Deterministic OpenFlow: Performance Evaluation of SDN Hardware for Avionic Networks

Peter Heise*†, Fabien Geyer* and Roman Obermaisser†
*Airbus Group Innovations, Munich, Germany
†University of Siegen, Siegen, Germany
Emails: {peter.heise@airbus.com, roman.obermaisser@uni-siegen.de}

***Abstract*—Due to special requirements avionic networking devices are typically quite expensive. One way to reduce costs is to make use of commercial off the shelf devices and configure them in a way that gives similar performance. In this paper we evaluate the use of OpenFlow in the avionics environment in terms of performance and configuration. The main feature of OpenFlow is fine-grained access to the switch's forwarding plane. While it was primarily designed to offer high configurability and reduction of cost through harmonization of interfaces, in newer versions OpenFlow added support for traffic policing. In OpenFlow this is realized with meters that allow for quality of service enforcement on a hardware level as well as an arbitrary mapping of meters to flows. This paper shows how to make use of OpenFlow's meter commands to achieve deterministic behavior and discusses its advantages and shortcomings. We then implement the proposed solution on a commercial off the shelf OpenFlow switch and compare the switching performance to a state of the art avionics switch used in current aircraft.**

Keywords: OpenFlow; Meter Band; Traffic Policing; Avionics Full-Duplex Switched Ethernet (AFDX); Avionics

I. Introduction

OpenFlow is a protocol that allows for fine-grained access to the switch's forwarding plane with the intention of taking the logic out of the switch so that hardware costs can go down. Mechanisms to achieve quality of service were not available in early versions, however, with OpenFlow 1.3 [1] a traffic limitation mechanism was introduced that potentially allows the design of safety critical networks.

The main task of safety critical networks is to provide guaranteed delivery of all packets. Such a network must ensure that packets are delivered before a deadline and not lost even in case of the failure of one element of the network. When speaking about deadline guarantee for packets, often the term deterministic is used. For a network architecture to be called deterministic, it must fulfill the following points: (a) formal verification of maximum end-to-end latencies and (b) mechanisms in the network to guarantee that ill-behaved end-systems will not interfere with well-behaved end-systems. This is generally achieved by the definition of a contract on the flows of the network, which defines how an end-system must send its packets on the network. An example of such contract is the Virtual Link in the Avionics Full-Duplex Switched Ethernet (AFDX) [2] technology, where packet sizes are limited as well as the time between two packets.

This paper shows how to map the mechanisms seen in AFDX onto an OpenFlow switch to allow the use of commercial off the shelf (COTS) hardware for a cheaper overall network. The contribution of this paper is twofold: we show how to achieve bounded latencies with OpenFlow by applying metering for each packet and second we implement and evaluate the concept on an *HP E3800* COTS switch. Finally the COTS hardware is compared to a state of the art AFDX switch currently in use in aircraft.

The outline of this paper is as followed. We first look at the background and related work in Section II and III. In Section IV, we introduce the concept to achieve bounded latency with OpenFlow and sketch the means for a mathematical model. Then we present in Section VI some synthetic measurements with the COTS switch and finally compare the behavior to a real aircraft switch in Section VII. Finally Section VIII summarizes and concludes our work.

II. Background

A. Avionics Full Duplex Ethernet (AFDX)

The technology used in aeronautics to build safety critical networks is Avionics Full Duplex Ethernet [2]. AFDX offers real time guarantees through statistical multiplexing without the need for time synchronization. All packets are mapped to a flow called the virtual link (VL). Virtual links are unidirectional flows from one end-system to one or more end-systems. Identification of VLs is done via the packet's destination MAC address as seen in Fig. 1. The upper 32 bits are constant while the lower 16 bits represent the virtual link. Each VL is characterized by a *Bandwidth Allocation Gap* (BAG), the time between two consecutive packets of the same flow, and a maximum (s_{max}) and minimum (s_{min}) frame size. Switches enforce compliance to this traffic description and can thus offer a guaranteed service time. Network calculus is then used to calculate the worst-case latency through the whole network.

48 bit MAC Destination Address	
Constant Bitfield	Virtual Link
32 bits	16 bits

Fig. 1. AFDX Destination MAC Structure [2]

For applications with higher reliability demand, a second network is available. The two networks are called network A and network B. End-systems running critical applications are connected to both networks via two network cards and submit the packets once via each network. Deduplication only takes

978-3-901882-77-7 © 2015 IFIP

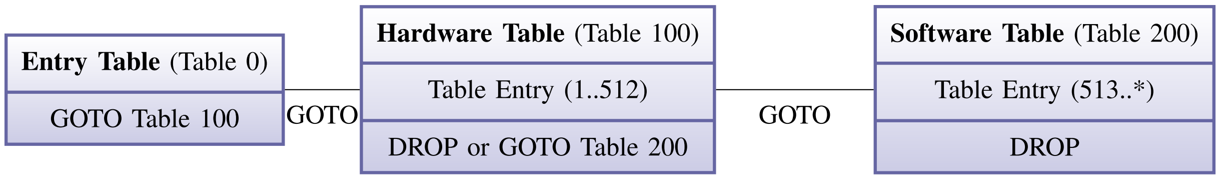

Fig. 3. OpenFlow 1.3 Table Model and Virtual Link Matching

MAC-Dest	MAC-Src	Ether-Type	Payload	SN	FCS

Fig. 2. Ethernet AFDX Packet Structure [2]

place at the receiving end-system and is transparent for the network. To identify the second packet a sequence number at the end of the packet payload is used as shown in Fig. 2.

B. OpenFlow

OpenFlow [1] is a communication protocol that controls a switch's forwarding behavior. It is running between a controller and multiple switches. Packets can be matched on different fields (i.e. destination MAC address) and then associated to an action or a set of actions. Actions include forwarding to ports, but also rewriting of certain parts of the packet like TTL or VLAN tags. In version 1.0 of the OpenFlow protocol the action set is modified by a single matching table entry and the action list associated to this match. In versions after 1.1.0 the action set is modified by instructions. Whenever a packet matches a table entry, one or several instructions are associated to this table entry. An instruction may then update, add or overwrite the action set of each packet.

Also starting with version 1.1.0 multiple tables were introduced. The example in Fig. 3 is based on the switch we had at hand. Packets arrive at the switch and will first be matched by a wild card entry in table 0. This wild card entry has an instruction with an immediate GOTO action which restarts the matching in table 100. Usually two kinds of tables will be present at a switch, one (or several) tables that match and forward packets on a hardware level and another (or several) tables that do the matching and forwarding in software. While hardware based matching will offer better throughput, its capabilities and matching entries are limited. Whenever more advanced capabilities are needed the software based matching can do so at the cost of losing hardware acceleration and thus performance.

Beginning with version 1.3.0 METER actions were introduced. A meter is for example a simple token bucket policer that can be instantiated and configured to a certain rate and burst. Whenever a flow exceeds the bucket's rate, the packet is dropped. If the packet complies with its traffic definition and the burst is not exceeded, the remaining actions in the action set will be executed. While the main application of meters is to rate limit packets sent to the controller it can also be reused to achieve quality of service behavior which is what is investigated in this paper.

III. Related work

Sonkoly et al. [3] propose a framework to provide quality of service (QoS) in the Openflow enabled Ofelia testbed. The framework implements controllers that are aware of vendor-specific extensions of the switches in use to allow the use of different vendors.

Seddiki et al. [4] introduce FlowQoS with per flow bandwidth guarantees on a user's home router. It aims for an end-user style application where controllers dynamically install QoS rules on switches upon user requests for services like video or voice over IP with decisions and rules based on current bandwidth utilization. The solution is implemented by instantiating a two-switch virtual topology based on Open vSwitch and the use of different queues on the vSwitches. The switches are controlled by OpenFlow.

Egilmez et al. [5] propose a new OpenFlow controller design supporting end-to-end quality of service. The controller continuously collects statistics from the switches to then dynamically place new requests on the correct path. As the controller only reacts on occurring congestion, such a monitor and react solution will not be able to handle hard guarantees.

He. et al. [6] conducted a measurement study on OpenFlow switches, measuring the latencies between an incoming packet that is not matching any rule towards the controller and installation of rules. They concluded, that the time needed for installation of new rules vastly depends on the implementation details of the specific switches and propose means on how to reduce the installation times.

Azodolmolky et al. [7] present an analytical model based on network calculus to determine an upper latency and buffer sizes in a controller-controlled switch. The model assumes that packets have to go into the controller first and then be forwarded on the switch. To prevent a single point of failure we assume to not have any controller interaction during operation, therefore approach in our paper can be based on normal, OpenFlow unaware, network calculus.

Jacobs et al. [8] studied the use of Gigabit Ethernet switches in aircraft networks and concluded, that the evaluated Gigabit Ethernet switches offer reliable low latency switching in over-provisioned or slightly under-provisioned scenarios.

Research regarding implementation of AFDX with COTS hardware has been performed in [9]. A commercial single-board computer was used running VxWorks with modified network drivers. The send and receive functions to access the driver are handled in a specific pseudo-partition of VxWorks, to allow communication from application partitions to the network. Only end-systems are considered and no switches.

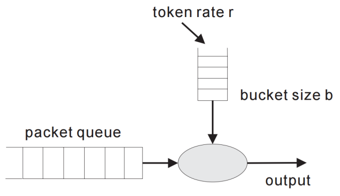

Fig. 4. Token Bucket Mechanism

IV. Deterministic OpenFlow

The way bounded latencies are achieved in AFDX is based on two steps: (1) Mapping of each incoming packet to a specific flow based on its MAC destination address and (2) assuring conformance to a predefined traffic behavior for this flow. If a packet can't be mapped to a flow in (1) or exceeds the rate allowed in (2), the packet will be dropped before entering the switch's backplane. This way only conformant packets are allowed in and with knowledge about the switch's line speed the maximum queuing time and buffer size can be calculated by using network calculus.

A. Token Bucket in AFDX and OpenFlow

The OpenFlow 1.3 [1] standard allows to instantiate meters with different types. One of the types in the standard is a simple rate limiter called *OFPMBT_DROP*. The DROP band models a token bucket behavior which allows to drop any packet that exceeds a certain flow's rate. It can calculate this rate either in kilobits/sec or in packets/sec.

The DROP band is based on the token bucket in Fig. 4 which works as follows. Tokens get added at a specific token rate r into a bucket with the size b. Whenever a packet arrives at the token bucket, it is allowed to consume all the tokens in the bucket.

If not enough tokens are available, the packet can either be delayed or dropped. For a bounded latency packets will be dropped, as packets not following their traffic contract will likely not do so in the future. The advantage of token buckets against other schedulers is its independence of a global time base as well as it allows for some burstiness in the traffic behavior.

Mathematically the traffic constrained by a token bucket can be modeled using network calculus [10] (NC). In NC traffic flows are modeled as $\alpha_{r,b}(t) = (rt + b)$ for $t > 0$ with $r = \frac{s_{max}}{BAG}$ and $b = s_{max}$ and s_{max} being the maximum packet size. A typical service curve β for a switch or router will usually look like $\beta_{R,T}(t) = R\lceil t - T \rceil^{+}$ with R being the output port bandwidth and T the static processing delay.

By applying NC the traffic characteristic of the output flow α^{*} through a switch offering a service curve β can then be calculated to according to

$$\alpha^{*} = \alpha\phi\beta(t) = \sup_{u \geq 0}\{\alpha(t+u) - \beta(u)\} \tag{1}$$

B. Deterministic behavior and end-to-end latencies

The above mechanism explains how to describe a single system. The service curse experienced through a series of concatenated systems can be described as $\beta_{R1,T1} \otimes \beta_{R2,T2} = \beta_{min(R1,R2),T1+T2}$. With that an end-to-end delay bound D_0 can be calculated as

$$D_0 = \frac{b}{R} + T \tag{2}$$

with $R = \min_i(R_i)$, $T = \sum_i T_i$ and b the burst of the arrival curve. This formula is known as the *Pay Bursts Only Once* theorem [10].

Besides the calculated traffic no other traffic is allowed on the network and all devices have to follow their traffic contract. Devices sending more packets than allowed will experience packet drops at the ingress. Such behavior is acceptable, as such devices can be considered broken and are not likely to be fixed by any means of the network itself.

C. Matching of Virtual Links

The matching of virtual links is based on the packet's destination MAC address. OpenFlow allows for exact and wildcard matching of destination MAC addresses. To separate the flows from each other an exact matching entry for each MAC address is needed, each mapped to an individual meter. To allow for AFDX's multicast behavior, multiple actions can be associated to each entry, e.g. by having multiple output actions.

D. Role of the OpenFlow controller

OpenFlow allows for installation of rules during the run-time of a switch. Such behavior is not desired in avionics, especially not during flight. While a single point of failure could be prevented by using multiple controllers, we assume the controllers only to install rules during a maintenance phase on ground and then leave all configuration as-is until the next maintenance. The controller will therefore be turned off during flight. The OpenFlow standard already supports this option by setting the switch to *fail-secure mode*. The switches will keep their configuration and unknown packets will be dropped.

V. Experiments and Results

The primary focus of this study is to see if available COTS switches can provide reliable low latency data transfer services and enforce traffic to guarantee such behavior even in failure cases. Therefore we compare a commercial off the shelf Hewlett Packard E3800 switch with a state of the art Rockwell Collins AFDX-3800 switch that is being used in current aircraft such as Airbus A380 and A350. The HP E3800 is running the latest firmware KA.15.16.0008 and offers support for OpenFlow 1.3. In the latter we will refer to the switches as COTS switch and AFDX switch.

As OpenFlow controller we use a modified version of NOX supporting OpenFlow 1.3 available on GitHub [11] and the utilities contained within this software. To measure latency and received packet rates an Anritsu MD1230B Ethernet Tester was used. The device offers network latency measurements with a precision better than $1\mu s$. The rules should be installed without no timeout and the switch's operational mode set to *fail-secure mode*. This way in case of controller failure, the switch will keep all installed rules and drop all non-matching packets.

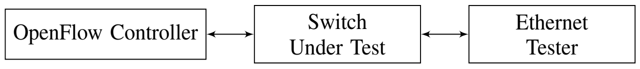

Fig. 5. Measurement setup used in Section V

In a first step we evaluate the impact and differences of hardware and software matching of rules on the COTS switch. The instructions will then be extended by actions for policing and the accuracy of such mechanism is checked. In a next step multiple policing meters will be activated and multiple flows matched. In a final step, the switch will be configured with a realistic airplane configuration and directly compared to the AFDX switch.

The following subsection presents and discusses results of our measurements. All measurements were done with the COTS switch set to 100 Mbps full-duplex mode for comparison to the AFDX switch. All measurements in this section used a measurement set-up as seen in Fig 5, where the Ethernet Tester is exclusively connected to the switch to measure loads.

A. *Hardware vs. software matching*

In a first step we evaluate the COTS switch's capabilities in basic switching. Therefore we instantiate a single matching rule without any meters to get information about the switch's line performance. We then change the amount of packets per second (pps). In Fig. 6 it can be seen, that the hardware based matching and forwarding works as expected and offers a constant static forwarding delay of around $7\mu s$.

Packets that are switched based on software-based matching experience a much higher latency. While those packets have to go through the hardware matching first and be forwarded to the switch's general purpose processor, put through a hash function and then send back down to the switching plane. The experienced latency is around 30 to 100 times

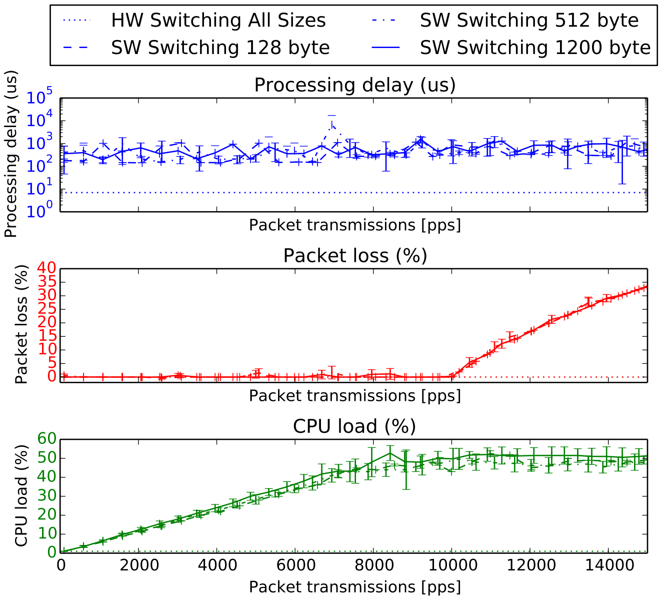

Fig. 6. Characteristics of forwarding for different packet sizes on COTS switch (errorbars indicate stdev of measurements)

higher than the hardware-matched rules and slightly increasing for higher pps values for the average case. Further the CPU load is slightly bigger for larger packets, which could be based on the internal hashing function in the switch that needs to process more payload. Also seen in Fig. 6 the load linearly increases up to 10.000 pps and then later converges at around 50% processor load.

This convergence is mainly based on the fact that packets are simply dropped above 10.000 pps at the hardware level instead of being forwarded to the software matching function. This seems sensible as the processor is also in charge of running maintenance and management tasks on the switch and without such a limit the switch would be an easy target for denial-of-service attacks.

A close-to-zero packet loss can only be observed at speeds of less than 2.000 pps on the whole switch. Between 2.000 and 10.000 pps some sporadic packet losses appear. Such behavior, however, might improve with further firmware updates for the software stack.

B. *Meter accuracy*

In a next step we evaluate the switch's capability of policing certain flows once matched by a hardware rule. Each meter is realized by a token bucket mechanism that needs configuration of a rate and a burst. In Fig. 7 it can be seen, that tested rates from 1 to 10.000 Kbps can be policed with the switch. The meter error was calculated according to Eq. (3).

$$e = \frac{|AllowedRate - ReceivedRate|}{AllowedRate} \tag{3}$$

The higher error in the lower meter rates can be explained by the granularity of the measurement using 64-byte packets being transmitted at full speed and the received rate being recorded.

The processor load is constant in idle, which indicates that the metering is certainly done on a hardware level and not

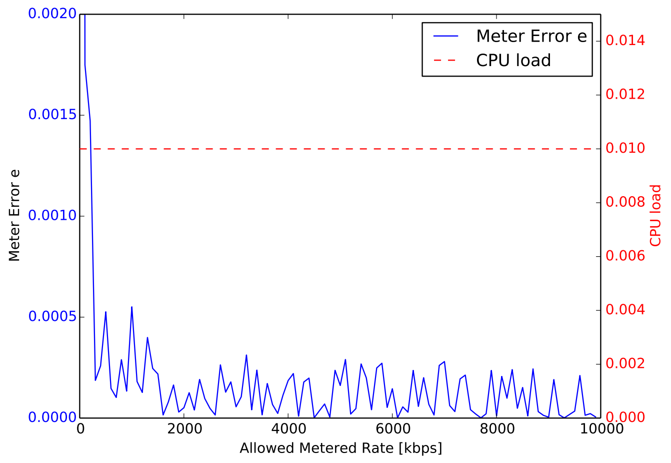

Fig. 7. Meter accuracy test on COTS switch

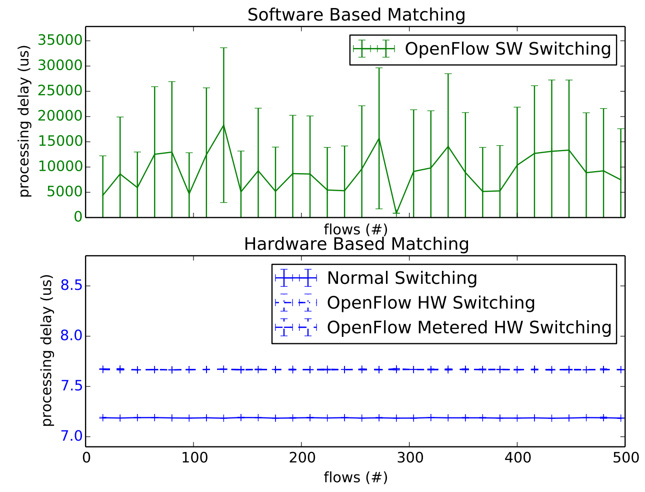

Fig. 8. Switching latency with multiple parallel flows

passed up to software. The COTS switch that was available allowed us to instantiate around 2000 meters which will be tested and used in the next section.

C. Multiple metered streams

The next section evaluates the switch's capabilities to handle multiple matching of flows including metering of those. The switch we had at hand was not able to map software matched flows to meters. The chart seen in Fig. 8 is composed of 4 separate measurements of the maximum latency seen.

The first measurements were done using *normal switching* without any OpenFlow features activated. We see a similar performance as seen before in Fig. 6. Removing the transmission delay, the switch offers a constant static processing delay of around $7\mu s$. No dependence on the number of flows can be seen, as the switch does not make a difference of the flows here and broadcasts all packets to all ports.

The next two measurements were done using OpenFlow HW matching, switching and metering of the matched flows. The latency again stays constant across the whole number of flows with an OpenFlow incurred delay of around $400ns$ compared to classical switching. No difference is seen between the use of meters and direct forwarding. Furthermore - not depicted here - no difference was seen between forwarding to one output port or several output ports for multicast purposes.

The last measurement in Fig. 8 concerns the software based matching of packets. The packets were limited to below 10.000pps in total because of the limitations seen in section V and only their diversity was increased. The maximum experienced latency is quite unreliable and a factor 30 to 6000 higher than hardware based ones.

D. Failing End-system

A failure in avionics networking is, for example a system that gets stuck in a loop and transmits data at a faster rate than expected. This failure is known in the literature as *babbling idiot*. To motivate the use of the proposed solution we evaluate

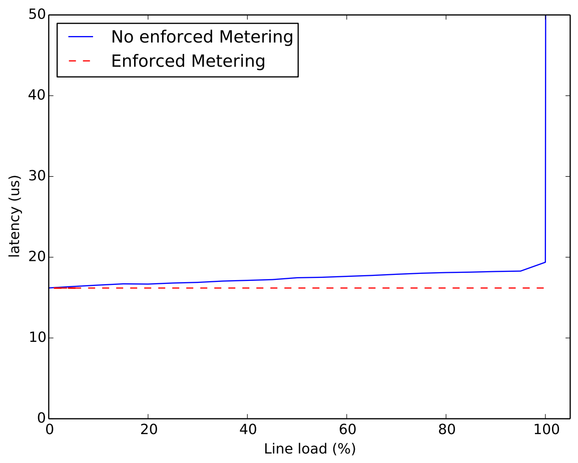

Fig. 9. Failing end-system comparison

the non-availability of quality of service enforcement in a real-time system and emulate a crashed end-system that is sending with up to line-speed. It can be seen in Fig. 9, that a failure which would lead to 100% line load affects all other traffic as well and would as such break the communication network. The only means to prevent such behavior is to disconnect the failing end-device from the network or block all its traffic.

VI. Realistic Scenario

In a last step we took a real aircraft AFDX switch configuration and tried to map it onto the COTS switch. We selected the switch with the highest load in a real aircraft configuration with its key parameters shown in Table I and II. Due to the limitations seen in the previous sections, only a mapping to the hardware matching was sensible but therefore also giving the limitation of being able to match 512 flows. The limitation to 512 matches however also meant, that the switch will not be capable to handle a real aircraft configuration yet.

Therefore, we reduced the configuration to the switches capabilities of 512 flows and ran tests from there to compare the COTS switch to the real AFDX switch. As seen in Fig. 10, we used a PC that replayed synthetically created PCAP files. The PCAP files were based on the switch configuration and

TABLE I. Realistic parameters used for measurement study

Number of VL on switch	649
Number of ports used on switch	19
Worst-case number of incoming packets	33736 pps
Number of VL destination ports	1 to 19
Traffic specification of VL	see Table II

TABLE II. AFDX configuration of A380 aircraft (from [12])

Bag (ms)	Number of VL
2	20
4	40
8	78
16	142
32	229
64	220
128	255

Frame length (bytes)	Number of VL
0-150	561
151-300	202
301-600	114
601-900	57
901-1200	12
1201-1500	35
>1500	3

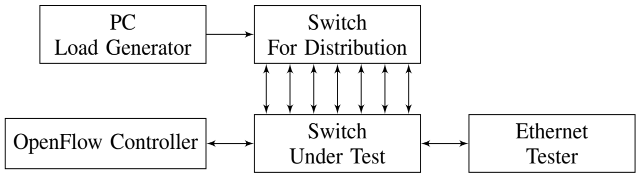

Fig. 10. Measurement setup with load generator used in Section VI

resembled thus the worst-case load the switch would see in a real aircraft. All virtual links had random starting offsets and multiple seeds were used during the measurements.

The generated load was fed into a 24 port switch that distributed the load across all its port according to the desired configuration. The distribution switch was also set-up with OpenFlow for easier configuration. All incoming traffic was then fed back into the distribution switch and forwarded to the load generator and logged to ensure the functionality of the tested switch.

At the same time an Ethernet Tester was directly connected to the *switch under test* and measured the latency through the switch by sending one test virtual link in and out of the switch.

The results in Fig. 11 indicate, that the COTS switch is very well capable of handling the reduced configuration with a similar latency characteristic. The AFDX switch has a minimally smaller processing delay of $5\mu s$ compared to $7\mu s$ in the COTS switch. The COTS switch on the other hand has a smaller average latency which might be due to the higher bandwidth in the switching backbone. The maximum values seen on both switches, however, are similar which therefore leaves us to conclude that the COTS switch is indeed able to handle the traffic.

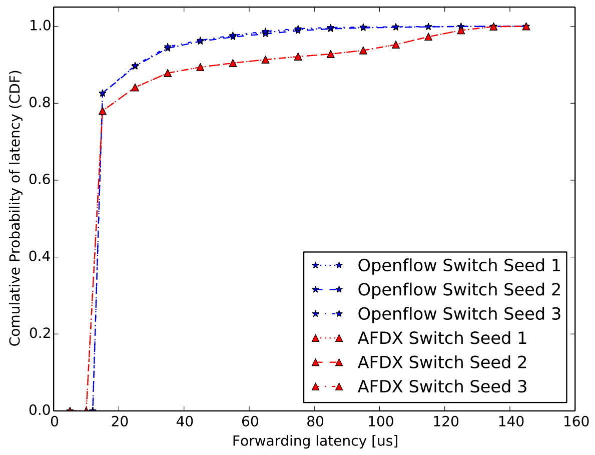

Fig. 11. Measurements of reduced realistic traffic configuration

VII. Conclusion

This paper evaluated the use of COTS switches in avionic environment. An approach was presented to achieve bounded latencies on standard OpenFlow switches. The approach was explained and a mathematical framework for calculation of upper bounds sketched. It was then implemented on a COTS switch supporting OpenFlow 1.3 and a measurement study was conducted.

Software based forwarding did not work as expected and results in a factor 30 to 6000 higher latency. However, when hardware based forwarding was used the switch offered good performance, with the limitation of only having 512 entries available. Newer generation switches will likely be able to handle more entries. A promising candidate, that was not available to us at the time of writing, is the Centec V350 switch which according to its data sheet would be able to handle and meter up to 2000 flows.

While this shortcoming did not allow the switch the handle the full aircraft traffic, it was able to handle most parts of it and offer similar performance as a state of the art switch that is currently in use in aircraft.

Acknowledgment

This work was supported in part by the German Federal Ministry of Economics and Technology (BMWi) under the *KASI* project (support code 20K1306B).

References

[1] Open Networking Foundation, "OpenFlow Switch Specification," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf

[2] Aeronautical Radio Incorporated, "Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX)," 2006.

[3] B. Sonkoly and A. Gulyás, "On QoS Support to Ofelia and OpenFlow," in *Workshop on Software Defined Networking (EWSDN)*, 2012.

[4] M. Seddiki, M. Shahbaz, S. Donovan, and S. Grover, "FlowQoS: Providing Per-Flow Quality of Service for Broadband Access Networks," in *Hot topics in software defined networking HotSDN '14*, 2014.

[5] H. Egilmez and S. Dane, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks," *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pp. 1–8, 2012.

[6] K. He, J. Khalid, S. Das, A. Gember-jacobson, and C. Prakash, "Latency in Software Defined Networks : Measurements and Mitigation Techniques," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2015.

[7] S. Azodolmolky, R. Nejabati, M. Pazouki, and P. Wieder, "An Analytical Model for Software Defined Networking : A Network Calculus-based Approach," in *Global Communications Conference (GLOBECOM)*, 2013.

[8] A. Jacobs, J. Wernicke, S. Oral, B. Gordon, and A. George, "Experimental characterization of QoS in commercial ethernet switches for statistically bounded latency in aircraft networks," *Proceedings - Conference on Local Computer Networks, LCN*, pp. 190–197, 2004.

[9] W. Yun-sheng, "The COTS based IMA prototype for hosted applications development," *Digital Avionics Systems Conference (DASC)*, 2012.

[10] J.-Y. Le Boudec and P. Thiran, "Network calculus: a theory of deterministic queuing systems for the internet," in *Vol. 2050. Springer Science & Business Media*, 2001.

[11] Ericsson Innovation Center in Brazil in a partnership with CPqD in technical collaboration with Ericsson Research., "CPqD OpenFlow 1.3 Software Switch," 2015. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[12] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Methods for bounding end-to-end delays on an AFDX network," *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pp. 193–202, 2006.