An implementation of a deeply programmable SDN switch based on a hybrid FPGA/CPU architecture

Enio Kaljic
Department of Telecommunications
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
enio.kaljic@etf.unsa.ba

Almir Maric
Department of Telecommunications
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
almir.maric@etf.unsa.ba

Pamela Njemcevic
Department of Telecommunications
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
pamela.njemcevic@etf.unsa.ba

***Abstract*—Software-Defined Networks (SDN) simplify tasks performed by the network switches and centralize the network management by clearly separating networking processes into an application, control, and data plane. An OpenFlow, the flagship of SDN implementation, has reduced the innovation of such networks by diminishing switching tasks to the simple lookup of packet flow tables. A novel hybrid architecture of a deeply programmable packet-switching node (DPPSN), based on field-programmable gate array (FPGA) and central processing unit (CPU) technologies, is proposed, with the aim of overcoming OpenFlow's limitations regarding the ability to implement new protocols and advanced packet processing functionalities. It has been demonstrated, through the implementation and experimental evaluation of the DPPSN, that it is justified to use hybrid FPGA/CPU architecture for this purpose.**

Keywords*—*data plane; architecture; flexibility; programmability; deeply programmable network; software-defined networking

I. Introduction

Software-defined networking (SDN) represent a new trend in telecommunications which intends to simplify the process of managing the network. Unlike traditional networks built from complex network devices which perform multiple tasks and whose management is in nature distributed, the architecture of SDN clearly recognizes three planes: an application, a control, and a data plane. The role of the application plane is to execute network applications for the quality of service (QoS), security, load balancing, etc. The control plane regulates the rules for the entire network based on the requests generated by network applications. Based on the set rules, the controller configures the switches in the data plane. The role of the switch in the data plane is simple – to forward packets based on the instructions given by the controller.

Although standardization of SDN began with the proposal of ForCES in RFC3654 [1], SDN emerged five years later with the introduction of OpenFlow [2]. According to ForCES specifications, the switch consists of a control element (CE) and a forwarding element (FE). This is not entirely compliant with the idea of SDN, according to which the control and the data plane are clearly separated and possibly executed on different devices. As described in RFC5812 [3], FE realizes the functionality of packet forwarding by the interconnection of logical function blocks (LFB). LFB represents a single data plane functionality such as address resolution protocol (ARP), Internet control message protocol (ICMP), longest prefix matching (LPM), etc. On the other hand, the control plane of OpenFlow consists of one or more controllers executed on independent servers, and the data plane contains simple switches. The OpenFlow switch is a pipeline structure built from the packet header parser, flow tables, and associated actions. Packets are classified into flows based on packet headers, and flows are handled according to the instructions given by the controller. The instructions are given in the form of table flow records. Given the flow-level granularity, it is very difficult to implement network functionalities which are executed on the packet level (e.g. ARP). Additionally, the OpenFlow switch programmability is limited to the level of the flow table configuration. That limits the ability to implement new protocols, advanced packet processing functionalities such as encryption, transcoding, deep packet inspection, etc. Motivated by mentioned OpenFlow switch limitations, a lot of research, as shown in surveys [4]–[8], is trying to solve the data plane programmability issues in a variety of ways.

By sharing the same motivation, a new model of deeply programmable packet-switching node (DPPSN), based on the hybrid architecture, is proposed and implemented in this paper. The hybrid architecture is chosen because the field-programmable gate array (FPGA) technology allows achieving high-speed processing while the use of the cental processing unit (CPU) provides complete flexibility in the implementation of advanced packet processing functionalities.

The content of this paper is organized as follows. A review of related work which attempted to solve the problem of data plane programmability in SDN by using hybrid FPGA/CPU architectures is given in Section II. Then, in Section III, a novel architecture of DPPSN and its experimental implementation are presented. In Section IV, an experimental evaluation of DPPSN through various tests and measurements is given. Finally, concluding considerations are given in Section V.

II. Related work

SwitchBlade, a hybrid FPGA/CPU platform for the realization of custom protocols on the programmable hardware, is introduced in [9]. The pipeline of SwitchBlade includes

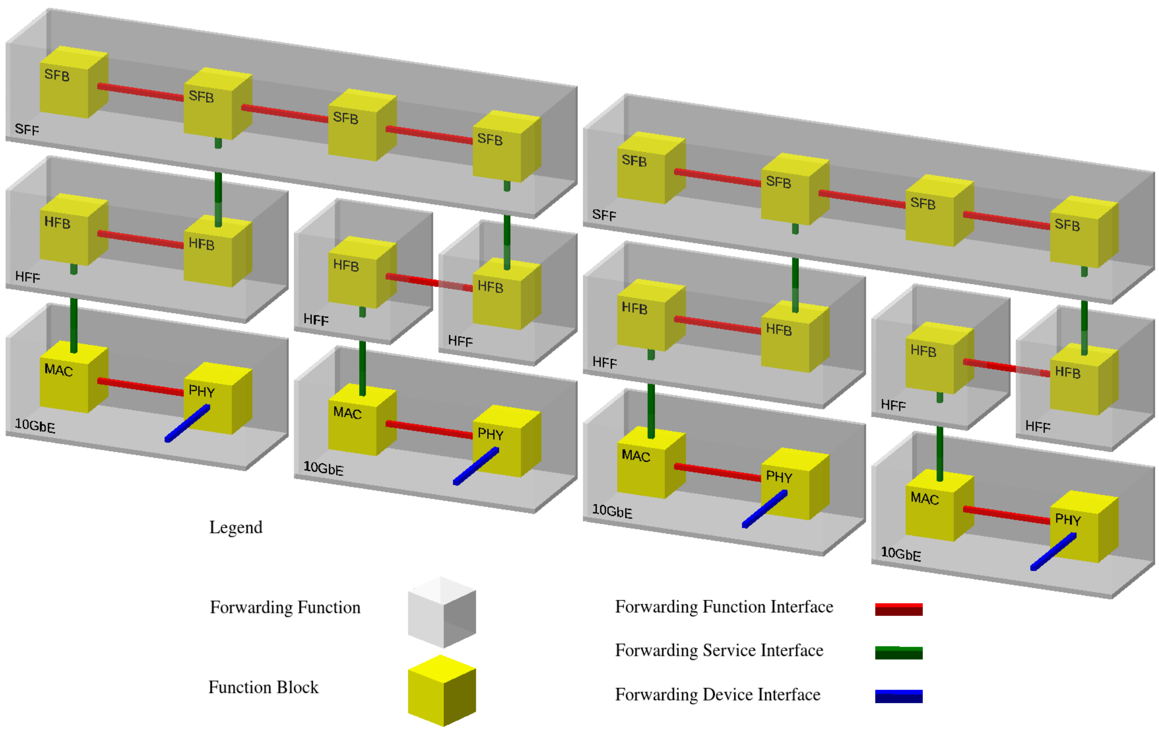
SFB
SFF
HFB
HFF
MAC
PHY
10GbE
Legend
Forwarding Function
Function Block
Forwarding Function Interface
Forwarding Service Interface
Forwarding Device Interface

Figure 1. DPPSN architecture

customizable hardware modules that can be combined to support new protocols. For more complex tasks which can not be implemented on hardware, there is support for software-based processing according to packet rules or flow rules. Another example of a hybrid FPGA/CPU platform is C-GEP, presented in [10]. C-GEP provides a flexible implementation of network nodes with the support for different types of network applications. Packet forwarding tasks perform the Virtex-6 FPGA chip on the C-GEP system board, while the COM Express PC is embedded to perform SDN control functionalities.

A system on a chip (SoC) platforms can also be seen as hybrid FPGA/CPU platforms. One such example is the ONetSwitch [11], a programmable platform based on the Xilinx Zynq-7045 SoC which includes the Kintex-7 FPGA programmable logic (PL) and the dual-core ARM Cortex-A9 processor system (PS) within the same chip. PL performs a flow table lookup, and in the case when there is no matching, software running on PS is used. Similar to ONetSwitch, a software-hardware co-design of the OpenFlow switch consisting of a software agent running on the PS and the hardware-based data plane in PL is proposed in [12].

These approaches are based either on the predetermined hardware-based pipeline structure or on the rule-based balancing of tasks between software and hardware. Also, these architectures are not hardware vendor independent. Due to all of the above, we propose a new architecture of the SDN switch, which is hardware vendor independent and offers complete flexibility in the organization of data plane functionalities.

III. Deeply Programmable Packet-Switching Node

In this section we introduce a novel architecture of a deeply programmable packet-switching node which can be used to build a fully flexible data plane in SDN. Then, taking into account the constraints of the chosen target platform, we present the implementation of an experimental SDN switch based on the DPPSN architecture.

A. *DPPSN architecture*

The architecture of the DPPSN, shown in Fig. 1, is a hybrid architecture based on the software part executed on general purpose CPU and the hardware part on the FPGA. Although architecture is inspired by ForCES, which realizes data plane functionalities through the interconnection of LFBs, it still introduces a novel approach to treating the problem of flexibility through three domains:

1) forwarding functions domain – treats the realization of forwarding functions (FF) by using function blocks (FB) and the forwarding function interface (FFI),
2) forwarding services domain – treats complex data plane functions, i.e. forwarding services (FS), and their interconnection through the forwarding service interface (FSI),
3) forwarding devices domain – treats the coupling of FBs and forwarding device's (FD) resources, and the usage of the forwarding device interface (FDI) for interconnection of FDs.

The new architecture provides complete flexibility in the way of realizing, configuring, and arranging FBs into FFs and FSs,

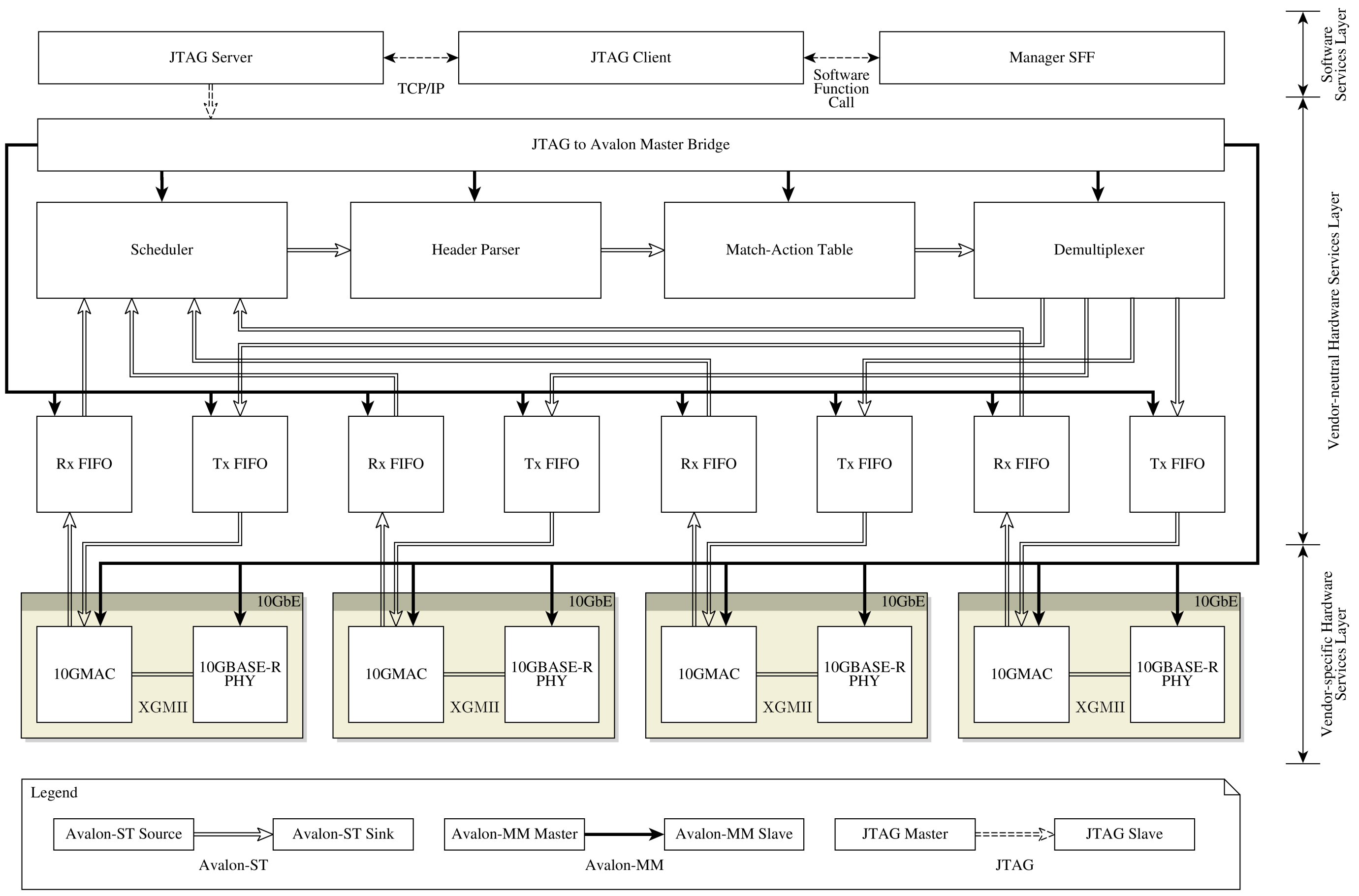

Figure 2. Architecture of experimental SDN switch based on the DPPSN

on one or more FDs. As the target platform has particular specificities in terms of communication interfaces and IP cores (e.g. for the realization of physical and MAC layer of 10Gbps Ethernet), the architecture is organized into three service layers:

1) software services layer – which implements a slow path of DPPSN using software function blocks (SFB) and software forwarding functions (SFF),
2) vendor-neutral hardware services layer – which implements a fast path of DPPSN using hardware function blocks (HFB) and hardware forwarding functions (HFF),
3) vendor-specific hardware services layer – which implements PHY and MAC protocols and interfaces.

B. Experimental SDN switch implementation

A platform based on the DE5-Net Stratix V GX FPGA was chosen for the implementation of the experimental SDN switch based on DPPSN. Similar to the NetFPGA-SUME [13], the DE5-Net development board with the help of Stratix V GX chip provides the ideal hardware solution for high-bandwidth and low-latency design. The Stratix V GX FPGA features 48 embedded transceivers that can transmit information at 12.5Gbps, allowing a direct low-latency connection to four external 10G SFP+ modules.

The experimental SDN switch, shown in Fig. 2, contains all the elements of the proposed DPPSN architectural model with three service layers. The link between the host and the FPGA was achieved via the USB-Blaster JTAG as an FSI. Although the JTAG interface was created in the 1990s as an industry standard for design verification and testing of printed electronic circuits, it is now used as an efficient interface for connecting microprocessor systems and peripheral hardware. For the FFI interconnection of HFBs and HFFs within the architecture, the Avalon [14] interface family was selected. Considering that the DE5-Net Stratix V GX FPGA has four XFI/SFP+ interfaces, 10Gbps Ethernet is supported as an FDI on a vendor-specific hardware services layer.

1) Software services layer: UML class diagram representing the architecture of the software services layer is shown in Fig. 3. Communication between SFBs of different FDs is envisaged by writing and reading shared memory. Each SFB implementing the FDI interface must comply appropriate shared memory access control protocols to prevent possible conflicts. Data transmission via the FFI interface is realized by direct calls of software functions *ffi_send* and *ffi_receive* from SFB instances. Given that Intel does not provide support for the direct implementation of JTAG interfaces in the form of shared or dynamic libraries, JTAG communication is realized

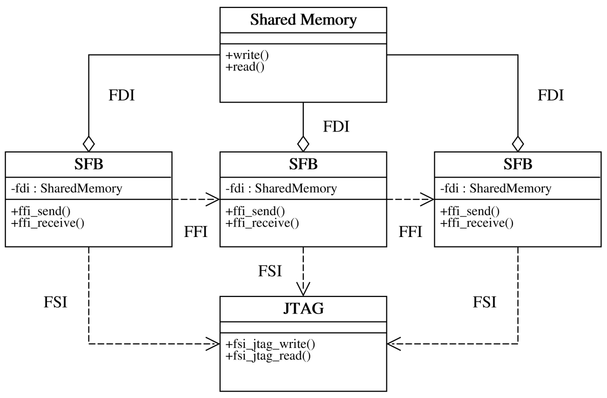

Figure 3. Software services layer architecture

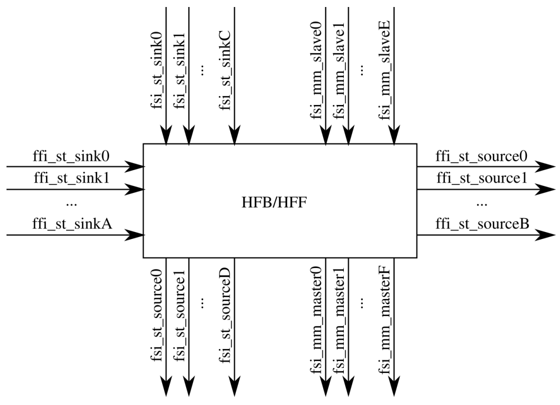

Figure 4. HFB/HFF interconnection interfaces

through the JTAG client and server. On one side, the JTAG server communicates with the FPGA chip via the USB-Blaster JTAG interface, and on the other with the JTAG client via a local TCP/IP connection. The FSI is implemented as a JTAG client application which supports writing and reading data from an FPGA chip by using two software functions: *fsi_jtag_write* and *fsi_jtag_read*.

2) Vendor-neutral hardware services layer: As already explained above, the link between the software service layer and the hardware service layer was realized using the JTAG interface. Since HFBs and HFFs use the Avalon interfaces, the conversion of JTAG and Avalon interfaces was implemented in a vendor-neutral hardware services layer. The conversion was made using Intel's IP core JTAG to Avalon Master Bridge. In the interconnection of HFBs and HFFs there are two ways of transmitting information:

1) a transfer of control and status information using Avalon Memory-Mapped (Avalon-MM) interface and
2) a transfer of packets using the Avalon Streaming (Avalon-ST) interface.

The basic functionalities of the experimental SDN switch were implemented in the vendor-neutral hardware services layer:

- ingress queuing,
- parsing of packet headers,
- management and lookup of forwarding table,
- egress queuing,
- management of ingress and egress scheduling processes.

Ingress and egress queues were implemented using Avalon-ST Single Clock FIFO core. Filling and emptying queues are performed via the Avalon-ST interface, while the Avalon-MM interface is used for queue length monitoring. The scheduling of ingress queues is configurable via the Avalon-MM interface. One of three possible scheduling disciplines can be selected: priority scheduling, round-robin scheduling and deficit round-robin scheduling. For deficit round-robin scheduling it is possible to set the quantum value for each queue.

The parser of packet headers, according to the configuration set via the Avalon-MM interface, extracts the selected octets from the packet header and joins them to the out-of-band signaling channel on the output Avalon-ST interface. The lookup of TCAM-based forwarding table performs according to the content of the out-of-band signaling channel. Manager SFF from the software services layer updates the content of the forwarding table via the Avalon-MM interface. The forwarding table also implements simple statistics as a match counter for each record from the table.

Demultiplexer dispatches packets to egress queues according to the instructions from the out-of-band signaling channel on the input Avalon-ST interface. The dispatching can be done into multiple egress queues (i.e. multicast), or packet can be kept in the buffer accessed by Manager SFF via the Avalon-MM interface. In this way, a packet connection between the layers of hardware and software was established.

3) Vendor-specific hardware services layer: FB for 10Gbps Ethernet was implemented using Intel's IP cores for 10GBASE-R PHY and 10GMAC. 10GBASE-R represents the implementation of the physical layer of the 10Gbps Ethernet link and is defined by clause 49 in the IEEE802.3-2008 specification. 10 Gigabit Media-Independent Interface (XGMII) connects 10GBASE-R PHY with MAC and Reconciliation Sublayer. An XFI/SFP+ interface connects 10GBASE-R PHY with a fiber optic module.

Through the Avalon-MM interface, it is possible to control the physical and MAC layer protocols in the sense of:

- setting the status of the network interface,
- settings the MAC address filter on each network interface,
- tracking various statistics about the type (e.g. correct, with error, unicast, multicast, broadcast) and quantity (e.g. number of octets, number of frames, distribution by frame size) of received and sent frames.

IV. Experimental evaluation of DPPSN

In this section we present an experimental evaluation of the novel DPPSN architecture through various tests and measurements performed over the experimental DPPSN.

A. Testing scenarios

To test an experimental SDN switch, a testbed was created as shown in Fig. 5. In the implemented testbed, the following testing scenarios were conducted:

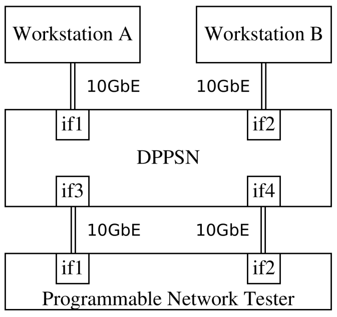

Figure 5. Experimental testbed

1) determination of reconfiguration latency of DPPSN,
2) determination of forwarding latency of experimental switch in relation to the reference switches,
3) determination of experimental switch throughput in relation to the reference switches.

The reconfiguration latency of the DPPSN is the time needed to completely change the functionality of DPPSN. Reconfiguration is achieved by programming a new bitstream into the configuration memory of the FPGA chip and by changing the software application on the host. Two computer workstations were connected to the experimental switch to determine the reconfiguration latency of the DPPSN. Then, using the ICMP-based ping software tool, the interruption time between workstations during full reconfiguration of the DPPSN was measured (i.e. reconfiguration latency). Since the ping tool does not allow a direct measurement of the connection interruption time, this is achieved indirectly by measuring the number of lost packets of CBR traffic within a fixed time window.

A programmable network tester FlueNT10G [15] was used to measure the forwarding latency and throughput of the experimental switch. The FlueNT10G is implemented on the NetFPGA-SUME development board, enabling precise generation and recording of traffic on 10Gbps network interfaces. Using the precision time protocol (PTP), it allows measurement with a resolution of 6.4ns. The architecture of the FlueNT10G consists of hardware design on an FPGA chip, software libraries, and measurement applications. The hardware part of the architecture contains a precise generator of Ethernet traffic that is previously recorded or synthesized and an Ethernet traffic receiver with the ability to accurately measure latency for each received packet. The software part of the architecture contains libraries that allow programmed execution of repeatable network experiments.

B. Measurements

In this subsection, we present and discuss the results of the tests carried out according to the described scenarios.

1) Reconfiguration latency: On workstations A and B that are connected via a 10GbE interface to the experimental switch, IP addresses 192.168.1.1 and 192.168.1.2 are set, respectively. Then using the command terminal of the workstation B, the ping tool was started, by which 600 packets of 64 octets at a rate of 10 packets/second were generated to the station A. At an arbitrary time (e.g. 23 seconds after the ping tool was started) a DPPSN reconfiguration procedure was initiated. At that point, there was a break in the connection between the workstations A and B. Upon completion of the DPPSN reconfiguration, the connection was re-established. After sending the last package, the ping tool generated statistics from which it was found that the total duration of the test was 60.818 seconds and that the percentage of lost packets was 19%. Given that CBR traffic is generated, the reconfiguration latency of DPPSN can be determined as a product of the duration of the test and the percentage of lost packets. Therefore, the reconfiguration latency, in this case, was 11.65678 seconds.

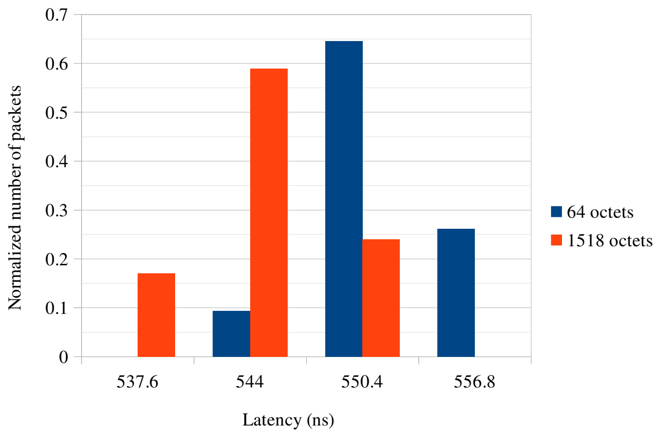

Figure 6. Forwarding latency histogram for 10Gbps CBR traffic load

2) Forwarding latency: The forwarding latency of the experimental switch was measured using a programmable network tester connected to another two 10GbE interface interfaces of the switch. The application generated packets, with sizes of 64 octets in the first test and 1518 octets in the second test, at the first interface of the programmable network tester at a constant speed of 10Gbps for 10 seconds. Then the application performed a calculation of the latency histogram for received packets. In Fig. 6 the resulting latency histogram is shown.

In the second case, the application generated packets in 13 tests with sizes ranging from 64 octets to 1518 octets, also at a constant speed of 10Gbps. In each test, the mean latency calculation for the received packets was performed. Testing was also conducted on two commodity switches: Cisco SG350X-48 and Ubiquiti ES-48-500W. Since commodity switches were realized as store & forward, and the experimental switch as a cut-through, it was necessary to compensate for packet storage delay in store & forward switches so that they could be compared to the experimental switch. Packet storage delay is a multiplication of the packet size expressed in segments of 8 octets and the 6.4ns storage delay of one packet segment. The switching latency of the switch, depending on the packet size, at a constant load of 10Gbps is shown in Fig. 7.

By analyzing the test results, it can be determined that the

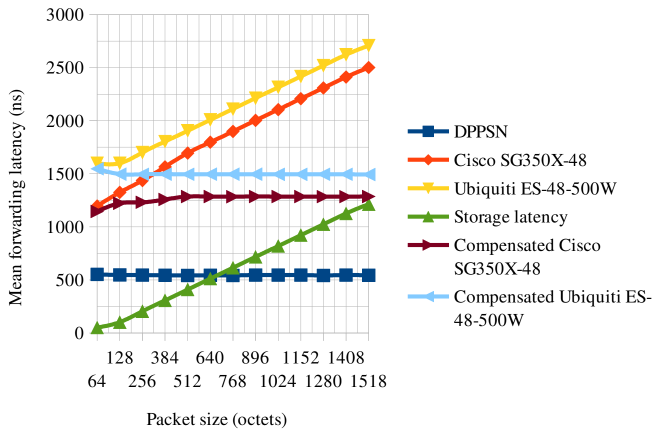

Figure 7. Forwarding latency for 10Gbps CBR traffic load

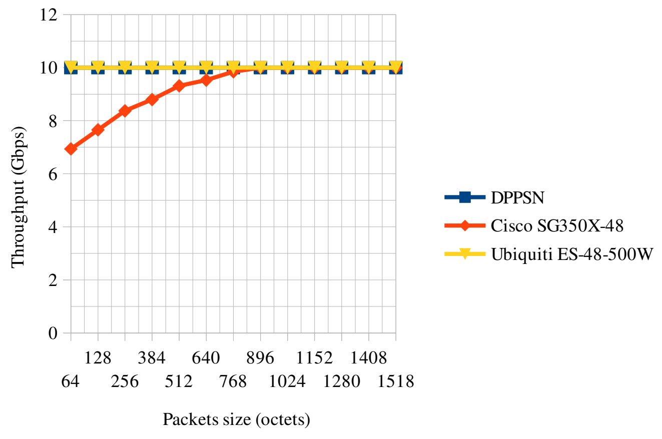

Figure 8. Achieved throughput for 10Gbps CBR traffic load

forwarding latency of the experimental switch is independent of the packet size, which is a consequence of the cut-through principle of operation. The mean latency of the experimental switch is approximately 545ns and is significantly less than the latency of commodity switches (1285ns and 1495ns). That is the consequence of simplified header parsing and forwarding table lookup operations according to principles of SDN.

3) Throughput: The throughput of the switch was measured using an application, which also generates packets in 13 tests with sizes distributed from 64 octets to 1518 octets. Fig. 8 shows the measured throughput of the experimental switch and two reference commodity switches.

Experimental results presented in this paper have shown that it is possible, based on the proposed architecture, to implement a working deeply programmable packet-switching node.

V. Conclusion

Although the concept of SDN has evolved through the emergence of OpenFlow, its over-simplicity has slowed down the innovation potential. In order to overcome the limitations of OpenFlow in terms of the ability to implement new protocols and advanced packet processing functionalities, a new hybrid FPGA/CPU architecture of a deeply programmable packet-switching node is proposed in this paper. Through the implementation and experimental evaluation of DPPSN, it has been shown that the use of hybrid FPGA/CPU architectures is justified. Moreover, it has been shown that the performances of the implemented experimental switch are comparable (or even better) with the performance of commodity switches, which encourages its use in modern telecommunication networks.

References

[1] H. M. Khosravi and T. A. Anderson, “Requirements for separation of IP control and forwarding,” RFC 3654, Dec. 2003, accessed: 2018-10-12. [Online]. Available: https://rfc-editor.org/rfc/rfc3654.txt

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] J. M. Halpern and J. H. Salim, “Forwarding and control element separation (ForCES) forwarding element model,” RFC 5812, Mar. 2010, accessed: 2018-10-12. [Online]. Available: https://rfc-editor.org/rfc/rfc5812.txt

[4] R. Amin, M. Reisslein, and N. Shah, “Hybrid sdn networks: A survey of existing approaches,” *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[5] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, “Advancing software-defined networks: A survey,” *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.

[6] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, “Reconfigurable network systems and software-defined networking,” *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1102–1124, July 2015.

[7] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[8] H. Farhad, H. Lee, and A. Nakao, “Data plane programmability in SDN,” in *2014 IEEE 22nd International Conference on Network Protocols*, Oct 2014, pp. 583–588.

[9] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster, “SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 183–194, Aug. 2010.

[10] P. Varga, L. Kovács, T. Tóthfalusi, and P. Orosz, “C-GEP: 100 Gbit/s capable, FPGA-based, reconfigurable networking equipment,” in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, July 2015, pp. 1–6.

[11] C. Hu, J. Yang, H. Zhao, and J. Lu, “Design of all programable innovation platform for software defined networking,” in *Presented as part of the Open Networking Summit 2014 (ONS 2014)*. Santa Clara, CA: USENIX, 2014.

[12] S. Zhou, W. Jiang, and V. Prasanna, “A programmable and scalable OpenFlow switch using heterogeneous SoC platforms,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’14. New York, NY, USA: ACM, 2014, pp. 239–240.

[13] J. Cao, X. Zheng, L. Sun, and J. Jin, “The development status and trend of NetFPGA,” in *2015 International Conference on Network and Information Systems for Computers*, Jan 2015, pp. 101–105.

[14] Intel, “Avalon interface specifications,” Sep 2018. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

[15] A. Oeldemann, T. Wild, and A. Herkersdorf, “Fluent10g: A programmable fpga-based network tester for multi-10-gigabit ethernet,” in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 178–1787.