Towards Embedded Packet Processing Devices for Rapid Prototyping of Avionic Applications

Fabien Geyer[*]
Technical University of Munich (TUM)
D-85748 Garching b. München, Germany
Email: fgeyer@net.in.tum.de

Max Winkel
Airbus Group Innovations
D-81663 Munich, Germany
Email: max-oliver.winkel@airbus.com

***Abstract*—Recent developments made in the area of network-specific reconfigurable hardware and associated description language – namely the P4 language and its back-ends – promise interesting features for rapid prototyping of packet processing devices. Due to the need for high flexibility and increasing trend towards softwarization, such solution is of interest for the aeronautical industry where custom network protocols are generally used.**

Our contributions in this paper are two fold. First, an analysis of the functionalities of P4 with respect to requirements usually necessary for applications and network protocols in the aeronautic industry is performed. In a second step, a performance evaluation of three different platforms was done. Those platforms represent three use-cases: an existing software-based back-end using Intel DPDK, a hardware network accelerator based on a network processor unit, and an FPGA-based platform. Performance of those platforms are compared to existing state-of-the-art hardware solutions used in by the aeronautical industry.

I. Introduction

In the last two decades, distributed embedded electronic applications have become the norm in a large part of the aeronautical industry. Those applications cover a large set of functionalities with different requirements, ranging from flight control with hard real-time and strict safety constraints, to passenger entertainment with less stringent constraints. Due to those constraints and safety aspects associated with aircrafts, specific equipments are generally used in order to fulfill those constraints. When such equipments are not available off-the-shelves, costly and time consuming developments have to be undertaken. This is especially true in the scope of networking equipments, since standard off-the-shelves devices for networking usually do not support aeronautical-specific network protocols or safety-related functionalities.

A prevailing solution commonly used to address this issue is to employ FPGAs (Field Programmable Gate Arrays) since they offer high customizability with high performance at moderate costs. The two main drawbacks of this approach are that current developments using FPGAs require a high level of expertise and long development times to produce efficient and bug-free devices. We propose in this paper to look at recent developments made in the area of network-specific reconfigurable hardware and associated tools, namely P4 (Programming Protocol-independent Packet Processors), recently proposed by Bosshart et al. in [1], and assess if they are a fit for aeronautical applications.

Our main contribution in this paper is an analysis of those new solutions in the scope of aeronautical applications in terms of offered features and performance. We first investigate their applicability from a functional point of view and identify missing features of the current approaches. We show that its high flexibility in combination with simple building blocks enabling a formal analysis make it an attractive platform for some network protocols used in aeronautical use-cases. However, some features such as controlling of egress packet scheduling and methods for time-based or time-triggered protocols for more advanced network protocols are undefined or vendor-specific. In order to evaluate P4 in an aeronautical context, we choose AFDX as a case-study and demonstrate that a simplified AFDX switch can be implemented using P4.

In a second step, we do a performance analysis using measurements on three different target hardware and investigate if those new developments and platform are sufficient for aeronautical applications from a performance point of view. Our first target is a purely software-based solution using Intel's Data Plane Development Kit framework (DPDK) [2], which is a set of libraries and drivers for fast packet processing in the Linux userland. Our second target is based around the Netronome Agilio CX platform, a hardware-based Network Flow Processor (NFP) which is able to offload most packet processing functionalities from the CPU. Our third target is based on a FPGA (Field-Programmable Gate Array), where packet processing is fully performed in hardware. Measurements done using a network analyzer show that those platforms are able to achieve packet processing latencies similar to those of devices used by the aeronautical industry and a commercial-of-the-shelf (COTS) Ethernet switch.

The rest of this paper is organized as follows. In Section II we present similar research studies. We then introduce in Section III the new advances made regarding network-specific reconfigurable hardware and their associated tools. In Section IV, we present its applicability to aeronautical requirements, with a concrete application to existing aeronautical network protocol and architectures. We do a performance evaluation of a target hardware in Section V with results regarding packet processing latency of frames and resource

[*]This work was performed while the author was with Airbus Group Innovations in Munich, Germany.

utilization. Finally, Section VI concludes our work.

II. Related work

Approaches towards a top-down description of data-plane in a high-level programming language have been proposed since the late 1990s and early 2000nd. Kohler et al. proposed Click in [3] which enables flexible packet processing in software, but with the drawback of difficulty regarding compilation to dedicated hardware.

More recently with the increasing use of FPGAs (Field Programmable Gate Array) for packet processing, Brebner and Jiang proposed the PX programming language in [4] with a compiler targeting FPGAs. Dedicated hardware for packet processing such as NPUs (Network Processor Unit) [5] or RMT (Reconfigurable Match Table) [6] have also been proposed. Song proposed POF (Protocol-Oblivious Forwarding) in [7], which defines an Flow Instruction Set which is used for processing packets.

Regarding purely software-based packet processing on commodity multi-core processors, various works have been performed on the performance of such platforms. Dobrescu et al. evaluated the predictability of such platform in [8]. They evaluated how contention for shared hardware resources such as caches can be taken into account for improving performance predictability, an important aspect in case of safety critical applications. More recently, Emmerich et al. benchmarked various Linux-based software stacks for software-based packet processing in [9] and identified various bottlenecks responsible for poor performance.

On the proposition of more advanced networking stacks for industrial applications, various work have been done in the scope of Quality-of-Service and auto-configuration. Henneke et al. provided a survey over the challenges and proposed solutions on applying Software-Defined Networking (SDN) paradigms to industrial networks in [10]. Various requirements such as application-aware QoS, timing performance, monitoring, security, reliability were reviewed, with the conclusion that experience on applying SDN to existing industrial networks is still lacking. Heise et al. proposed to apply SDN paradigms to avionic networks with real-time guarantees in [11]. It was showed that deterministic network functionalities similar to the one commonly found in real-time networks could be achieved using SDN and OpenFlow. While those works have shown the possible applicability of SDN to industrial networks, a P4-based solution as presented in this paper might be more tailored to embedded applications where simpler functionalities are required.

III. A new approach for packet processing devices

A. Main promises of P4

In conjunction with the current trend towards softwarization of functionalities in the field of communication networks with the advent of Software Defined Networking and related technologies, a recent development called P4 [1] – Programming Protocol-Independent Packet Processor – proposes a flexible way to specify packet processing devices. The main promises of the P4 programming language and toolchain are:

1) A simple specification of packet processing pipelines using a high-level Domain Specific Language (DSL), requiring no expert knowledge about the final hardware. This DSL was specially designed to be expressive enough for the various actions necessary in network protocols, while restrictive enough to enable simple compilation to dedicated target hardwares. Sample snippets of P4 descriptions for standard Ethernet and IPv4 routing are given in Listings 1 and 2. The complete specification of the P4 language is available on the P4 website [12, 13].
2) Compilation of specification for different hardware targets, ranging from FPGAs (Field Programmable Gate Array) to NPUs (Network Processing Unit) to finally purely software solutions targeting multi-core and many-core processors, as presented later in Section V;
3) Reconfigurability in order to modify the behavior of packet-processing devices in the field;
4) Possibility to test packet processing pipelines using well-known network emulation tools such as mininet [14] and ability to emulate complete network architectures.

This approach is also in line with model driven engineering, where high level descriptions of systems are used in order to formally verify various properties of systems.

Currently, two different P4 standards are evolving in parallel: $P4_{14}$, which is the original P4 and subject of this paper, and $P4_{16}$, a major redesign of the language with an object oriented approach. If not stated otherwise, the text refers to $P4_{14}$ only.

Listing 1: Example of Ethernet frame format definition in P4

```
header_type ethernet_t {
  fields {
    dstAddr   : 48;
    srcAddr   : 48;
    etherType : 16;
  }
}
```

Listing 2: Example of IPv4 packet routing in P4

```
action route_ipv4(dst_port, dst_mac, src_mac, vid) {
  modify_field(standard_metadata.egress_spec, dst_port);
  modify_field(ethernet.dst_addr, dst_mac);
  modify_field(ethernet.src_addr, src_mac);
  modify_field(vlan_tag.vid, vid);
  add_to_field(ipv4.ttl, -1);
}
```

B. Abstract forwarding model

P4 uses a generic packet processing pipeline as a basis called *abstract forwarding model*. This model applied to a switch is illustrated here in Figure 1. Packets are first parsed according to customizable frame format definitions.

Based on the fields and associated values of the protocols, so-called *match+action tables* are used in order to process packets. Available actions include packet modification (changing field value, adding or removing headers), replication (for broadcast or multicast), dropping packets or triggering of flow control (namely update of action tables such as counters or policers). Those match+action tables are conceptually similar to the ones used in OpenFlow switches.

IV. Applicability for aeronautical applications

As illustrated in Section III, P4 promises various properties which make it attractive for the aeronautical industry. We investigate in this section the current features of P4 and their applicability to aeronautical applications.

A. The good parts of P4

The main advantage of P4 is the decorrelation between the behavior of a packet processing device and the hardware which is used. It means that engineers are not tied to a specific set of network protocols implemented by hardware vendors. This is especially relevant in the aeronautical industry since the two following constraints are usually present:

1) Specific network protocols only used by the aeronautical industry are used (e.g. ARINC standards);
2) For safety reasons, devices must usually only implement the required protocols and functionalities, meaning that no additional features should be implemented or used.

Those two constraints usually prevent COTS devices to be used since they may not support the required protocols, or implement a larger set of protocols than the ones which are required. With P4, the usability of COTS devices increases.

The second advantage of P4 is the simplicity and constraints put on the abstract forwarding model presented earlier in Figure 1. Since P4 forbids dynamic memory allocation and iterations with unknown counts – unlike more generic programming languages such as C – formal derivations of worst-case execution time and resource usage of a P4 program are fairly straightforward. This means that per-packet latency, memory footprint and maximum throughput of a packet-processing pipeline can be determined at compile time. This is again relevant in the aeronautical industry since constraints on those cost factors are required in real-time applications. In conjunction with FPGA based platforms for P4 such as [15], deterministic processing may be achieved in hardware components.

Finally, regarding the features supported by P4 in terms of packet processing actions, it covers most of the use-cases relevant for network protocols used by the aeronautical industry. Missing features are listed in the next section.

B. Avenues for improvement

While P4 offers a lot of flexibility for expressing packet-processing pipelines, some features are still missing for more advanced uses needed in avionic applications.

Egress packet scheduling cannot be directly described by P4. While there is some limited support for defining the priority of a packet in case of targets supporting Strict Priority Queuing (SPQ), more advanced schedulers such as Weighted Fair Queuing [16] or Deficit Round Robing [17] are not defined in P4. There are vendor-specific interfaces to control the scheduling, such as described in Section V-A3. Nevertheless, in general, the description of more advanced Quality-of-Service architectures which are envisioned for next-generation aeronautical backbones such as the one presented in [18] is limited with P4.

Since safety is an important aspect of aeronautical applications, specification and programing languages need to have defined behavior. In the 2014 specification of P4 [12], some aspects are incompletely specified, as for instance overflow of integers, casting between different data types, exception handling, and initial values of table entries and packet attributes.

Finally, time-based or time-triggered protocols cannot be directly described using P4 since there are no primitives for describing access to a clocking information. This drawback prevents the implementation of time-synchronization protocols for packet timestamping, or egress scheduling based on time information. Such protocols and mechanisms need to implemented around P4 in a target specific manner and may eventually be interfaced, for example, by vendor-specific metadata.

We note that the drawbacks listed here are with respect to the 2014 specification of P4 [12], being the version which is supported by the majority of platforms available at the time of writing. A new version of the language has been published under the name $P4_{16}$ [13], along with the Portable Switch Architecture (PSA) [19] defining a set of standardized common capabilities of network switches. Issues regarding undefined behaviors have been addressed in $P4_{16}$. For functionalities needing timing information, timestamps at ingress and egress have been added in the Portable Switch Architecture with a recommendation to use microsecond precision. The use-cases targeted for this timing information are inband telemetry for measuring queuing latencies, and checking of timeouts or keep-alive in network protocols.

Due to the promises of P4 and its applicability to a large variety of use-cases, improvements have been proposed in the literature. For instance, a solution for the specification of egress packet scheduling has been recently proposed by Sivaraman et al. in [20]. Extensions of P4 switches with other languages are also being investigated in order to simplify the addition of functionalities to switches. For example, the Domino programming language has recently been proposed by Sivaraman et al. in [21].

C. Case-study: AFDX switching

In order to evaluate the applicability of P4 to aeronautical use-cases, we propose to apply P4 to the case-study of Avionics Full-Duplex Switched Ethernet (AFDX), an Ethernet-based protocol for safety-critical applications standardized in ARINC 664 Part 7 [22].

An AFDX network is composed of *end-systems* and *switches* as nodes. End-systems serve as source and destination nodes in the network, over which applications may send data according to bandwidth restrictions to avoid overloading. One fundamental building block of AFDX is the notion of *virtual link* (VL), which can be seen as rate-constrained network

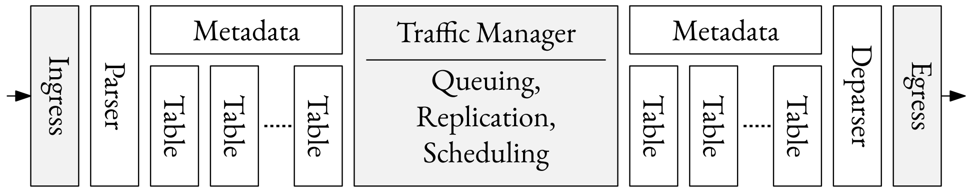


Figure 1: P4 Abstract Forwarding Model of a switch

tunnels. The parameters describing a VL are: the emitter end-system of this VL, the list of receiving end-systems, static routes between emitter and receivers, the Bandwidth Allocation Gap (BAG), as well as minimum and maximum frame length (s_{min} and s_{max}). The BAG is defined as the minimum time interval between the first bit of two consecutive frames from the same VL.

We will focus in the rest of this section on the implementation of a simplified AFDX switch with P4. Switches must ensure the following functionalities for each frame entering the switch:

- Identification of the Virtual Link;
- Frame filtering and policing based on Virtual Link parameters: allowed input port, BAG and frame length limits;
- Forwarding of the frame to the correct output ports based on the Virtual Link routing configuration.

Frames in AFDX are based on the standard Ethernet frame format. An addressing schema is defined in ARINC 664 Part 7 (Section 3.2.5) in order to encode the Virtual Link identification number in the last 16 bits of the destination MAC address. Listing 3 represents how this identifier can be easily extracted using P4.

Listing 3: Simplified AFDX frame header in P4

```
header_type afdx_t {
  fields {
    dstConst   : 32;
    dstVlinkID : 16;
    srcAddr    : 48;
    etherType  : 16;
  }
}
header afdx_t afdx;
```

Switch configuration and routes in P4 are saved in so-called *tables*. For our case-study, we define a table containing the allowed input port of each Virtual Link and its output port, as illustrated in Listing 4. Incoming packets with invalid Virtual Link identifiers are dropped here.

Listing 4: AFDX forwarding table

```
table tbl_forward_virtual_link {
  reads {
    standard_metadata.ingress_port : exact;
    afdx.dst_vlink_id              : exact;
  }
  actions {
    drop;
    forward;
  }
  size : MAX_VIRTUAL_LINKS;
}
```

The P4 function `ingress` is called for each incoming frame. Listing 5 describes a simplified ingress function for AFDX with basic frame integrity checking (with respect to the ARINC 664 requirements) application of the table from Listing 4, and policing.

Listing 5: AFDX ingress function

```
control ingress {
  integrity_check();
  apply(tbl_forward_virtual_link);
  traffic_policing();
}
```

Regarding policing of AFDX frames, a simple process based on validating the BAG timing properties and minimum and maximum frame sizes is described in the ARINC 664 Part 7 standard. While frame sizes validation is possible with P4, filtering based on the time between frames is not (as mentioned earlier in Section IV-B). Since from a functional point of view the goal is to limit the bandwidth of each Virtual Link, the standard policing mechanisms provided by P4 may be used as an alternative. Those mechanisms are based on the use of the two token buckets, as defined in RFC 2698 [23]. Those meters can be easily created in P4, as illustrated in Listing 6.

Listing 6: Policing based on

```
meter vlink_bandwidth_bytes {
  type   : bytes;
  direct : tbl_forward_virtual_link;
  result : scheduling_metadata.color_bytes;
}
```

Finally, the last step is to forward frames to the correct output ports. This is illustrated in Listing 7 with the use of multicast by a vendor-specific mechanism.

Listing 7: forwarding

```
action forward(egress_ports) {
  modify_field(standard_metadata.egress_spec,
        EGRESS_SPEC_MULTICAST);
  modify_field(intrinsic_metadata.egress_port_bitmap,
        egress_ports);
}
```

We listed here only a subset of the functionalities needed by an AFDX switch. More advanced features such as of operational modes, priority-based packet scheduling, and monitoring functionalities based on SNMP can also be implemented using P4.

V. Performance evaluation

A. Presentation

We propose in this section to do a performance evaluation of three different P4 targets: software-based, software-based with hardware acceleration and FPGA-based platform.

1) Software based target: The first target is a purely software based solution, which has been proposed by Laki et al. in [24]. This target is based on the Intel Data Plane Development Kit framework (DPDK) [2], which is a set of libraries and drivers for fast packet processing in the Linux userland. As illustrated in Figure 2, DPDK enables developers to bypass the kernel and process frames directly in user-space. Standard kernel overheads are avoided using DPDK, namely system calls, context switching on blocking I/O, data copying from kernel to user space or interrupt handling in kernel. Benchmarks have shown that DPDK enables much faster packet processing, as shown for example in [9]. Predictable performance without the interference of the Linux scheduler and other processes may be achieved by pinning the DPDK process to dedicated CPU cores.

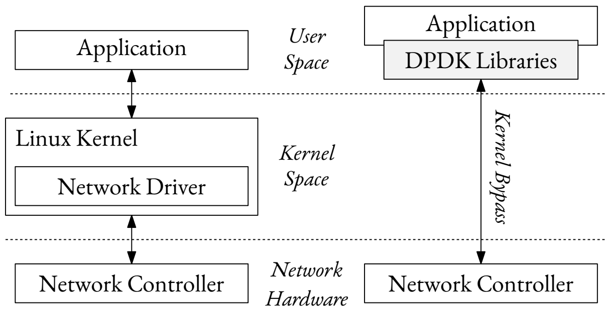


Figure 2: Overview of the DPDK framework

A standard PC equipped with an Intel i7-2600 CPU[2] with 4 physical cores and an Intel I340 T4 network card[3] was used here. This network card has 4 ports supporting 1 Gbit/s Ethernet. Regarding software, Ubuntu 16.04 with the default kernel, DPDK 16.04 and the software presented in [24][4] was used. This software is compiler from P4 to a DPDK-based application. Minor modifications to the code from [24] were made to add profiling and remove some unnecessary overhead.

Such a platform might be interesting for functional testing or in services with short life-cycles and no hard real-time guarantees needed. Typical services such as passenger connectivity could fit these requirements, since on-board passenger devices have a fast update rate, with changing needs and protocols.

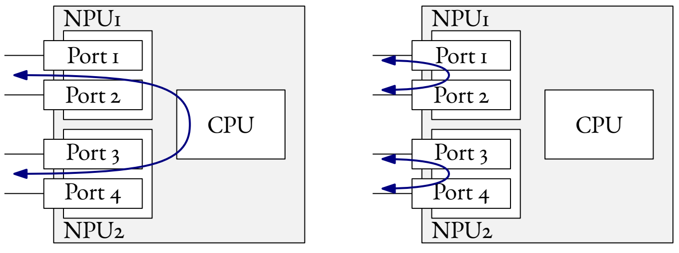
Still, it should be kept in mind, that the performance analysis of the software target was performed on a PC and the results are not directly transferable to embedded platforms with usually very limited resources (CPU processing power, memory, caches, interface bandwidths). If to be used in field, an in-depth performance analysis has to be performed using the specific hardware and software.

[2]Intel i7-2600: https://ark.intel.com/products/52213
[3]Intel I340 T4: https://ark.intel.com/products/49186
[4]P4@ELTE software from [24]: https://github.com/P4ELTE/p4c

2) Network processor platform: The second target is based around the Netronome Agilio CX SmartNIC [25], a hardware-based Network Flow Processor (NFP) which is able to offload the packet processing from the CPU. Those network cards are based on the NFP-4xxx silicon, a many-core architecture with 72 programmable cores with 8 threads each, and 2GB DRAM for lookup and state tables. A proprietary P4 compiler from Netronome was used for the evaluation.

The cards which were used have two ports supporting 10 Gbit/s Ethernet. In order to provide comparable results with the previous target with four ports, two cards were used. To enable communication between the two cards, frames need to be copied from one card to the other. A simple DPDK program without any packet processing logic running on the local CPU of the test platform was used for this purpose. As illustrated in Figure 3a, frames coming from one NPU (Network Processing Unit) are first processed on board, then copied to the main memory for forwarding by the CPU, and finally copied to the second NPU. In order to minimize the latency jitter, all packets are always sent to the CPU, even if the final destination is on the same NPU and could be forwarded without CPU involvement.

While the Agilio CX SmartNIC requires a server platform and as such is not suited for the integration into an embedded environment, the NFP-4xxx silicon on which it is based, might be. Therefore, to additionally evaluate the performance of the chip, a second series of measurements was performed without using the datapath through the CPU, assuming that the performance of the SmartNIC in this setup is mainly constrained by the NFP silicon. The packets were directly forwarded back to back from one physical port to the other, as illustrated in Figure 3b. To get comparable results, the same P4 program was used in both cases with different table configurations programming the destination of each packet.



(a) With CPU processing (b) Without CPU processing

Figure 3: Data paths used in the network processor platform

3) Prototype FPGA based target: The FPGA target is based on the Xilinx Zynq-7035 MPSoC [26]. It features a dual core ARM Cortex-A9 processor system (PS) which is closely coupled to a programmable logic (PL) based on the Kintex architecture with 275k logic cells and, in our configuration, eight high speed serial "GTX" transceivers, able to operate at up to 10.3125 GHz bit rate. Five of these transceivers are used as 10GBASE-R Ethernet ports. The main part of the firmware is an Ethernet switch with optional TSN (Time Sensitive Networking) features. The switch is connected to the external 10GBASE-R ports, as well as to internal virtual network interface cards (VNIC) connected to the CPU (as illustrated in Figure 4).

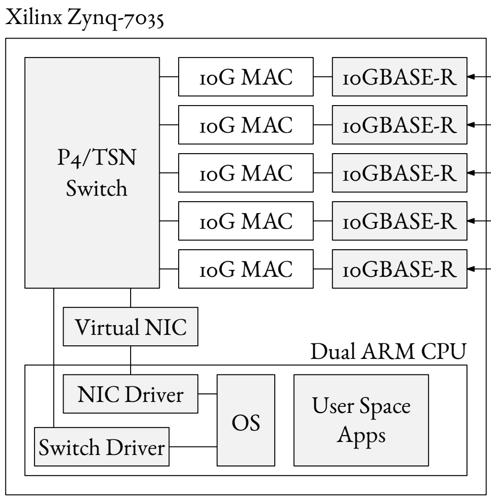


Figure 4: Overview of the implemented Zynq firmware

The packet processing and forwarding is performed completely by the switch core within the PL without any involvement of the PS, allowing maximum throughput and minimum latency. The architecture of the switch itself is outlined in Figure 5. The incoming Ethernet packets from each port (external or internal) are queued to ingress buffers from which they are multiplexed in a round-robin manner to the central processing pipeline. There, the MAC header information are extracted and looked up in hashed content addressable memories (CAMs) to determine the destination port(s) and queues. A demultiplexer distributes the packets to the selected queues and seamlessly duplicates the packets for multicasting. If the selected queues (implemented as dedicated block RAMs) are full, the packets may be buffered in external DDR3 memory.

Each port has 8 egress queues. The order in which the non-empty queues are selected for transmission is determined by a combination of strict priority [27, Sec 8.6.8.1], round robin and the TSN algorithms CBS (Credit Based Shaper) [27, Sec 8.6.8.2] and TAS (Time Aware Shaper) [28, Sec 8.6.8.4]. The parameters for the different algorithms are run time configurable per queue. For the following measurements, only one queue was used per port without any scheduling or traffic shaping.

In the current implementation, the packet processing logic is hardcoded in VHDL but is already prepared to be substituted by P4 generated cores to enable fast and flexible modifications to the behavior. For that, we are currently aiming at two very promising approaches. Xilinx has included a P4 compiler into its SDNet Development Environment [29] which translates the P4 description into an IP core which can be integrated to the FPGA firmware. A drawback might be the restriction to Xilinx FPGAs. The second approach comes from Netcope Technologies which offers a P4 to VHDL compiler [30] together with a suite of networking IP cores. In either approach, the generated packet processing pipeline has a fixed function completely described by the P4 program which is a very important aspect with regard to safety assessment and certification. Unfortunately, at the time of writing none of the two approaches were ready for implementation into our firmware.

It's further to be noted, that the switch core was originally designed for a 1 Gbit/s switch and therefore currently has a limited internal bandwidth of around 34 Gbit/s which obviously is not sufficient to serve five 10 Gbit/s interfaces at full line rate. It's therefore possible for the ingress buffers to overflow causing packets to be dropped at ingress before processing and before assigning priorities.

B. *Measurement setup*

Our measurement setup is presented in Figure 6. Traffic was generated by an Anritsu Network Analyzer MD1230B [31] on four different Ethernet links, generating traffic for a utilization from 0 % to 100 % (ie. up to 4 Gbit/s for the software based platform, and 40 Gbit/s for the NPU and FPGA platforms). This traffic is then processed and forwarded to the according output ports on the target platform.

Regarding the P4 program which was used for making the measurements, we used here a simple layer 2 Ethernet switch. This P4 program parses the Ethernet header and decides which output port(s) to use based on a learned or statically, run-time configured MAC address table.

C. *Framerate*

Figure 7 presents the framerate achieved by the three platforms for different packet sizes. Note that an ideal platform would be able to forward 100 % of the workload for the transmitted framerate presented in Figure 7. For packet sizes of 64 B, the software-based platform reaches a bottleneck at around 2.2 Mpps or 1.1 Gbit/s, meaning that it is only able to process 28 % of the full workload of 4 Gbit/s.

In case of the network processor platform, the bottleneck is reached at 8.7 Gbit/s with the CPU and 23 Gbit/s without CPU, meaning it is able to process respectively 21.8 % and 57.5 % of the full workload of 40 Gbit/s. The difference between the two measurements is due to the direct forwarding of all packets without any involvement of the CPU or main memory, which is also the bottleneck for the software platform.

The FPGA platform is only able to process a maximum of 5.6 Gbit/s with 64 B packet size. Using 1518 B packets, the maximum throughput of 28 Gbit/s is closer to the internal bandwidth limit of 34 Gbit/s. In addition to the internal bandwidth, overhead in the processing pipeline (mainly waiting states) limits the throughput of the FPGA target. Again, it's to be noted, that the switch in use was originally designed for 1 Gbit/s line rate only and further optimizations are necessary for this configuration.

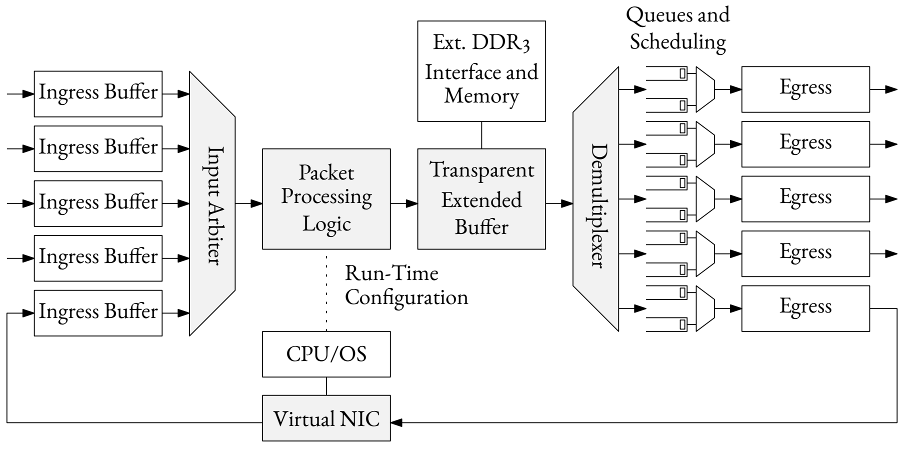


Figure 5: Simplified outline of the FPGA switch core

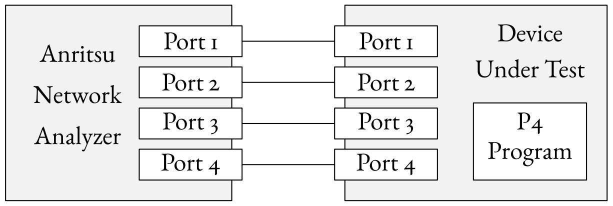


Figure 6: Measurement setup used for the performance evaluation. Link speeds were either 1 Gbit/s or 10 Gbit/s

D. Packet processing latency

Figure 8 presents the packet processing latency as a function of the time between two frames (or framegap). We notice that for framegaps larger than 1 µs the processing latency is of 24 µs for packet sizes of 1518 B for the software-based and network processor platforms. Since both approaches require copies of the frames from the network cards to the CPU, similar latencies are expected. For framegaps smaller than 1 µs the processing latency of the software-based platform increases up to 1 ms depending on the packet size. The packet processing is not able to keep up with the incoming rate and packets are buffered leading to the increased latency. Once the buffers are full, packets will be dropped, as already shown in Figure 7.

The network processor platform without CPU is able to better cope with the more intensive traffic, which can be explained by the fact that the processing is completely performed by the NPU, without CPU involvement or the need to copy packets.

The FPGA-based platform produces the best latencies, with values around 1.2 µs without buffering (i.e. for large framegaps). Once the internal bandwidth limit is hit, packets are buffered at ingress and eventually dropped. The latencies up to 15.8 µs observed in this region of small framegaps correspond to the capacity of the ingress buffers, which are much smaller compared to the software/network processor implementations and thus leading to smaller latencies in these situations, but potentially more packet losses during short, intense traffic bursts. However, in network systems with hard real-time requirements, buffering of packets is unintended in order to keep the worst-case latency short and predictable.

A comparison between the measured values on the three platforms and previous work [11] done on an industrial AFDX switch and an HP E3800 switch is presented in Table I. The gap in processing latencies for software-based P4 platforms compared to purely hardware-based ones still makes those platforms attractive for use-cases where latency requirements are less strict.

Switch	**Proc. latency**
Rockwell Collins AFDX switch	5 µs
HP E3800 without OpenFlow	7.2 µs
HP E3800 with OpenFlow	7.7 µs
HP E3800 with software switching	613 µs *(avg.)*
(Section V-A1) P4 software switch with DPDK	24 µs
(Section V-A2) P4 switch with NPU and CPU	24 µs
(Section V-A2) P4 switch with NPU and w/o CPU	5.8 µs
(Section V-A3) Switch with FPGA platform	1.2 µs

Table I: Comparison of packet processing latency of the evaluated P4 switch with numerical results from [11]

E. Profiling of software-based target

Software profiling was performed in order to better understand the software-based platform from Section V-A1. Figure 9 presents the profiling of the P4 program using the `operf`[5] statistical profiler for Linux. This tool enables us to evaluate how much time is spent in each function of the P4 program and the system. Note that only 2 cores of the CPU were used for this measurement, explaining the maximum value of 200 %. Three different function groups are presented in Figure 9:

- P4 primitives: *Parser*, *Table* and *Actions*, which is the part taking the most resources (up to 70 %);
- DPDK primitives: *Ethernet Driver*, *Ethernet Library* and *Run Time Environment (RTE)*, which take relatively low resources compared to the P4 primitives;
- Other functions: the C standard library (`libc`), Linux kernel, and overhead.

[5] http://oprofile.sourceforge.net

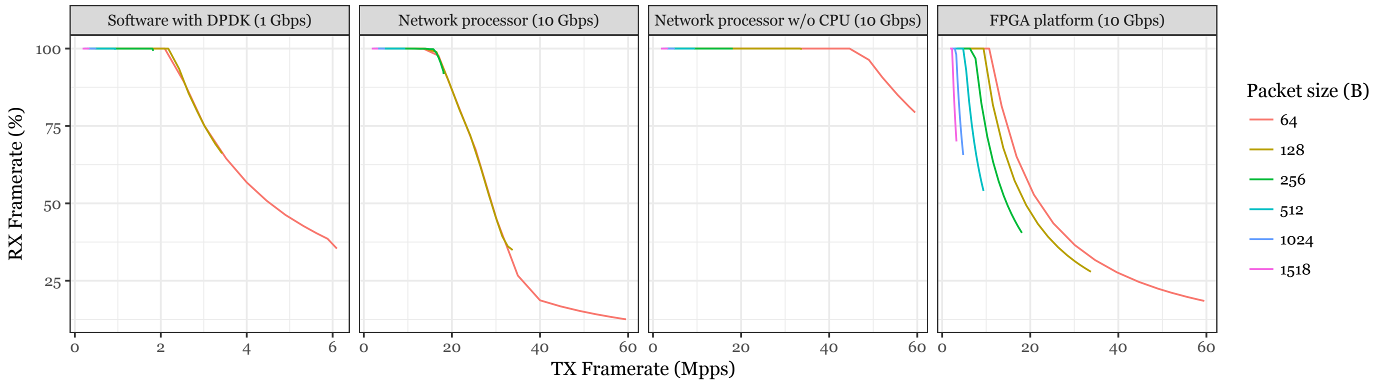


Figure 7: Framerate processed by the P4 switch

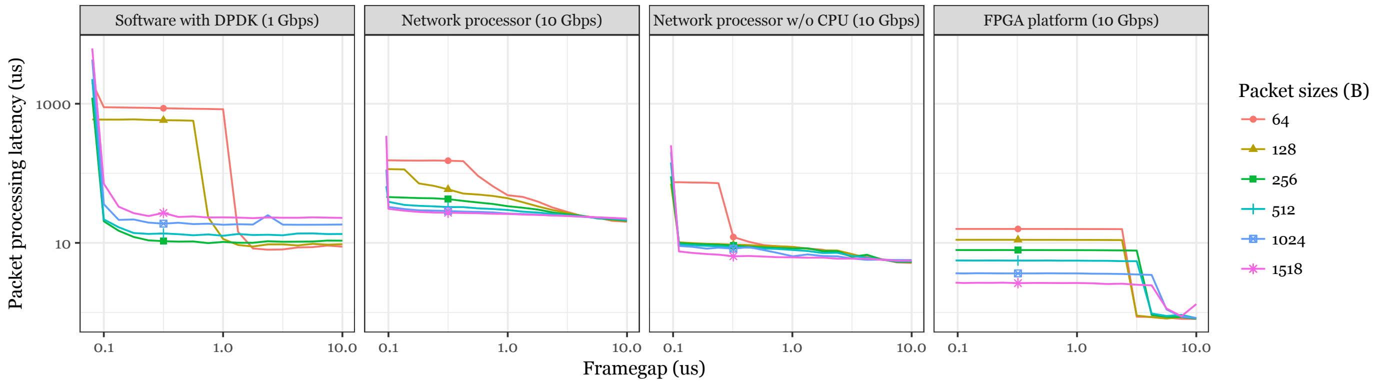


Figure 8: Packet processing latency as a function of the time between two frames

While in the tested setup the P4 program is able to almost fully process the 4 Gbit/s of traffic as shown in Section V-C, some work on reducing the P4 resources must be done in case additional ports would be used.

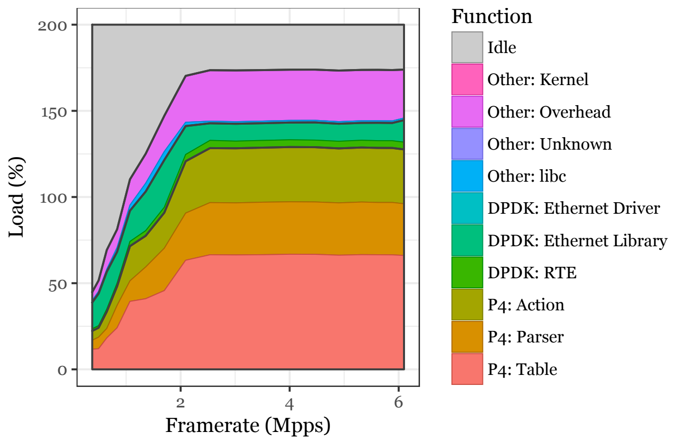


Figure 9: Profiling of the Linux computer with 64 B packet sizes

VI. Conclusion

We investigated in this paper recent developments made in the area of network-specific reconfigurable hardware and associated description language, namely the P4 programing language. This new development allows faster development of customized packet processing devices such as Ethernet switches or routers without the need for a deep knowledge about the target hardware architecture.

We presented in this paper P4 and its functionalities. We showed that its high flexibility in combination with simple building block enabling a formal analysis make it an attractive platform for some network protocols used in aeronautical use-cases. While some features such as definition of advanced egress packet scheduling and methods for time-based or time-triggered protocols are still lacking for more advanced network protocols, recent additions to the language in $P4_{16}$ and the Portable Switch Architecture make it an attractive platform. A case study of implementing AFDX was also performed in order to demonstrate that aeronautical protocols may be implemented using P4.

A performance evaluation of a simple P4 program was carried out on three different platforms: purely software-based target based on Intel DPDK, a hardware network accelerator based on a Network Processor Unit, and a FPGA-based platform. A comparison with an aeronautical and a COTS switch showed that while hardware based platform outperform software-based solutions on processing latencies, the difference between software and hardware solutions would be acceptable in some applications.

Since P4 is still under active development, with changes adding incompatibility between its different version, it is not yet ready for production use in aeronautical use-cases with long lifetimes. Nevertheless, initiatives such as the Portable Switch Architectures lead the way to standardized capabilities, meaning more stability in the future. Such platform is of high importance in the area of prototyping where functional validation and some indication about performance evaluation are necessary. Its simple cost model and associated formal analysis also make it a good target for future certification of packet processing devices.

Future work will include more in-depth studies of P4 and its platforms, with possible evaluations and deployment in services where frequent update is necessary, such as passenger connectivity. Further, TSN techniques like stream reservation and bandwidth allocation using the dedicated scheduling algorithms in conjunction with P4 will be investigated. Another area of interest would be examining methods, models and associated tools for certification of such approaches.

Acknowledgements

The authors would like to thank AED Engineering GmbH for providing access to the FPGA switch core and Netronome as well as its Open-NFP community for their support and fruitful discussions on the Agilio CX SmartNIC.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[2] Intel, "Intel DPDK: Data Plane Development Kit," Accessed 2017/10/24. [Online]. Available: http://dpdk.org

[3] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[4] G. Brebner and W. Jiang, "High-Speed Packet Processing using Reconfigurable Computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, Jan. 2014.

[5] R. Giladi, *Network Processors: Architecture, Programming, and Implementation*. Morgan Kaufmann, 2008.

[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013, pp. 99–110.

[7] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane," in *Proceedings of the 2nd ACM SIGCOMM workshop on Hot topics in Software Defined Networking*, 2013, pp. 127–132.

[8] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, Apr. 2012, pp. 141–154.

[9] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems," *Proceedings of the 14th International Conference on Networks (ICN 2015)*, pp. 78–83, Apr. 2015.

[10] D. Henneke, L. Wisniewski, and J. Jasperneite, "Analysis of Realizing a Future Industrial Network by Means of Software-Defined Networking (SDN)," in *12th IEEE World Conference on Factory Communication Systems (WFCS 2016), Aveiro, Portugal*, Aveiro, Portugal, May 2016.

[13] ——, "P4$_{16}$ Language Specification," Version 1.0.0, May 2017. [Online]. Available: http://p4.org/spec/

[11] P. Heise, F. Geyer, and R. Obermaisser, "Deterministic OpenFlow: Performance Evaluation of SDN Hardware for Avionic Networks," in *Proceedings of the 11th International Conference on Network and Service Management (CNSM)*, Nov. 2015, pp. 372–377.

[12] The P4 Language Consortium, "The P4 Language Specification," Version 1.0.4, May 2017. [Online]. Available: http://p4.org/spec/

[14] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. ACM, Oct. 2010, pp. 19:1–19:6.

[15] The P4 Language Consortium, "P4→NetFPGA: A low-cost solution for testing P4 programs in hardware," Accessed 2017/10/24. [Online]. Available: https://p4.org/p4/p4-netfpga-a-low-cost-solution-for-testing-p4-programs-in-hardware/

[16] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, Aug. 1989.

[17] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375–385, Jun. 1996.

[18] F. Geyer, S. Schneele, M. Heinisch, and P. Klose, "Simulation and Performance Evaluation of an Aircraft Cabin Network Node," in *Proceedings of the 4th International Workshop on Aircraft System Technologies*, ser. AST 2013, F. T. Otto von Estorff, Ed. Shaker-Verlag, Apr. 2013, pp. 323–332.

[19] The P4 Language Consortium, "P4$_{16}$ Portable Switch Architecture (PSA)," (draft), May 2017. [Online]. Available: http://p4.org/spec/

[20] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable Packet Scheduling at Line Rate," in *Proceedings of the ACM SIGCOMM 2016 Conference*, 2016, pp. 44–57.

[21] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet Transactions: High-Level Programming for Line-Rate Switches," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16, 2016, pp. 15–28.

[22] Aeronautical Radio Inc., "ARINC Specification 664P7-1: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network," Sep. 2009.

[23] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker," RFC 2698 (Informational), Internet Engineering Task Force, Sep. 1999.

[24] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, and M. Tejfel, "High Speed Packet Forwarding Compiled from Protocol Independent Data Plane Specifications," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 629–630.

[25] Netronome Systems, Inc., "Product Brief: Agilio® CX 2x10GbE SmartNIC," 2017. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[26] Xilinx Inc, "Zynq-7000 All Programmable SoC," 2017. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[27] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec 2014.

[28] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q—as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q—/Cor 1-2015)*, pp. 1–57, March 2016.

[29] Xilinx Inc, "SDNet Development Environment," 2017. [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdnet.html

[30] Netcope Technologies, a.s., "Netcope - P4 to VHDL," 2017. [Online]. Available: https://www.netcope.com/en/products/p4-to-vhdl

[31] Anritsu, "Data Quality Analyzer MD1230B," Accessed 2017/10/24. [Online]. Available: http://www.anritsu.com/en-US/test-measurement/products/MD1230B