed and compared with a bit-vector solution. The proposed method can achieve 107 Gbit/s data rate in a single chip. Summarizing the results, the proposed algorithm is more efficient for large rulesets with small tuples, than a bit-vector solution. However, when increasing the header fields in the rule, a bit-vector solution occupies less resources.

Fong *et al.* [51] proposed a high-performance traffic filtering algorithm. The architecture was implemented in a single Virtex-5 FPGA chip, using 11-tuple OpenFlow-like rulesets. This was the first high-throughput capable design, which was able handle more than 5-tuple. The method supports up to 10^4 complex rules in a one chip solution. The ParaSplit implementation can achieve 1 Tbit/s throughput in a single FPGA chip, operating with more than ten ParaSplit instances. The proposed architecture is a decision-tree based design, in which the HyperSplit [36] algorithm was used for the tree setup. The search engines operate in parallel, and the final result is aggregated from intermediate results. The design uses dual-port Block RAMs to achieve 120 Gbit/s of throughput combined with 10^4 ruleset size.

Qu *et al.* [73] proposed a scalable, 2-dimensional, horizontally and vertically pipelined architecture. The design is based on PE modules, which are typically pipeline stages and can handle both range and prefix match. The horizontal direction of the pipeline architecture propagates the bit vectors, and the vertical direction propagates the bits of the header subfields. In addition, each row contains a Priority Encode module for the result matching. The main principle of the work is to achieve a high throughput capable, 15-tuple based classification engine combined with a simultaneous update algorithm (e.g., modify, delete and insert operations). The authors recognized the importance of the SDN technology, and proposed a classification system working with an OpenFlow-like ruleset. The maximum achievable throughput of the implemented algorithm can be even 190 Gbit/s with 1 million updates per second, in a case of 10^3 rule set size. Using a Virtex-6 FPGA, the core frequency is about 300 MHz. The authors further elaborated this work in [74].

Chang and Hsueh [75] proposed Virtex-6 FPGA-based classification schemes for OpenFlow switches. The presented method operates on 12-tuple fields and able to process more than 5×10^3 OpenFlow rules. The FPGA implementation is based on a fully pipelined architecture and uses the StrideBV [53], as a high-performance classification solution in the prefix match stages. Test results show that the designed system is able to operate at 380 MHz frequency on a top chip from the Virtex-6 family.

Abdulhassan and Ahmadi [76] proposed an R-tree based solution for OpenFlow rule classification (15 fields rule set). The paper presents techniques for R-tree construction from a rule-set, and also for parallel queries from the tree. Experimental results show that the solution is able to process 350000 packets per second with 32 parallel threads. Test results show that the proposed method uses less memory accesses than HyperCuts [38].

Table II provides an overview of the surveyed hardware accelerated traffic classification solutions.

f. Lessons Learned in Packet Classification

All of the overviewed works apply high-degree parallelism to achieve throughput of 100 Gbit/s and above. Clearly the best practice for a high-performance traffic classification engine is a parallel, hardware-based pipeline architecture with multiple search engines. The internal scheme can be based on trees or bit-vectors. Despite of the common tasks (e.g., handling highest priority match, on-the-fly rule update), each design has its own implementation challenge.

In terms of performance, the discussed classifier designs show a large variance based on the FPGA device they are implemented in. All of the overviewed solutions can perform at 100 Gbit/s with a varying operational frequency that depends on the used FPGA resources and internal routing scheme. We recommend [51][53][60][61][65][67] as fundamental works in the field of high performance packet classification.

Beyond throughput, other differentiators are the complexity of the filtering rules (i.e., the number of tuples) and the possibility to perform dynamic ruleset update. State-of-the-art classifiers support more than 5-tuple rule constructions and perform the update task without interrupting the matching process. Reflecting to the adaptivity of the software defined networking technologies, most advanced designs support Openflow-based rule definition or allow the modification of the rule structure.

Table II – An overview of HW accelerated traffic classification solutions

Reference	Through-put	Chip Class	Oper. Freq.	Base Solution	Notes
Gamage et al. [52]	200 Gbit/s (theoretical)	Virtex-2 Pro	100 MHz	5-tuple based, high throughput capable architecture based on parallel TCAMs.	Relies heavily upon multiple TCAMs.
StrideBV [53]	> 400 Gbit/s	Virtex-6	N/A	StrideBV: Simple, bit vector-based lookup scheme, no ruleset features.	Performance is independent from rulesets. The idea is an enhancement of FSBV [60].
Qi et al. [54]	> 100 Gbit/s	Virtex-6	115.4-139.1MHz	HyperSplit [36], a memory-efficient, tree-based binary search algorithm.	Optimization algorithms reducing the # of pipeline stages & control the memory usage.
HyPaFilter [55][57]	40 Gbit/s	Virtex-7	180 MHz	A firewall accelerated by FPGA. Combined a fast rule set update mechanism with two matcher algorithms.	MPFC: Massively Parallel Firewall Circuits as specialized & StrideBV as generic matcher.
HyPaFilter+ [58]	92.16 Gbit/s	Virtex-7	180 MHz	HyPaFilter with various optimizations.	Leverage geometric rule representations to significantly reduce the need for SW processing of the majority of the traffic.
Kennedy & Wang [62]	> 200 Gbit/s	Stratix III	433 MHz	Improved version of HyperCuts [37]: modified cutting scheme & rule storage. Wee and Pak [63] further improved it.	Evaluated on multiple engines, can achieve high throughput with low power consumption.
Qu et al. [65]	120 Gbit/s	Virtex-6	235 MHz / 337 MHz	Fast updatable classifier algorithm, based on DBV (Dynamic Bit Vector)	High throughput together with dynamic update of the rule set. Supports OpenFlow.
Khatami & Ahmadi [66]	120 Gbit/s	Virtex-6	384 MHz	Decision tree-based, fast updatable.	N dec. trees are mapped into multi-pipeline (N); Each stage operates on one header field.
Pus et al. [67]	100 Gbit/s	Virtex-5	125 MHz	Classification: 1) longest prefix match; 2) rule set mapping with perfect hashing; 3) checks packet against rule.	Uses perfect hashing for searching. A perfect hash is stored in off-line (SRAM) memory.
Yang et al. [68]	135 Gbit/s	Virtex-5	99.64 MHz - 146 MHz	D^2BS (Dynamic Discrete Bit Selection with preparation & classification phases	Preparation: sub-partitions the orig. ruleset into mem. cells; classification: apply rules.
Hager et al. (SFL) [69]	N/A	N/A	N/A	Fast rule-set modification algo.: uses update buffers and linked list techniques for rule set modification.	SFL increases responsiveness of classification; very good packet processing performance. Platform independent.
Nakahara et al. [70]	640 Gbit/s	Virtex-7	600 MHz	DSP block-based FPGA solution.	Classifier: firtst stage is a modif. HiCuts [71]; the resulting groups are processed by DSPs.
Sun et al. [72]	107 Gbit/s	Virtex-6	N/A	Parallel arch.; intermediate result aggre-gation in decomposition-based designs.	Evaluated up to 32-tuple; performs better than the bit-vector (clock period and resource util).
Orosz et al. (C-GEP) [47]	100 Gbit/s	Virtex-6	312.5 MHz	Parallel, lookup-based Field-Split Bit Vector (FSBV) [60].	14-tuple parser/classifier. Transparent reconfi-guration of rules.
Fong et al. (ParaSplit) [51]	120 Gbit/s	Virtex-5	120 MHz	HyperSplit [36] used for tree setup. Rule set partitioning algo' based on range-point conversion, opt. by Simulated Annealing.	11-tuple, OpenFlow-like rulesets. claims over-Tbit/s capabilities for Virtex-7. Operating with more than 10 ParaSplit instances.
Qu et al. [73][74]	190 Gbit/s	Virtex-6	300 MHz	Two-dimensionally-pipelined architecture, stages can handle range & prefix match.	15-tuple, OpenFlow-like ruleset. Classification is combined with simultaneous rule-update.
Chang & Hsueh [75]	> 400 Gbit/s (> StrideBV)	Virtex-6	380 MHz	Fully pipelined architecture, using StrideBV [53].	12-tuple, OpenFlow-like ruleset. Achieving high throughput, low latency.
Abdulhassan & Ahmadi [76]	> 100 Gbit/s	N/A	N/A	R-tree construction from a ruleset; parallel queries from the tree.	15-tuple, OpenFlow rule classification. Uses less memory than HyperCuts [38].

V. Third Phase: Payload Inspection

After analyzing the packet header, and further, protocol-related information during packet parsing and classification, the payload can be inspected to support further decisions. The main reason of this inspection can determine the preferred approach on how to look for information within the payload. Examples include identification of some malicious content (e.g., virus) with pattern matching (fingerprint analysis), or application identification with flow behavioral analysis. Thus, the complexity of the inspection can range from looking for a typical pattern in independent packets to the stateful analysis of messages, considering protocol-specific dynamics and payload signatures as well.

The taxonomy of these algorithms starts by defining the four main methods: Packet-Based No State (PBNS), Packet-Based per-Flow State (PBFS) Message-Based per-Flow State (MBFS), and Message-Based per-Protocol State (MBPS) [77]. A full taxonomy describing their subtypes, connections between them, as well as comparing their performance (resource requirements, identification capabilities, limitations, failure factors, and constraints of their speed) is an interesting issue, although outside of the scope of this current paper.

1. *Challenges and Solutions for On-the-fly Payload Inspection*

Without fast adaptation, temporal classification, and analysis, errors occur and wrong decisions are made (i.e., malicious traffic get traversed, useful traffic may get dropped). Although the term *payload inspection* is sometimes considered as a synonym [78] of *DPI*, the latter is somehow a wider term, not limited the analysis to any specific part of the packet. Dainotti *et al.* [78] observed properly that the terms are not defined firmly (or used consequently) in the domain of DPI-related research.

There are two main groups of methods exist for payload inspection: signature-based (pattern-matching), and inference-based (machine learning) techniques. Besides the increasing speed requirements of on-the-fly analysis, their main performance-bounds are similar:

- they are the most computationally expensive part of the deep packet inspection toolchain, and
- they require increasing amount of memory as more information is available for the decision-making.

Callado *et al.* [79] compiled a general survey on traffic classification, which is one of the main application areas of DPI. Their study generally noted that signature-based classification is inferior to others for accuracy, although inference-based machine-learning techniques are reported to reach similar accuracy [80] as payload inspection.

The Handbook of Exact String Matching Algorithms by Charras and Lecroq [81] summarizes 35 of the classic algorithms ranging from the Brute Force method (exact checking of patterns, and moving ahead with exactly one position in case of no match) to fast methods such as the Alpha Skip Search algorithm. The preprocessing and searching complexity of each algorithm is presented in this Handbook, as well as example implementation C code. This is a thorough reference, although merely focuses on algorithms and their software implementations – and as such, it does not mention hardware-related acceleration techniques.

Modern signature-based (in other words: pattern matching) algorithms can be traced back to the Aho-Corasick method [82], in which the input is inspected symbol by symbol against a predetermined set of signatures, organized in a structured way within a Deterministic Finite Automaton (DFA) [83]. Before the matching process, the string pattern set is compiled into a DFA. Then the input steam is scanned byte-by-byte by the DFA to find all the matching string patterns in it. During the preprocessing stage, three functions are generated, which are the "goto" function, the "failure" function and the "output" function. The "goto" function outputs the next state given the current state and input character. The "failure" function outputs the next state when the input character cannot continue to match a string pattern from the current state. The "output" function outputs the pattern ID when one of the string patterns has been matched by the current state. An example of a small matching set is shown by Figure 10 [84].

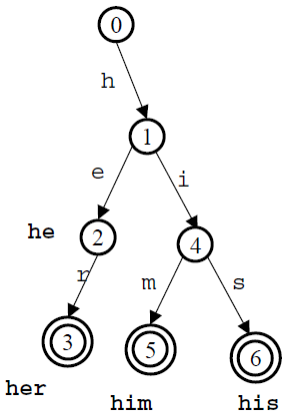


Figure 10. AC algorithm with pattern set {he, her, him, his} [84]

Tuck *et al.* [85] describes various techniques that enhance the worst-case performance of Aho-Corasick algorithm. The idea of their specialized algorithm comes from the analogy between IP lookup and string matching, and applies bitmap and path compression to the Aho-Corasick method.

A comprehensive study of the Aho-Corasick automat and the failure transitions-based implementations, together with their comparison regarding memory footprint and the analysis throughput can be found in [86]. Liu *et al.* describes an effective enhancement of this approach by reducing the memory usage of the DFA with their own compression algorithm, utilizing CMPs (Chip Multi-Processors) [87].

The Boyer-Moore algorithm [90] is also one of the early algorithms and is another widely used algorithm for string matching. It is based on two heuristics: bad-character heuristic and good suffix heuristic. The novel aspect of the Boyer-Moore algorithm and the reason for its effectiveness is that

character matching is performed right-to-left. The bad character heuristic shifts the search string to align the mismatching character with the rightmost position at which the mismatching character appears in the search string. If the mismatch occurs in the middle of the search string, then there is suffix that matches. The good suffix heuristic shifts the search string to the next occurrence of the suffix in the string. The Boyer-Moore algorithm shows a sublinear performance in the average case [91].

One of the biggest challenges of this field is to keep up with the continuous change of application protocol signatures. An automatic method is described in [92], where the aim of analyzing traffic traces is to find the smallest set of signatures for the biggest coverage ratio for a specific application.

2. *Why Decoding Higher Layers is not Easy to Implement in Hardware?*

Before going into the details of hardware-assisted payload inspection, let us discuss the challenges of a relatively simpler task: parsing of multi-encapsulated protocol-headers.

First of all, we distinguish two types of parsing algorithms.

The first one operates with pre-calculated header offsets [25]. The protocol structure is written on a high-level description language [13]. In this case, the starting indices of the protocol fields are known, and these constant values can be calculated in software. The algorithm supports the updating of the parser to include a new protocol, but it always needs intervention.

The second type of packet parsing solution covers a real-life challenge: to detect and decode any unknown protocol combinations. The analysis of complex header structures leads to a more complex implementation, since extracting encapsulated protocol header stacks can result in many additional header-starting offsets. If the encapsulation gets deeper, the calculation of the header-starting bytes gets more and more difficult. Port identifiers from an Ethernet+IPv4+UDP packet can be easily parsed, assuming that the IPv4 header length is fixed to 20 bytes. Contrariwise, if a network packet contains dynamic IPv4 header with its size varying from 20 bytes to 60 bytes, the algorithm has to operate with eleven possible UDP header offsets. In case of IP-in-IP embedding and assuming dynamic IPv4 header length, the number of possible port offsets is 22.

Another challenge of decoding higher layer information by using hardware-based techniques is related to the data path width. If the processing method uses a 512-bit-wide data path, the packet headers may span between data words. This situation can be handled by applying control signals or delayed processing of the cohesive words. Naturally, this problem does not exist when the header structure is known: if the parser works with pre-calculated offsets, the field extraction is not a difficult task. Unfortunately, this is not the case in the practice of general Internet traffic, when the protocol structure of the packet header is unknown.

As the parser moves deeper into the packet, the field extraction becomes more and more difficult. The number of possible encapsulations has a limit, considering the size of the FPGA, the available operating frequency, and the time available for extraction.

This is the reason why parsing behind the transport layer header is difficult to perform effectively in hardware.

There are many trade-offs related to deep parsing of even the packet header, e.g., calculation time and complexity. Software implementation of pattern matching and other behavioral analysis methods is more feasible despite the relatively high processing delay, which, on the other hand can be improved with a distributed design.

A further difficulty comes in beyond the transport layer: frequent changes in the application layer headers and traffic patterns. The changes occur regularly, and it is hard to follow this dynamic behavior with a hardware implementation, since the code modification and firmware update are time-consuming tasks.

After discussing the issues of *multi-encapsulated protocol headers*, it can be argued how challenging is the inspection of the payload, which bears even more irregular features than the header. Still, there are some impressive hardware-based methods that can supplement software-based payload inspection to achieve higher performance.

3. *Hardware-Assisted Payload Inspection*

The purpose of hardware-assisted payload inspection is to reduce the resource requirements of software-based inspections by pre-selecting flows. This is either through successful matching of some inspection rules (hit: no further inspection required), or not yet matching any rule. Successful pre-selection can mean massive reduction of traffic volume forwarded to software-based payload inspection – and reduction of further processing entities.

Aside from complexity issues, payload inspection can be algorithmically very similar to packet filtering. Instead of matching patterns merely in the packet headers, the search is applied throughout the packet.

Nevertheless, complexity of the general payload inspection problem grows exponentially with

- the number of possible rules to be applied,
- the analysis length within a packet, and
- the number of packets to be analyzed within the flows.

As mentioned earlier, depending on the aim of the DPI, hitting accuracy (correlating with analysis complexity) can be somewhat sacrificed for time – in other words, analysis speed.

The natural solution for reducing the analysis complexity is to analyze packet headers only; with a limited number of rules, applicable only to the headers. This is the packet filtering phase. A limited set of rules – that provide the best hitting ratio for the given number of rules – can be chosen for payload inspection, as well. Preferable selection of those rules into this limited ruleset, which are known to be matching in the beginning of the payload, further improves the efficiency of hardware-based payload inspection.

Similar to some advanced software-based payload inspection methods, it can also be a sane decision to inspect merely the first few packets of a flow. Although this method has the drawback of keeping flow-information (e.g., 5-tuples)

in memory, it massively reduces the number of packets to be analyzed, when elephant (high volume) or tortoise (long lasting) flows are present.

Utilizing machine-learning, and inference-based methods are not usable efficiently in hardware-based payload inspection, since they continuously require the update of flow status and statistics, consuming such amount of memory that is not available to be addressed in real-time in these systems.

a. Classic Algorithms and Designs

Still, besides the abovementioned obstacles, there are application areas where hardware-assisted payload inspection comes up as part of the solution. For these areas (with the above mentioned limitations), the general pattern matching methods [81][82][88][89][90][93][94] can be and are implemented in FPGA. Machine-learning techniques are not always efficient to implement in FPGA, however neural networks can be applied for classification (phase II) and pattern recognition (phase III) as discussed in V.3.e.

A classic survey on hardware-assisted pattern matching methods is presented by Fide and Jenks [93]. They found that the Aho-Corasick algorithm and the Boyer-Moore algorithm are used widely, and their implementation is often tailored to better fit the given technology or the proposed architecture. In relation to hardware implementation elements, they list

- CAM-based implementations,
- Bloom-filter-based methods,
- NFA (Non-deterministic Finite Automaton) for regular expression matching, and
- Bitmap and path compression of the fingerprints.

AbuHmed *et al.* [10] provided a hardware-focused survey on DPI for Intrusion Detection Systems. The challenges listed are related to the various issues with signatures, the complexity of the search algorithm, and the encrypted data. The DPI design objectives they list include (i) deterministic performance, (ii) memory efficiency, (iii) dynamic update, (iv) signature handling efficiency, and (v) scalability, besides (vi) additional functions. It is important to note that FPGAs have clear advantage in (i) and (iii) when compared to network processors or GPUs. When they compared the existing architectures, the superiority of FPGA designs over the others was very clear.

A more recent survey by Ahmed and Khare [91] describes the hardware implementations of various pattern-matching methods, and compares them by theoretical search times. These include the Brute Force method, the Knuth-Morris-Pratt (KMP) algorithm [81], the Aho-Corasick algorithm and the FM-index [94] (a data structure that allows compression of the input text together with permitting fast substring queries).

b. Advanced Design Examples for FPGA-assisted Payload Inspection

The first experiments for implementing regular expressions in custom hardware were presented by Floyd and Ullman [95]. Their paper showed that an NFA can be efficiently implemented using a programmable logic array. An NFA do not have to obey certain rules of state machines, hence their transitions are *not* uniquely determined by their source state and input symbols, and reading an input symbol is *not* required for each state transition.

The Content Addressable Memory (CAM) approach is used widely for pattern matching – through switches, routers, and DPI solutions, as well. The internals of various scalable pattern-matching methods based on Decoded partial CAM (DpCAM) and Perfect Hashing memory (PHmem) approaches) applied in FPGAs for Multi-Gigabit Ethernet payload inspection is described in [96].

Dhanapriya and Vasanthanayaki [97] present the Word Split Hash algorithm (WSHA), which is an optimized hash-based algorithm, to compare payload against attack patterns. This method is an alternative to the DpCAM and PHmem approaches, and is verified through VHDL simulations, hence clearly feasible to be implemented in FPGA.

Wang *et al.* [98] describe a pattern matching algorithm based on a skip counting automata, implemented through a three-state CAM (TCAM). Besides the two states of traditional CAMs (0 and 1), TCAMs work with a third state X (don't care) as well. This helps increasing effectiveness when the TCAM is used for regular expression-based pattern matching.

In the first FPGA-based NFA (Nondeterministic Finite Automaton) implementation, Sidhu and Prasanna [99] aimed to minimize the time and space required for the pattern matching task. Furthermore, Clark *et al.* [100] managed to show that very good space efficiency can be achieved for regular expression matching. This was the first published FPGA-design that covers all pattern matching operations of Snort [101] at the time (over 1500 rules; which grows over 50000 nowadays). This showed that using FPGAs are feasible for pattern matching in that scale.

Both NFAs and DFAs within the FPGA are built as pipelines of character matching units. While NFAs are chosen for shorter time and smaller space requirements for *constructing* the automaton, DFAs are preferable when minimizing the size of the *completed* automaton. Beside this, Moscola *et al.* [102] demonstrated that using DFAs perform significantly better than NFAs in throughput for stream-by-stream, and multi-context searching tasks.

Becchi and Crowley [103] proposed a hybrid automaton, aiming to utilize the advantages of DFAs – that perform regular matching in linear time – and NFAs – that have lower number of states, hence requires less resource for a similar implementation. Their algorithm aims to transform an NFA into a DFA while keeping its size at the minimum through interrupting the subset construction operation at those NFA states whose expansion would cause state explosion to happen. The resulting hybrid-FA has a memory storage requirement comparable to those of an NFA solution, and an average case memory bandwidth (speed) requirement similar to that of a single DFA solution.

Kumar *et al.* [104] introduced a new representation for regular expressions, which they call the delayed input DFA, (D^2FA). Its key characteristic is the significant reduction of space requirements for a DFA by replacing its multiple transitions with a single default transition. After showing that

the construction of an efficient D^{2}FA from DFA is NP-hard, they present heuristics for D^{2}FA construction that provide deterministic performance guarantees. The resulting D^{2}FA has a very small memory requirement, 95% lower than the DFA.

Ficara *et al.* [105] focused on the memory savings of DFAs as well, and developed the main idea of D^{2}FA further. They introduced a novel compact representation scheme (named δFA), which is based on the observation that, since most adjacent states share several common transitions, it is possible to delete most of them by taking into account the different ones only. The δ in δFA denotes the differences between adjacent states. It requires only a state transition per character (like other DFAs), thus allowing fast string matching.

Yusuf *et al.* [107] compared CAM, DFA and NFA based implementations of the pattern matching challenge to their BCAM (Binary Decision Diagram-based CAM) that uses bit-level resource sharing. They present a comprehensive comparison of these implementations, focusing on space (lookup-table) efficiency and performance. Their solution (in a multiple-pipelined implementation) reached a throughput of 12-14Gbit/s by using 70680 lookup-tables at a Virtex-2 FPGA, which is significantly more efficient than the other, listed solutions [108][109][110][111][112].

The Extended Finite Automaton (XFA), proposed by Smith *et al.* [106] is somewhat orthogonal to the above solutions, since it augments finite state automata with finite scratch memory and instructions to manipulate this memory. They show that for a large class of IDS signatures, XFAs show similar performance in speed as DFAs do, and has somewhat smaller resource requirements than NFAs.

Selvaraj and Ganapathi [113] use regular pattern matching DFA (Deterministic Finite Automaton), and propose to compress DFA states in order to better fit into memory, which is a scarce resource. The processing time of DFA is proportional to the length of the input string, but the memory cost of a DFA is quite large. In order to decrease this cost, various compression methods can be used, among which they found the Delayed Dictionary Compression (DDC) seemed the most promising. Their experimental results meet their expectations: the proposed method gives better memory utilization and time computation results for payload pattern matching – in their example: Internet worm detection.

Wang thoroughly details the Stride-based payload inspection method in [84]. The dissertation argues that traditional DFA-based DPI solutions cannot be used for inspection of file distribution in P2P networks due to the potential out-of-order manner of the data delivery. To address this problem, a hybrid finite automaton called Skip-Stride-Neighbor Finite Automaton (S^{2}NFA) is proposed. Instead of comparing the input stream character by character with patterns in the rule set (like in a normal DFA), this Stride-based DFA method picks tag characters from the input stream and feeds the "fingerprint" of these tag characters to the automaton for the matching examination. Since the fingerprint is normally much shorter than the original input stream, the number of state visits required by the matching process can be significantly reduced.

c. *Further Hardware-assisted Payload Inspection Solutions*

Graphical Processing Units are reaching remarkably great performance recently, making them a potential target platform for pattern matching algorithms. Hung *et al.* [114] presents an efficient GPU-based multiple pattern matching algorithm for packet filtering, whereas Lin *et al.* [115] describes an architecture utilizing CPUs and GPUs, implementing a Length-bounded Hybrid Pattern-Matching Algorithm (LHPMA) for DPI. The LHPMA first categorizes the packet according to a pre-determined payload length bound. The packets whose lengths exceed the bound are delivered to the CPU for rapid prefiltering; otherwise, the packets are assigned directly to the GPU for full pattern matching. The authors show that better performance can be obtained when reducing the payload length diversity.

Clustering algorithms mainly aim for traffic classification, and use-pre-processed packet- or flow-information that are provided by the earlier mentioned, hardware-assisted packet parsing and filtering methods.

Bakhshi and Ghita [116] proposed a two-phased machine learning classification mechanism, with Netflow data as input. The individual flow classes are derived per application through k-means clustering as the first phase of the mechanism. The derived flow classes are then used to train and test a supervised C5.0 based decision tree. Although the accuracy of this method is well above 96% in the reported cases, its processing time is in the range of several minutes. It competes well with alternative methods such as Bayes networks or decision tables, but these machine learning methods are not yet feasible for (quasi-)real-time decision making.

Another k-means clustering method is described by Du and Zhang [117] for which they presented an advanced flow-extraction method. Instead of using Netflow, they identify the 3way TCP handshake, and the TCP FIN/RST of the otherwise encrypted traffic, and define the flows based on this extracted knowledge. Applying the k-means clustering for traffic classification provides a good hit-ratio, although this is far from real time operation.

The stochastic packet inspection method KISS [118] targets the UDP traffic, and aims to classify that. The method uses statistical signatures that are automatically inferred from training data, by the means of a Chi-Square like test, which extracts the protocol "syntax", but ignores the protocol semantic and synchronization rules. The signatures feed a decision engine based on Support Vector Machines. The overall method has a reported accuracy around 99% for the chosen traffic classes; although there are no reports available on calculation time.

Hullar *et al.* [119] showed that effective classification of P2P traffic can be performed based on the first few bytes of the first packet of each flow. The proposed classifiers are based on standard stochastic models and state-of-the-art machine learning methods (such as Random Forests, Markov-models or Context Trees) and can reach a remarkable accuracy over 95% using as limited data as the first 16 bytes of the first

(or the first backward) packet of each flow.

Another large problem space is revealed when pattern matching is not an option, because the traffic is encrypted. It is no use of encrypting malicious traffic that targets Layer 3 or Layer 4, hence pattern matching techniques can be used in these cases. On the other hand, detecting attacks that are related to Layer 7, or classify traffic for its applications require other methods. Velan *et al.* compiled a comprehensive survey of methods for encrypted traffic classification and analysis in [120].

Neural networks are another approach for payload inspection. Besides providing a very good list of packet-based anomaly detection approaches (from support vector machines through Markov models, fuzzy networks and Bayesian networks to gravitational search algorithms (GSAs)) Sheikhan and Jadidi present their own approach with GSA-optimized neural networks [121].

There are a large number of system architectures that implement payload inspection in software, and use the hardware merely for packet- or flow-based filtering and sampling. A good example of this is TOPAS [122], which utilizes PSAMP [123] for packet sampling, Netflow for flow-sampling, and Snort for software-based payload inspection. Such architectures are quite widespread recently; although not discussed further here, since these are not equipped with hardware-assisted payload inspection.

d. Bloom Filters

The Bloom filter is a probabilistic data structure with the aim of testing a data x being member of a given set of elements n. It can be used in the packet classification as well as the payload inspection phases of the DPI processing chain. The main benefit of its application is that a large set of classification rules can be stored and accessed in a very compact form within the FPGA hardware, where memory capacity is likely a resource constraint.

Since it is a probabilistic structure, the main advantages over the common hash tables or binary trees are the fixed size and the constant query and insertion time that is independent on the number of elements within the structure. The Bloom filter provides a trade-off between effectiveness in size and the rate of false positive answers for queries, i.e., an element is reported to be part of the set despite it is actually not.

Let us consider a bit vector of size m. Initially all bits are set to zero. k independent $h(x)$ hash functions are defined for querying and inserting elements in set n (see Figure 11). If we are considering an input x, each hash function $h(x)$ generates an index value i, where $0 \le i < n$.

Index i maps to a position within the bit array m. According to the number of hash functions, k index values are generated. During the insertion procedure, all of the k indices are set to 1 within the setup. Querying for an element implies the same hashing procedure, i.e., applying the same k hash functions for the input x.

The resulted k index values now represent bit positions that are checked whether they were previously set to 1. Any calculated position holding zero value implies that input x is a new element and was not inserted previously.

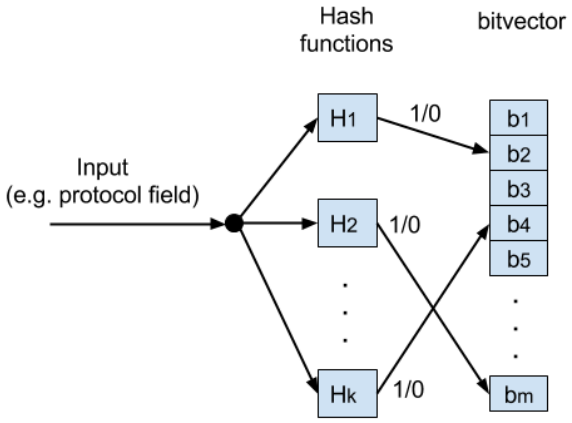


Figure 11. A Bloom-filter model

Otherwise x is an existing element of m. In order to minimize the false positive rate, determining the optimal size of m and the optimal number of hash functions is a key design step. The false positive rate of the filter can be approximated by (1).

$$\text{fp} = (1 - e^{-\frac{k \times n}{m}})^k, \quad (1)$$

where k is the number of hash functions, m is the size of the bit array and n is the number of elements stored in m.

In the last decade, applying Bloom filters in many fields of network communications was a common alternative when on-chip memory size was a limitation or frequent accessing of an off-chip memory resulted in high latency.

Prya *et al.* present a combined hierarchical approach based on an all-length Bloom filter for the source prefix field and an H-trie data structure for the destination prefix field [124]. In this method the Bloom filter, as a pre-filter, is employed to filter out sub-strings of the source field, which have no matches in the set. At the second stage, for the sub-strings with matches found by the pre-filter, an H-trie-based filtering is applied for the destination field in order to find the highest priority match within the rule set. The Bloom filter requires a small amount of memory and therefore can be implemented using on-chip memory as well. This makes it an optimal method to be applied inside a reconfigurable hardware such as FPGA. Simulation results showed that the proposed method provides higher search performance in average and worst cases, and requires lower memory. The novelty of the applied Bloom filter is that it accommodates to different lengths of source prefixes within one single filter instance and besides the search result, it also provides a pointer to the source prefix in the H-trie.

Lim *et al.* extend the tuple pruning algorithm for traffic filtering [125]. The proposal replaces individual field lookups with a Bloom filter and thus it reduces the search space and therefore the number of off-chip memory accesses. Common packet classification algorithms (i.e., tuple space pruning and cross-producting) perform individual off-chip memory lookups for each protocol field to decrease the search space. Authors propose a method to replace each field lookup with an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/COMST.2018.2876196, IEEE Communications Surveys & Tutorials

on-chip (source or destination) Bloom filter to reduce the number of unnecessary off-chip memory accesses. In this design, hash functions must adapt to prefixes of any length as input within a single filter. Authors apply cyclic redundancy check (CRC) generators to accommodate to variable input size.

Dharmapurikar *et al.* [126] replace the TCAM-based traffic classification with an optimized algorithmic approach. In this proposal the common crossproduct algorithm had been modified and extended with internal Bloom filters to reduce external memory consumption as well as the number of memory lookups. Using two SRAM chips, the performance of the system is 38 Mpps.

Ahmandi *et al.* [127] introduced a *k*-stage pipelined Bloom filter to improve power efficiency. They analyzed different traffic traces. The authors showed that in a 4-stage pipelined Bloom filter, 75% of the incoming mismatched packets are detected by the first three stages. They concluded that power consumption can be optimized by utilizing only one hashing functions for each of the first three stages, while the last stage operates with $k-3$ parallel hashing functions. The main benefit of the pipeline architecture, in terms of power consumption, is that the lookup process will stop as soon as the first mismatch reached. In this case there is no need to pass the packet to the next stages since the input item is not a member of the bit array.

e. Neural Networks

The Neural Network or Artificial Neural Network (ANN) is a widespread model in several research areas for classification and pattern recognition tasks. The principal idea of the method is to use a simple processing unit for a simple task, and to connect the units to each other in a flexible – hence potentially very complex way. This results in a complex, parallel-processing network architecture for complex calculations.

The basic processing unit in a standard neural network is known as the neuron. According to its functionalities, a neuron can be represented as an adder-multiplier element. In a neural network architecture, the neurons are organized in layers (Figure 12), which are connected in a pipeline model (e.g., FFNN – Feed-Forward Neural Network). The number of layers and the number of neurons within a layer defines the model. Each neuron uses a weight to calculate the response to its input parameters.

Constructing a neural network is a three-step method. First of all, the number of layers and neurons have to be defined for the given problem space. The second step is to apply a learning algorithm on the model, to calculate the weight of each processing unit. After the training procedure, the third step is the test phase with a sample data set. The sample data set is often a subset of the training data set.

Artificial Neural Networks often use transfer functions (e.g., log-sigmoid function) between the network layers. These transfer functions convert the dataset into a given range (e.g., [0,1], [-0,1]).

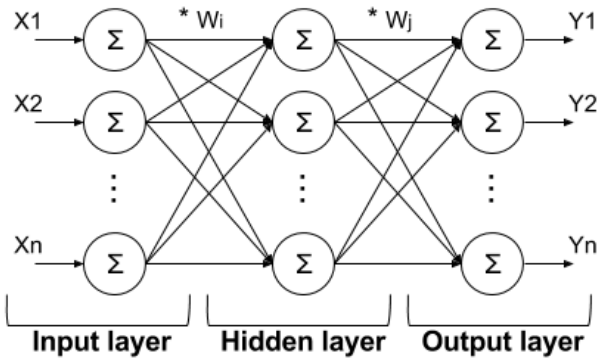


Figure 12. Feed-Forward Neural Network model (X_i denotes the i^{th} input item, W_i and W_j are the weights of the network and Y_i is the i_{th} output item)

The training phase is a major, highly important procedure, because the weights (W_i, … W_j in Figure 12) of the network are specified at this phase. We differentiate supervised and unsupervised learning algorithms. The training procedure is supervised, if the learning data set contains the input parameters and the expected output too (e.g., Levenberg-Marquardt [128] [129], Bayesian Regularization [130] [131]), otherwise it is an unsupervised procedure.

Based on the structure of the layers, we also differentiate two types of networks:

1) Feed-forward network: This is a pipeline structure, in which the same input parameter results the same output.

2) Backpropagation: The output of the network also depends on the previous state.

A special kind of neural network is the autoencoder [132], which has the same number of neurons in the input and in the output layers. This solution approximates the input parameters through dimensionality reduction, using one or more hidden layers and transfer functions. The autoencoder aims to learn the compressed form of the multidimensional input data. An autoencoder-based system often consists of a decoder part, which restores the original input from the neural network output, using further transfer functions (with further weights and bias values). The autoencoder is also known as an unsupervised learning algorithm, which can be used for pretraining of a supervised data set. Although it applies lossy compression, it can be effectively used for cluster analysis and outlier detection. In case of a network Intrusion Detection System, it can be used for anomaly detection. Other research areas include image processing, speech recognition and noise reduction, next to the general dimension reduction capability.

In recent years, several neural network models were published in the packet classification and in the network traffic detection research areas. Some of these solutions [133][134][135] are based on software implementations and offline test results. Auld *et al.* [133] proposed a Bayesian trained network, using hundreds of input parameters (e.g., packet count, variance of the payload size, bandwidth) for training. The solution focuses on TCP flows and uses tcptrace [136] for the flow feature calculation. Using 248 parameters, the training time takes almost 40 hours, and the average accuracy of the classification reaches 95% (99% for the data tested and trained on the same day). To reduce the training

time, the proposed work also showed that the minimum number of parameters is 128 to reach 95% accuracy.

Using low number of input parameters or hidden-layer neurons, the training time could be reduced. Zhou *et al.* [134] proposed a 10-30 neuron-based network and showed that the difference of the accuracy when using 30 and 10 neurons is not considerable. The authors also compared the Naive Bayesian method with the Feed-Forward Neural Network model. The results show that the latter method is more efficient for traffic classification tasks.

Since the real-time calculation of many hundreds of parameters is difficult, some of the research groups are used only few parameters for the arbitration. Research shows that the size of the first few packets in a flow lifetime carries enough information for classification tasks [137].

Zelina *et al.* [138] proposed a packet-size based classification model. It uses early detection: classify the flow based on the first few packets. Authors use the first 6 packets of the flow to train the NN model, and classify 5 applications (SSH, Skype, HTTP, POP3, Bittorrent) with an average 60-70% accuracy.

Peng *et al.* [139] proposed HRBF (Hierarchical Radial Basis Function network) model, an effective early stage traffic identification algorithm. RBF is a kind of feed-forward network, organized in a cascaded (hierarchical) scheme. The authors used the packet sizes, and statistics from the packet sizes for the training procedure. The hierarchical tree structure results high (95-98%) accuracy on the test samples.

Using packet classification in routing and forwarding degrades the problem space to 5-tuple [140], or just to IP lookup. Moallem *et al.* [141] demonstrates a 4.5 ns IP lookup technique using a 12-layer neural network scheme. The parallel, pipelined architecture is designed for hardware, operating at even 400 Gbit/s throughput. The network uses backpropagation to learn new IP addresses.

The determinative part of the NN-based classification methods focuses on P2P traffic identification. Peng *et al.* [142] proposed a Web and P2P traffic identification scheme, using flexible neural trees (FNT). This is a special kind of ANN, in which the leaf nodes are inputs, and the other nodes are neurons. The recognition results in 86% average accuracy, which shows better decision rate than RBFNN (Radial Basis Function Neural Network).

Sun *et al.* [143] proposed a nonlinear pattern recognition model, probabilistic neural network (PNN). The architecture does not need a training procedure, in which the hidden layer compares the input parameters with probabilities. The work results in 91% accuracy for P2P, and 98% accuracy for Web traffic identification. The average accuracy is higher than the RBFNN model.

Jingquan *et al.* [144] proposed a Bayesian Regularization based NN, using 20 flow parameters (e.g., TCP/IP control field size, transport layer features). The network classifies between two output categories: P2P and non-P2P. After the offline training, the software-based simulation shows 100% recognition accuracy.

Yiran *et al.* [145] proposed a back-propagation algorithm combined with LVQ NN. The input parameters are selected from the TCP and UDP packets of a given P2P application, and the research results 95% identification precision.

Fuke *et al.* [146] also proposed a backpropagation network scheme for P2P or non-P2P class detection. The input parameters of the software-based simulation are: mean squared deviation, switching frequency and average rates of the flow's packet size, next to the packet quantity and total bytes of the flow. The discovery rate is between 93-99%, applying on encrypted and unknown P2P traffic, as well.

Table III provides an overview of the surveyed hardware-based or hardware-optimized payload inspection methods. There are numerous further implementations of the classical methods summarized in the beginning of this chapter, although these mainly differ in device-specific implementation (rather than base theories or methods), hence do not appear in the table.

f. Lessons Learned in Payload Inspection

The novel methods and implementations for hardware-accelerated payload inspection utilize either TCAMs, or FPGAs, or both. The ultimate instrument for payload inspection should work with close-to 100% accuracy, it should be capable of applying ever-growing number of rules, it is optimized for memory, and its decision time is being kept low. Such a complex requirement is hard to meet, although its elements can be addressed individually and the resulting methods can be combined to achieve certain targets. As new, malicious patterns appear and get shared by the community daily, the rule-sets for matching needs to be flexibly modifiable. The rule descriptors are optimized for better memory consumption and faster decision-making, although this optimization takes some time, as well.

Besides summarizing the various novel methods, Table III points out the novelties and advantages of each method, as well. As current best practice suggests, heterogeneous systems with co-existing hardware accelerated payload inspection and software-based solutions satisfy current needs the best. We suggest the following papers as fundamental literature in the field of payload inspection: [79][91][93][100][103][104][106].

Table III – An overview of advanced, HW accelerated payload inspection solutions – excluding the numerous classic methods

Reference	Platform	Method	Notes
Sourdis et al. [96]	HW	CAM-based pattern-matching methods (e.g. DpCAM, PHmem) applied for FPGA.	Scalable; multi-Gigabit payload inspection.
Dhanapriya and Vasanthanayaki [97]	HW	Word Split Hash algorithm (WSHA) – an optimized, hash-based algorithm, implemented in FPGA.	Compares to DpCAM and PHmem.

Wang et al. [98]	HW	Skip counting automata, implemented through a three-state CAM (TCAM).	Pattern matching with FPGA and TCAM.
Sidhu and Prasanna [99]	HW	The first FPGA-based implementation of pattern matching that uses Non-Deterministic Finite Automaton (NFA). Wildcards are zero or more characters.	The implementation's performance compares well with CPU-based GNU cgrep.
Clark et al. [100]	HW	Improved NFA-based pattern matching; the length of wildcards are specified by either a lower bound or an upper bound.	Implementing all Snort [101] rules at the time, and matching against each rule.
Moscola et al. [102]	HW	Using Deterministic Finite Automaton (DFA) in order to decrease the size of the completed automaton.	DFAs perform better than NFAs for stream-by-stream and multi-context searching tasks.
Becchi and Crowley [103]	SW	Combines NFA and DFA advantages. Transforms NFA into a DFA while keeps its size at the minimum through interrupting subset construction.	Hybrid-FA: memory storage is comparable to NFA, speed is comparable to DFA.
Kumar et al. [104]	HW/SW	Proposes D^2FA (Delayed Input DFA) that requires 95% less memory than DFA.	DFA to D^2FA construction needs heuristics.
Ficara et al. [105]	HW/SW	Suggests δFA, that is based on deleting the common transitions of adjacent states.	Compact, but also as fast as other DFAs.
Yusuf et al. [107]	HW	Introduces the BCAM (Binary Decision Diagram-based CAM) that uses bit-level resource sharing. It has a Multiple-Pipelined implementation, as well (MPBCAM).	MPBCAM performs significantly better than some DFA, NFA, or CAM solutions.
Smith et al. [106]	HW/SW	Proposes XFA (Extended Finite Automaton), that augments finite state automata with finite scratch memory and instructions to manipulate this memory.	In speed, it is similar to DFAs, but consumes slightly less resources than NFAs.
Selvaraj & Ganapathi [113]	SW	Uses Deterministic Finite Automaton with compressed states, through Delayed Dictionary Compression.	Gives better memory util. & time computation results for payload pattern matching.
Wang [84]	HW/SW	Skip-Stride Neighbour Finite Automaton – picks tag characters from input stream & passes the "fingerprint" of these tag chars for the matching examination.	Short fingerprints: # of state visits required by the matching process is significantly reduced.
Hung et al. [114]	HW	GPU-based multiple pattern matching algorithm for filtering malicious packets by using a Bloom filter.	The proposed algorithm significantly enhances performance over sequential algorithms.
Lin et al. [115]	Hybrid	Utilizing CPUs and GPUs, implementing a length-bounded hybrid pattern-matching algorithm (LHMPA).	Pattern matching on shorter packets by GPU, longer packets are prefiltered by CPU.
Bakhshi & Ghita [116]	SW	Two-phased machine learning classification mechanism with Netflow input. Phases: 1) Flow classes by k-means, 2) train & test a supervised decision tree.	High accuracy (95%), but not yet feasible for (quasi-) real-time operation.
Du & Zhang [117]	SW	Flows defined by 3-way TCP-handshake; then classification through k-means clustering.	Motivation: more & more traffic is encrypted. Far from real-time.
KISS [118]	SW	Stochastic packet inspection method with a Chi-Square like test, for UDP traffic classification. Signatures are fed into a support-vector based decision engine.	Very high accuracy (99%); no reports on calculation time.
Hullar et al. [119]	SW	Apply stochastic models & machine learning methods (Random Forests, Markov-models or Context Trees) on the first 16 bytes of the 1st packet of the flow.	Effective classification over 95% accuracy; works on encrypted traffic.
Velan et al. [120]	-	Comprehensive survey on methods for encrypted traffic classification & analysis.	Classification done w/o pattern matching.
Sheikhan & Jadidi [121]	SW	Gravitational search algorithm (GSAs) GSA-optimized neural networks. Effective to monitor abnormal traffic flows gigabytes traffic environment.	Accuracy is about 97.8%.
TOPAS [122]	SW	Utilizes PSAMP [123] for packet sampling, Netflow for flow-sampling, and Snort [101] for software-based payload inspection.	A typical heterogeneous system without the application of HW acceleration.

VI. Conclusion

After splitting the Deep Packet Inspection tasks into three phases – packet parsing, packet classification and payload inspection –, we aimed at listing advantages and disadvantages of hardware and software-based approaches. Besides the detailed presentation of related proposals and scientific results, we highlighted and explained the best practices for each DPI phase.

Once we accept the limitations of the physical resources, two factors to trade-off are time versus computational complexity – the latter has close correlation with hitting accuracy. Software-based solutions have the advantage of being able to inspect the packet payload deeply, comparing against vast number of rules. This also scale relatively well, by using cheap CPU and memory units, hence allowing easy parallelization. On the other hand, time-related requirements imply non-CPU-based hardware assistance.

After discussing trade-off issues for operating frequency, internal bus sizes, memory availability, on-chip physical space, on-chip parallelization, and computational depth, we showed practical approaches for supporting software-based DPI with dedicated hardware.

The real time criterion for packet parsing and classification can be met by hardware even at 400 Gbit/s line-rate with current FPGAs, hence we suggest these phases of DPI to be supported by dedicated hardware. Although packet parsing and classification tasks require a smaller ruleset than payload inspection, the latter can also be accelerated through hardware. In case the rule-set for hardware-based payload inspection is tuned for a high hit-rate (with still low computational complexity), there are less flows and packets left for software-based payload inspection. This way the required resources can be reduced – less parallel operating machines, less power consumption, less occupied space – hence the operational cost of the DPI system can be significantly dropped.

VII. Acknowledgements

We would like to express our sincere gratitude to the editors and the anonymous reviewers of IEEE Communications Surveys and Tutorials for their thorough review, and insightful suggestions.

References

[1] C. Xu, S. Chen, J. Su, S. M. Yiu, L. C. K. Hui, "A Survey on Regular Expression Matching for Deep Packet Inspection: Applications, Algorithms, and Hardware Platforms," IEEE Communications Surverys & Tutorials vol. 18, 2016, pp. 2991-3029.

[2] S. Alkateb, "5 Things You Need to Know about Deep Packet Inspection (DPI)," white paper, Cavium Networks, 2011

[3] Napatech, "Building Intelligent Mobile Data Services Using Deep Packet Instpection," white paper, 2011

[4] Radisys, "DPI: Deep Packet Inspection Motivations, Technology, and Approaches for Improving Broadband Service Provider ROI," white paper, 2010

[5] Xilinx, "What is an FPGA?," https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html,viewed on 19/02/2018

[6] NXP, "C-5 Network Processor - Silicon Revision D0," https://www.nxp.com/docs/en/data-sheet/C5NPD0-DS.pdf, viewed on 19/02/2018

[7] Nvidia, "What is GPU-accelerated Computing?," http://www.nvidia.com/object/what-is-gpu-computing.html,viewed on 19/02/2018

[8] K. Septinus, P. Pirsch, H. Blume, U. Mayer, "A fully programmable FSM-based Processing Engine for Gigabytes/s header parsing," in Proc. International Conference on Embedded Computer Systems, 2010, Samos, Greece

[9] M. Avalle, F. Risso, R. Sisto, "Scalable Algorithms for NFA Multi-Striding and NFA-Based Deep Packet Inspection on GPUs," IEEE/ACM Transaction on Networking vol. 24, 2016, pp. 1704-1717.

[10] T. AbuHmed, A. Mohaisen, D. Nyang, "A Survey on Deep Packet Inspection for Intrusion Detection Systems," Magazine of Korea Telecommunication Society, 24 (11), 2007, pp. 25-36.

[11] L. Wei, L. Hongyu, Z. Xiaoliang, "A network data security analysis method based on DPI technology," in Proc. 7th IEEE International Conference on Software Engineering and Service Science, 2016, Beijing, China

[12] P. Kobiersky, J. Korenek, L. Polack, "Packet Header Analysis and Field Extraction for Multigigabit Networks," in Proc. 12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2009, Liberec, pp. 96-101.

[13] M. Attig, G. Brebner, "400 Gb/s Programmable Packet Parsing on a Single FPGA," in Proc. ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems, 2011, pp. 12-23.

[14] G. Brebner, W. Jiang, "High-Speed Packet Processing using Reconfigurable Computing," IEEE Micro, vol. 34, 2014, pp. 8-18.

[15] V. Moreno, J. Ramos, P. M. Santiago del Río, J. L. García-Dorado, F. J. Gomez-Arribas and J. Aracil, "Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1364-1390, thirdquarter 2015.

[16] T. Tóthfalusi, L. Kovács, P. Orosz, P. Varga, "100 Gbit/s network monitoring with on-the-fly reconfigurable rules for multi-encapsulated packets," in Proc. IEEE 16th International Conference on High Performance Switching and Rouging (HPSR), 2015.

[17] P. Varga, L. Kovács, T. Tóthfalusi, P. Orosz, "C-GEP: 100 Gbit/s capable, FPGA-based, reconfigurable networking equipment," in Proc. IEEE 16th International Conference on High Performance Switching and Routing (HPSR), 2015.

[18] HiTech Global Inc. 100 Gbps Ethernet MAC IP Core, [Online], http://www.hitechglobal.com/IPCores/100GigEthernet-MAC-PCS.htm, viewed on 20/07/2018

[19] Intel Altera 100 Gbps Ethernet MAC IP Core, [Online], https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-ll100gb-ethernet.html, viewed on 20/07/2018

[20] Mantaro 100 Gbps Ethernet MAC IP Core, [Online], http://www.mantaro.com/products/fpga-ip-cores.htm, viewed on 20/07/2018

[21] Hiteksys 100 Gbps Ethernet MAC IP Core, [Online], http://hiteksys.com/products/fpga-ip-cores/100g-ethernet, viewed on 20/07/2018

[22] IEEE 802.3bs-2017, "IEEE Standard for Ethernet - Amendment 10: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gb/s and 400 Gb/s Operation," http://ieeexplore.ieee.org/document/8207825/, viewed on 19/02/2018

[23] Xilinx Inc., "400G High Speed Ethernet Subsystem (400G HSEC)," https://www.xilinx.com/products/intellectual-property/em-di-400gemac.html, viewed on 19/02/2018

[24] Precise-ITC Inc., "400G Ethernet CDMAC IP core," http://www2.precise-itc.com/x6/index.php/400ge-mac/, viewed on 19/02/2018

[25] C. Kozanitis, J. Huber, S. Singh, G. Varghese, "Leaping multiple headers in a single bound: wire-speed parsing using the Kangaroo system," in Proc. 29th IEEE Conference on Computer Communications, 2010, San Diego, CA USA, pp. 830-838.

[26] NetFPGA, "The NetFPGA is:," http://www.netfpga.org, viewed on 19/02/2018

[27] G. Gibb, G. Varghese, M. Horowitz, N. McKeown, "Design Principles for Packet Parser," ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2013, San Jose, CA, USA, pp. 13-24.

[28] T. Duan, J. Shen, P. Wang, S. Liu, "A Self-Adaptive Programming Mechanism for Reconfigurable Parsing and Processing," China Communications vol. 13, 2016, pp. 87-97.

[29] V. Pus, L. Kekely, J. Korenek, "Low-Latency Modular Packet Header Parser for FPGA," in Proc. 9th ACM/IEEE Symposium on Architecture for Networking and Communications Systems, 2012, Austin, Texas, USA, pp. 77-78.

[30] P. Benácek, V. Pus, H. Kubátová, "P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers," in Proc. IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines, 2016.

[31] P. Benácek, V. Pus, J. Korenak, M. Kekely, "Line rate programmable packet processing in 100Gb networks," in Proc. 27th International

Conference on Field Programmable Logic and Applications, 2017, Ghent, Belgium

[32] X. L. Hu, X. R. Wang, L. Wang, H. Li, "Design of extensible forwarding element architecture and its key technology verification," in Proc. International Conference on Integrated Circuits and Microsystems, 2016, Chengdu, China

[33] J. Li, Z. Sun. B. Han, "P5: Programmable Parsers with Packet-level Parallel Processing for FPGA-based Switches," in Proc. ACM/IEEE Symposium on Architectures for Network and Communications Systems, 2017, Beijing, China

[34] V. Pus, L. Kekely, J. Korenek, "Design Methodology of Configurable High Performance Packet Parser for FPGA," in Proc. 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2014, Warsaw, Poland

[35] J. S. da Silva, F-R. Boyer, J. M. P. Langlois, "P4-compatible High-level Synthesis of Low Latency 100 Gb/s Streaming Packet Parsers in FPGAs," to appear in Proc. 26th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'18), February 2018, Monterey, California USA

[36] Y. Qi, L. Xu, B. Yang, Y. Xue, J. Li, "Packet Classification Algorithms: From Theory to Practice," in Proc. INFOCOM, 2009, Rio de Janeiro, pp. 648-656.

[37] P. Gupta, N. McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," in Proc. 7th IEEE Symposium on High Performance Interconnects, Stanford, CA, USA, 1999.

[38] S. Singh, F. Baboescu, G. Varghese, J. Wang, "Packet Classification Using Multidimensional Cutting," in Proc. SIGCOMM'03 Applications, technologies, architectures, and protocols for computer communications, 2003, New York, USA, pp. 213-224.

[39] C. L. Hsieh and N. Weng, "Many-field packet classification for software-defined networking switches," *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Santa Clara, CA, 2016, pp. 13-24.

[40] Y. K. Chang, C. Lee, C. Su, "Multi-Field Range Encoding for Packet Classification in TCAM," in Proc. INFOCOM, 2011, Shanghai, pp. 196-200.

[41] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplam, E. Porat, "On Finding an Optimal TCAM Encoding Scheme for Packet Classification," in Proc. INFOCOM, 2013, Turin, pp. 2049-2057.

[42] Y. Sun, M. S. Kim, "Tree-Based Minimization of TCAM Entries for Packet Classification," in Proc. 7th IEEE Symposium on Consumer Communications and Networking Conference, 2010, Las Vegas, pp. 1-5.

[43] Y. Cheng, P. Wang, "Scalable Multi-Match Packet Classification Using TCAM and SRAM," IEEE Transactions on Computers, vol. 65, 2016, pp. 2257-2269.

[44] X. Li, Y. Lin, "TaPaC: A TCAM-Assisted Algorithmic Packet Classification with Bounded Worst-Case Performance," IEEE Global Communications Conference, 2016.

[45] S. Hager, P. John, A. Fiessler, B. Scheuermann, "Minflate: Combining rule set minimization with jump-based expansion for fast packet classification," in Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2016, Santa Clara, CA, USA

[46] S. Zheng, X. Bi, J. Luo, "An Efficient Total Prefix Length-Based Clustering Packet Classification Algorithm," in Proc. International Conference on Network and Information Systems for Computers, 2016, Wuhan, China

[47] P. Orosz, T. Tóthfalusi, P. Varga, "C-GEP: Adaptive network management with reconfigurable hardware," in Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015.

[48] D. E. Taylor, J. S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels," in Proc. INFOCOM, 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005, pp. 269-280.

[49] A. Kennedy, X. Wang, Z. Liu, B. Liu, "Low Power Architecture for High Speed Packet Classification," in Proc. 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, November 6-7, 2008, San Jose, CA, USA, pp. 131-140.

[50] W. Jiang, V. K. Prasanna, "Parallel IP Lookup using Multiple SRAM-based Pipelines," in Proc. IEEE International Symposium on Parallel and Distributed Processing, 2008, Miami, FL, pp. 1-14.

[51] J. Fong, X. Wang, Y. Qi, J. Li, W. Jiang, "ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification," in Proc. IEEE 20th Annual Symposium on High-Performance Interconnects, 2012, Santa Clara, CA, pp. 1-8.

[52] S. Gamage, A. Pasqual, "High Performance Parallel Scalable Packet Classification Architecture with Popular Rule Caching," in Proc. 18th IEEE International Conference on Networks, 2012.

[53] T. Ganegedara, V. K. Prasanna, "StrideBV: Single Chip 400G+ Packet Classification," in Proc. IEEE 13th International Conference on High Performance Switching and Routing, 2012, Belgrade, pp. 1-6.

[54] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, V. Prasanna, "Multi-dimensional Packet Classification on FPGA: 100 Gbit/s and Beyond," in Proc. International Conference on Field-Programmable Technology, 2010, Beijing, pp. 241-248.

[55] A. Fiessler, S. Hager, B. Scheuermann, "Flexible line speed network packet classification using hybrid on-chip matching circuits," in Proc. 18th International Conference on High Performance Switching and Routing, 2017, Campinas, Brazil

[56] S. Hager, F. Winkler, B. Scheuermann and K. Reinhardt, "MPFC: Massively Parallel Firewall Circuits," in LCN'14, Sep. 2014, pp. 305-313.

[57] A. Fiessler, S. Hager, B. Scheuermann, A. W. Moore, "HyPaFilter – A versatile hybrid FPGA packet filter," in Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2016.

[58] A. Fiessler, C. Lorenz, S. Hager, B. Scheuermann, A. W. Moore, "HyPaFilter+: Enhanced Hybrid Packet Filtering Using Hardware Assisted Classification and Header Space Analysis," IEEE/ACM Transactions on Networking vol. 25, 2017, pp. 3655-3669.

[59] W. Jiang, V. K. Prasanna, "Scalable Packet Classification on FPGA," in Proc. IEEE Transactions on Very Large Scale Integration Systems, 2012, pp. 1668-1680.

[60] W. Jiang, V. K. Prasanna, "Field-Split Parallel Architecture for High Performance Multi-Match Packet Classification Using FPGAs," in Proc. 21th ACM Symposium on Parallelism in Algorithms and Architectures, 2009.

[61] A. Sanny, T. Ganegedara, V. K. Prasanna, "A Comparison of Ruleset Feature Independent Packet Classification Engines on FPGA," in Proc. IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2013, Cambridge, pp. 124-133.

[62] A. Kennedy, X. Wang, "Ultra-High Throughput Low-Power Packet Classification," in Proc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2013, pp. 286-299.

[63] J. Wee, W. Pak, "Fast packet classification based on hybrid cutting," IEEE Communications Letters, 2017.

[64] T. Ganegedara, W. Jiang, V. K. Prasanna, "A Scalable and Modular Architecture for High-Performance Packet Classification," in Proc. IEEE Transactions on Parallel and Distributed Systems, 2014, pp. 1135-1144.

[65] Y. R. Qu, S. Zhou, V. K. Prasanna, "High-performance Architecture for Dynamically Updatable Packet Classification on FPGA," in Proc. 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2013, pp. 125-136.

[66] R. I. Khatami, M. Ahmadi, "High Throughput Multi Pipeline Packet Classifier on FPGA," 17th CSI International Symposium on Computer Architecture and Digital Systems, 2013, Tehran, pp. 137-138.

[67] V. Pus, J. Korenek, "Fast and Scalable Packet Classification Using Perfect Hash Functions," in Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2009, Monterey, California, USA, pp. 229-236.

[68] B. Yang, J. Fong, W. Jiang, Y. Xue, J. Li, "Practical Multituple Packet Classification Using Dynamic Discrete Bit Selection," IEEE Transactions on Computers, vol. 63, 2014, pp. 424-434.

[69] S. Hager, S. Brack, B. Scheuermann, "The Small, the Fast, and the Lazy (SFL): A General Approach for Fast and Flexible Packet Classification," in Proc, IEEE 41st Conference on Local Computer Networks, 2016, Dubai, United Arab Emirates

[70] H. Nakahara, T. Sasao, H. Iwamoto, M. Matsuura, "LUT Cascades Based on Edge-Valued Multi-Valued Decision Diagrams: Application to Packet Classification," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2016, pp. 73-86.

[71] P. Gupta and N. McKeown, "Packet classification on multiple fields," ACM SIGCOMM, pp. 147-160, 1999

[72] L. Sun, H. Le, V. K. Prasanna, "Optimizing Decomposition-based Packet Classification Implementation on FPGAs," in Proc. International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, pp. 170-175.

[73] Y. R. Qu, V. K. Prasanna, "Fast Dynamically Updatable Packet Classifier On FPGA," in Proc. 23rd International Conference on Field Programmable Logic and Applications, 2013, Porto, pp. 1-4.

[74] Y. R. Qu, V. K. Prasanna, "High-Performance and Dynamically Updatable Packet Classification Engine on FPGA," IEEE Transactions on Parallel and Distributed Systems, 2016, pp. 197-209.
[75] Y. Chang, Ch. Hsueh, "Range-Enhanced Packet Classification Design on FPGA," IEEE Transactions on Emerging Topics in Computing, vol. 4, 2016, pp. 214-224.
[76] A. Abdulhassan, M. Ahmadi, "Parallel many fields packet classification technique using R-tree," in Proc. Annual Conference on New Trends in Information & Communications Technology Applications, 2017, Baghdad, Iraq
[77] F. Risso, M. Baldi, O. Morandi, A. Baldini, P. Monclus, "Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation," in Proc. IEEE ICC'08, 2008, Beijing, China, pp. 5869-5875.
[78] A. Dainotti, A. Pescapé, K. C. Claffy, "Issues and Future Directions in Traffic Classification," IEEE Network, Jan/Feb, 2012, pp. 35-40.
[79] A. Callado, C. Kamienski, G. Szabó, B.P. Gerő, J. Kelner, S. Fernandes, D Sadok, "A Survey on Internet Traffic Identification," IEEE Communications Surveys & Tutorials, vol. 11, no. 3, July 2009.
[80] T. T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," IEEE Communications Surveys & Tutorials, vol. 10, no. 4, 2008, pp. 56-76.
[81] C. Charras, T. Lecroq, "Handbook of Exact String Matching Algorithms," King's College Publications, 2004.
[82] A. Aho and M. Corasick, "Efficient string matching: an aid to bibliographic search," Communications of the ACM, 1975, pp. 333-340.
[83] S. Kumar, J. Turner, J. Williams, "Advanced algorithm for fast and scalable deep packet inspection," in Proc. ACM/IEEE Symposium on Architecture for Networking and Communications systems, 2006, San Jose, CA, USA
[84] X. Wang, "High Performance Stride-based Network Payload Inspection," PhD Thesis, Dublin City University, 2012.
[85] N. Tuck, T. Sherwood, B. Calder, G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," In Proc. IEEE INFOCOM 2004, vol. 4, pp. 2628-2639.
[86] A. Bremler-Barr, Y. Harcholy, D. Hay, "Space-time Tradeoffs in Software-based Deep Packet Inspection," in Proc. IEEE 12th International Conference on High Performance Switching and Routing (HPSR), 2011, Catragena, Spain, pp.1-8.
[87] T. Liu, Y. Sun, L. Guo, "Fast and Memory-Efficient Traffic Classification with Deep Packet Inspection in CMP Architecture," in Proc. IEEE 5th International Conference on Networking, Architecture, and Storage (NAS), 2010, Macau, China, pp. 208-217.
[88] B. Commentz-Walter, "A string matching algorithm fast on the average," Automata, Languages and Programming. ICALP 1979. Lecture Notes in Computer Science, vol 71. Springer, Berlin, Heidelberg
[89] S. Wu, U. Manber, "A fast algorithm for multi-pattern searching," Tech. Report, TR-94-17, Dept. of Comp. Science, Univ of Arizona, 1994.
[90] R.S. Boyer, J.S. Moore, "A Fast String Searching Algorithm," Communications of the ACM, 20 (10), pp. 762–772, 1977.
[91] G. F. Ahmed, N. Khare, "Hardware based String Matching Algorithms: A Survey," International Journal of Computer Applications, 88 (11), 2014.
[92] G. Szabó, Z. Turányi, L. Toka, S. Molnár, A. Santos, "Automatic Protocol Signature Generation Framework for Deep Packet Inspection," in Proc. ICST 5th Conference on Performance Evaluation Methodologies and Tools, 2011, Paris, France, pp. 291-299.
[93] S. Fide, S. Jenks, "A Survey of String Matching Approaches in Hardware,"" Department of Electrical Engineering and Computer Science, University of California, Irvine, Tech. Rep. TR SPDS 06-01. 2006.
[94] P. Ferragina, G. Manzini, "Indexing Compressed Text," Journal of the ACM (JACM), 52 (4). pp. 552-581, 2005.
[95] R. W. Floyd, J. D. Ullman, "The Compilation of Regular Expressions into Integrated Circuits," Journal of ACM, vol. 29, no.3, 1982. pp. 603-622.
[96] I. Sourdis, D. N. Pnevmatikatos, S. Vassiliadis, "Scalable Multigigabit Pattern Matching for Packet Inspection," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 2, 2008, pp. 156-166.
[97] M. Dhanapriya, C. Vasanthanayaki, "Hardware Based Pattern Matching Technique for Packet Inspection of High Speed Network," in Proc. of International Conference on Control, Automation, Communication and Energy Conservation, 2009.
[98] F. Wang, Y. Hong, J. Jin, "Research on Regular Expression Data Packet Matching Algorithm Based on Three State Content Addressable Memory," International Journal of Simulation - Systems, Science and Technology, 16 (5A), 2015, pp. 8.1-8.5.
[99] R. Sidhu, V. K. Prasanna, "Fast regular expression matching using FPGAs," in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001), Rohnert Park, CA, USA, April 2001.
[100] C. R. Clark, D. E. Schimmel, "Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns," In Proc. 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lecture Notes in Computer Science, vol 2778. Springer, Berlin
[101] Cisco, "Snort Users Manual," Technical Report, 2018.
[102] J. Moscola, J. Lockwood, R. P. Loui, M. Pachos, "Implementation of a content-scanning module for an internet firewall," in Proc. 11th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 2003), Napa, USA, April 2003.
[103] M. Becchi, P. Crowley, "A hybrid finite automaton for practical deep packet inspection," CoNEXT, 2007.
[104] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in Proc. ACM Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '06), 2006, Pisa, Italy, pp. 339-350.
[105] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, A. Di Pietro, "An improved DFA for fast regular expression matching," SIGCOMM Computer Communication Review 38, Number 5, 2008, pp. 29-40.
[106] R. Smith, C. Estan, S. Jha, "XFA: Faster Signature Matching with Extended Automata," in Proc. IEEE Symposium on Security and Privacy (SP 2008), Oakland, CA, USA, May, 2008.
[107] S. Yusuf, W. Luk, "Bitwise Optimised CAM for Network Intrusion Detection Systems," In Proc. 15th International Conference on Field Programmable Logic and Applications (FPL 2005), IEEE, Tampere, Finland, August, 2005.
[108] N. J. Larsson, "Structures of string matching and data compression," PhD thesis, Dept. of Computer Science, Lund University, 1999.
[109] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, J. W. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters," IEEE Micro, vol. 24, no. 1, Jan-Feb. 2004.
[110] Z. K. Baker, V. K. Prasanna, "Automatic Synthesis of Efficient Intrusion Detection Systems on FPGAs," in Proc. 14th Field Programmable Logic and Applications (FPL 2004), IEEE, Leuven, Belgium, August 2004, pp. 311–321.
[111] Y. H. Cho, W. H. Mangione-Smith, "Deep Packet Filter with Dedicated Logic and Read Only Memories," in Proc. 14th Field Programmable Logic and Applications (FPL 2004), IEEE, Leuven, Belgium, August 2004, pp. 125–134.
[112] M. Gokhale, D. Dubois, M. Boorman, S. Poole, V. Hogsett, "Granidt: Towards Gigabit Rate Network Intrusion Detection Technology," In Proc. 12th International Conference on Field Programmable Logic and Applications (FPL 2002), Lecture Notes in Computer Science, vol 2438. Springer, Berlin, pp. 404–413.
[113] D. Selvaraj and P. Ganapathi, "Packet Payload Monitoring for Internet Worm Content Detection Using Deterministic Finite Automaton with Delayed Dictionary Compression," Journal of Computer Networks and Communications, Article ID 206867, 2014, pp. 1-9.
[114] C.L. Hung, C.Y. Lin, P.C. Wu, "An Efficient GPU-Based Multiple Pattern Matching Algorithm for Packet Filtering," Journal of Signal Processing Systems, 86 (2), 2017, pp. 347-358.
[115] Y.S. Lin, C.L. Lee, Y.C. Chen, "Length-Bounded Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection," Algorithms, 10 (16), 2017, pp. 1-13.
[116] T. Bakhshi, B. Ghita, "On Internet Traffic Classification: A Two-Phased Machine Learning Approach," Journal of Computer Networks and Communications, Article ID 2048302, 2016, pp. 1-21.
[117] Y. Du, R. Zhang, "Design of a method for encrypted P2P traffic identification using K-means algorithm," Telecommunication Systems (53), 2013, pp. 163–168.
[118] A. Finamore, M. Mellia, M. Meo, D. Rossi, "KISS: Stochastic Packet Inspection," Traffic Monitoring and Analysis, LNCS Volume 5537, 2009, pp. 117-125.
[119] B. Hullar, S. Laki, A. Gyorgy, "Efficient Methods for Early Protocol Identification," IEEE Journal on Selected Areas in Communications, 32 (10), 2014, pp. 1907-1918.

[120] P. Velan, M. Cermak, P. Celeda, M. Drasar, "A Survey of Methods for Encrypted Traffic Classification and Analysis," International Journal of Network Management, 2014, pp. 1–24.
[121] M. Sheikhan, Z. Jadidi, "Flow-based anomaly detection in high-speed links using modified GSA-optimized neural network," Neural Computing and Applications, 24 (3), 2014, pp 599–611.
[122] G. Munz, N. Weber, G. Carle, "Signature Detection in Sampled Packets," in Proc. of IEEE Workshop on Monitoring, Attack Detection and Mitigation (MonAM), 2007.
[123] B. Claise, A. Johnson, J. Quittek, "Packet Sampling (PSAMP) Protocol Specifications," IETF RFC5476, 2009.
[124] A.G. Alagu Priya, H. Lim, "Hierarchical packet classification using a Bloom filter and rule-priority tries," Computer Communications 33 (10), 2010, pp. 1215-1225.
[125] H. Lim, S.Y. Kim, "Tuple Pruning Using Bloom Filter for Packet Classification," IEEE Micro, 2010, pp. 48-58.
[126] S. Dharmapurikar, H. Song, J. Turner, J. Lockwood "Fast Packet Classification Using Bloom filters," in Proc. ACM/IEEE Symposium on Architecture for Networking and Communications systems, 2006, San Jose, USA, pp. 61-70.
[127] M. Ahmadi. S. Wong, "K-Stage Pipelined Bloom Filter for Packet Classification," in Proc. International Conference on Computational Science and Engineering, 2009, Vancouver, Canada, pp. 64-70.
[128] J. Nocedal and S. Wright, "Numerical Optimization," Springer, New York, 1999.
[129] C. Kelley, "Iterative Methods for Optimization," SIAM Publications, Philadelphia, 1999.
[130] F. Burden, D. Winkler, "Bayesian Regularization of Neural Networks," Artificial Neural Networks, pp. 23-42., Springer
[131] R. M. Neal, "Bayesian learning for neural networks," Springer, 1996.
[132] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning," MIT Press, 2016.
[133] T. Auld, A. W. Moore, S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification," IEEE Transactions on Neural Networks, 2007, pp. 223-239.
[134] W. Zhou, L. Dong, L. Bic, M. Zhou, L. Chen, "Internet traffic classification using feed-forward neural network," in Proc. International Conference on Computational Problem-Solving, 2011.
[135] M. Elmahgiubi, O. Ahmed, S. Areibi, G. Grewal, "Efficient algorithm selection for packet classification using machine learning," in Proc. IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks, 2016, Toronto, ON, Canada
[136] S. Ostermann, "tcptrace," http://www.tcptrace.org, viewed on 19/02/2018
[137] A. Este, F. Gringoli, L. Salgarelli, "On the Stability of the Information Carried by Traffic Flow Features at the Packet Level," in Proc. ACM SIGCOMM Computer Communication Review, 2009, pp. 13-18.
[138] M. Zelina, M. Oravec, "Early Detection of Network Applications using Neural Networks," IEEE ELMAR, 2011.
[139] L. Peng, B. Yang, Y. Chen, "Hierarchical RBF Neural Network Using for Early Stage Internet Traffic Identification," in Proc. IEEE 17th International Conference on Computational Science and Engineering, 2014.
[140] M. Turcanik, "Packet Filtering by Artificial Neural Network," in Proc. International Conference on Military Technologies, 2015.
[141] M. M. Moallem, N. Yazdani, K. Faez, H. Taheri, "A Multilayer Neural Network for IP Lookup and Packet Classification," in Proc. The 9th Asia-Pacific Conference on Communications, 2003.
[142] L. Peng, H. Zhang, B. Yang, Y. Chen, M. T. Qassrawi, G. Lu, "Traffic Identification Using Flexible Neural Trees," in Proc. 18th International Workshop on Quality of Service, 2010.
[143] R. Sun, B. Yang, L. Peng, Z. Chen, L. Zhang, S. Jing, "Traffic Classification Using Probabilistic Neural Networks," in Proc. 6th International Conference on Natural Computation, 2010.
[144] Z. Jingquan, L. Yanxia, C. Zhenzhen, L. Juan, Z. Linkai, C. Jiebao, "P2P Traffic Identification Based on Bayesian Regularization BP Neural Network," in Proc. 12th IEEE International Conference on Communication Technology, 2010.
[145] G. Yiran, W. Suoping, "Traffic Identification Method for Specific P2P Based on Multilayer Tree Combination Classification by BP-LVQ Neural-Network," in Proc. International Forum on Information Technology and Applications, 2010.
[146] S. Fuke, C. Pan, R. Xiaoli, "Research of P2P Traffic Identification Based on BP Neural Network," in Proc. Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2007.

Péter Orosz is an assistant professor and the head of Smart Communications Laboratory at the Department of Telecommunications and Media Informatics, BME Hungary. He received his Computer Science master degree in the field of software engineering (2003) and Ph.D. in infocommunication systems (2010) at the University of Debrecen, Hungary. Previously he has been working for the University of Debrecen. His research interest covers communication networks, network and service management, QoS-QoE managed networks, online QoE prediction for media services, and hardware acceleration of network functions.

Tamás Tóthfalusi is currently a Ph.D. student at the Budapest University of Technology and Economics (BME). He received the B.Sc. degree in Infocommunication Networks as part of the degree program in Engineering Information Technology at the University of Debrecen from 2006 to 2010, and the M.Sc. degree in Hardware Programming at the University of Debrecen from 2010 to 2012. His research interests include hardware (FPGA) based acceleration of network processes, network management and measurement, VoLTE signaling and neural networks. He is a member of the SmartCom Laboratory at BME. He has been involved in industrial research and development projects in these topics.

Pál Varga is Associate Professor at the Department of Telecommunications and Media Informatics, BME, Hungary, where he got his M.Sc. (1997) and Ph.D. (2011) degrees from. Besides, he is director in AITIA International Inc. Earlier he was working for Ericsson Hungary and Tecnomen Ireland, as software design engineer and system architect, respectively. His main research interest include communication systems, network monitoring, network performance measurements, root cause analysis, fault localization, traffic classification, end-to-end QoS and SLA issues, as well as hardware acceleration, and Internet of Things. He has been involved in various industrial as well as European research and development projects in these topics.