FTT-OpenFlow, on the way towards real-time SDN

Cédric Ternon
Computer Science Department
Université Libre de Bruxelles
cternon@ulb.ac.be

Joël Goossens
Computer Science Department
Université Libre de Bruxelles
jgoosens@ulb.ac.be

Jean-Michel Dricot
OPERA Department - Wireless Communications Group
Université Libre de Bruxelles
jdricot@ulb.ac.be

ABSTRACT
Software-defined networking proposes a new paradigm to operate computer networks. Where routers and switches execute predetermined distributed protocols, OpenFlow offers to replace them with devices where the logic that determines the flows of packets is freely programmable and centralized. Applied to the field of real-time networks, this freedom would allow design networks to overcome existing standards and to make experimentation easier. However, neither OpenFlow nor Ethernet were designed having real-time constraints in mind. FTT-Ethernet is a master-slave protocol allowing the meeting of real-time constraints using commodity Ethernet hardware. This paper's aim is to study: (i) how FTT (Flexible Time Triggered) principles could be applied to OpenFlow, allowing its usage in a hard real-time context; (ii) which benefits OpenFlow can bring to the FTT paradigm.

Keywords
FTT: Flexible Time Triggered; SDN: Software Defined Networking; real-time; OpenFlow

1. INTRODUCTION
Devices composing a computer network are designed to implement predefined network protocols (like IP, MPLS) and operate according to them. The behavior of a given network is only configurable to the extent provided by the used protocols. A new paradigm remove these protocol limitations by means of network programmability. This paradigm is SDN [6] standing for Software Defined Networking. A contemporary implementation of the concept is OpenFlow [6]. The standard defines network switches as machines manipulating packets with an instruction set and provides a language to program them. The logic determining flows of packets is no longer distributed among network devices but centralized in a software controller. This paradigm shift allows one to design computer networks in a more flexible way. Using it allows one to implement existing protocols and design new behaviors. According to the authors of the standard, this will make research and experimentation in the network field easier [11].

In the context of real-time task scheduling, designers are free to conceive any scheduling policy e.g. RM, DM, EDF, *etc.* [16] later implemented in the operating system's scheduler by means of software. In the context of real-time networks, such a freedom does not exist. A multitude of norms and technologies exist, forming a straightjacket that it is impossible to escape without creating a new protocol. Would the argued flexibility of software defined networks permit their use in a real-time network context? Does the expressivity of OpenFlow free one to conceive arbitrary packets scheduling policies and implement them through it? If so, it would generate a new and unprecedented freedom in the domain of real-time networks. A novel research field like mixed criticality systems [3] could be considered beyond the existing technologies. Moreover, as OpenFlow switches may be dynamically configured, multi-operation mode [14] networks may be defined. Such behavior is interesting e.g. in the field of avionics. Aircrafts go through different phases: parking, taxiing, take-off, cruising and landing. It might be desirable that the network adapts to those different phases.

FTT (Flexible Time Triggered) [2, 13, 8, 10] is a paradigm developed for hard real-time communications which has been ported to different network technologies. Its Ethernet version enables the meeting of real-time constraints using commodity Ethernet hardware. This paper studies first how FTT principles could be applied to OpenFlow, allowing its usage in a hard real-time context. Our contribution is presented afterward: FTT-OpenFlow, a novel protocol based on the FTT paradigm and taking advantage of OpenFlow possibilities for sporadic flows handling.

The remainder of this paper is organized as follows: Section 2 states the related work on FTT; Section 3 presents the proposed protocol; Section 4 concludes the paper and lists future works while Section 6 eventually produces the references.

2. RELATED WORK
FTT is a paradigm developed for hard real-time communications. It has been developed with the goal to handle time triggered traffic (periodic flows) as well as event triggered traffic (sporadic flows). It is a centralized architecture with a master in charge of the scheduling of transmissions on the network. The choice of the scheduling policy is free and this one must only be running in the master. The master also runs an online admission mechanism to accept new

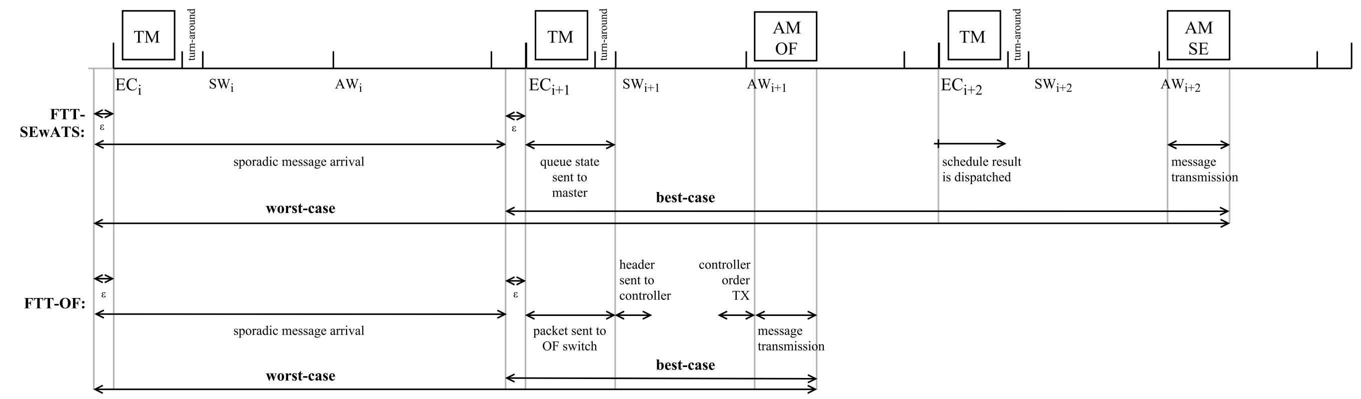

Figure 1: FTT-SEwATS and FTT-OF sporadic message transmission timeline

flows only if they are schedulable by the policy in use. The paradigm has been implemented on different network technologies like CAN [2] and Ethernet [13, 8, 10]. The different versions developed for Ethernet are hereafter detailed.

2.1 FTT-Ethernet

With its non deterministic handling of medium contention, Ethernet is not suited to handle real-time traffic [4]. However its widespread diffusion went along with high availability and low prices. This led players of the real-time network industry to use it partially, AFDX and TTEthernet being such examples. More specifically, FTT-Ethernet [13] goes further as it allows hard real-time communications using non modified Ethernet hardware. This is realized by delegating the scheduling of any transmission on the network to a master node. The master may use any scheduling policy. As the master schedules transmissions in non overlapping timeframes, any non deterministic contention is completely avoided. In practice, the time is divided in EC (Elementary Cycles) of a fixed duration, all starting with a TM (Trigger Message) sent by the master. Every EC includes two windows: (i) one dedicated to periodic traffic (synchronous window) and (ii) a second one dedicated to sporadic and non real-time traffic (asynchronous window). Every TM includes transmission times of periodic messages scheduled in the consecutive synchronous window. For sporadic and non real-time traffic, the master polls the stations during the asynchronous window.

2.2 FTT-SE: Switched Ethernet

As long as only one station is connected to each port, the use of a switch rather than a hub allows an enhancement of the FTT-Ethernet protocol. This enhancement is called FTT-SE for Switched Ethernet [8]. In such a micro-segmented setup, collisions are no longer possible. However the overflow of an output queue will lead to message loss. Using the FTT paradigm completely avoids such overflows. If inversions of messages inside a given synchronous window are acceptable, micro-segmentation offers a chance to reduce the time constraints on the stations. In FTT-Ethernet, transmissions of periodic messages have to occur within precise temporal bounds. In FTT-SE, all stations with a message scheduled by the TM are allowed to transmit it at the start of the synchronous window. The switch will then transparently serialize them with its queuing mechanism.

2.3 Asynchronous Traffic Signalling

The use of micro-segmentation in combination with full duplex links leads to an improvement of the FTT-SE protocol hereafter referred to as FTT-SEwATS [10]. The improvement lies in the fact that the download link of the master is left free during its transmission of the TM and the following turn-around window. This window is the amount of time required by the stations to be able to react to the TM. The master download link is used during this free period as a channel allowing sporadic traffic senders to request bandwidth. They do this by sending the master their sporadic traffic queuing status. This avoids periodically polling the senders and reserving bandwidth based on their minimum inter-transmission times. The timeline of transmission of a unique sporadic message not competing with other ones is given on Figure 1. Though not modifying the worst case, this scheme allows better average latency of sporadic messages and better use of available bandwidth. The price is a limitation on the amount of sporadic flows. Indeed, it is necessary that the serialization of sporadic senders queuing status by the switch doesn't incur after the free period. The maximum amount of sporadic flows is computable and depends on:

- trigger message transmission time;
- turn-around time;
- queuing status transmission time.

If the computed amount falls under requirements, two approaches are proposed. First one is to reduce the rate at which sporadic senders transmit their queuing status below once per EC. This supports an arbitrary amount of sporadic flows with a negative impact on latency. The other one is to extend the signaling period to the synchronous window. This implies adapting the periodic message scheduler to count for this extra traffic on the master download link. The advantage is an absence of impact on the slaves application given that no master directed application level messages are required. A noticeable drawback lies in the fact that without such extension, some periodic message sets would result in empty synchronous windows in some EC. With such extension, a minimum length of synchronous window is needed for queuing status transmission. This has a negative impact on bandwidth usage and sporadic messages schedulability.

2.4 HaRTES

The protocols mentioned in Sections 2.1–2.3 use COTS (Commercial off-the-shelf) components. While this was a key aspect of their development, this absence of control on the switch also has certain drawbacks. For example, (i) the system is based on strict compliance of the nodes to the protocol. Hence, a unique malfunctioning or non FTT node is able to jeopardize the whole system. Also, (ii) by design, all the traffic of a given EC has to be be cleared off inside it. Consequently, a single node having its receiving path full in a given EC may force to delay some broadcast or multicast traffic. (iii) The signaling scheme used in Section 2.3 adds latency and presents some scalability issues. To overcome those drawbacks, HaRTES: Hard Real-Time Ethernet Switching, an implementation of FTT in a switch using FPGA technology is proposed in [15].

3. FTT-OPENFLOW

3.1 FTT through OpenFlow layer

As Ethernet, OpenFlow was not built with real-time constraints in mind. There are no mentions to time except in the "Time scheduled bundles Extension" [12]. However, this extension only applies modifications on a set of OpenFlow switches in a synchronized manner. Moreover, like on traditional switches, priorities are managed with the help of output queues. The amount of queues available is usually not sufficient to implement a fixed priorities policy (e.g. RM) in a realistic manner, for example. The research conducted on the FTT paradigm will therefore be an excellent starting point to enable OpenFlow usage in a real-time context. OpenFlow and FTT-Ethernet are two centralized systems with the OpenFlow controller on one side and the FTT master on the other. Intuitively, these two components may be combined.

It is possible to implement traditional network components in an OpenFlow switch. More specifically, an OpenFlow controller may inject rules in an OpenFlow switch leading it to act as a traditional layer 2 switch [7]. Therefore, as the FTT-SEwATS paradigm (detailed in Section 2.3) only requires a layer 2 switch to function, it can be directly used in such a setup. No modifications are needed and all the properties of the protocol will remain unchanged.

The subject of the next section is to propose an improvement of the FTT-SEwATS protocol taking advantage of OpenFlow possibilities.

3.2 Sporadic traffic

The different versions of FTT-Ethernet handle periodic traffic in an elegant way. The centralized architecture certainly produces an overhead. This one is however diminished by a unique TM scheduling all the periodic traffic of a given EC. FTT-SE reduced the time constraints on the periodic traffic producers, allowing them to directly emit their message once scheduled by a TM. Sporadic traffic handling is the field where improvements can be brought. In FTT-Ethernet and FTT-SE, the master periodically polls the sporadic producers. The period is the minimum inter-transmission time: T_{mit}. The deadline of a sporadic flow can therefore not be smaller than its T_{mit}. The activation time: T_{act} is the delay between sporadic message deployment in a slave queue and its inclusion in the master scheduler. In the worst case, $T_{\text{act}} = T_{\text{mit}} - \epsilon$, this happens when the message arrives just after the last poll. Moreover, bandwidth is statically reserved and will be lost if no message is present when the station is polled. FTT-SEwATS does better: bandwidth is now dynamically allocated. At the beginning of each EC, during TM transmission and the following turn-around period, the stations transmit their sporadic queuing status to the master. In the worst-case, T_{act} is reduced to 2 EC and does not depend anymore on T_{mit}. The queuing status message preparation by the stations can not be considered atomic. Hence, at the end of each EC, there is a period ϵ, in which the time left is too scarce to permit inclusion of a new sporadic message in the following queuing status transmission. The worst case delay between message arrival and end of transmission is therefore given by:

$$\epsilon + \text{EC}_i + \text{EC}_{i+1} + \text{TM}_{i+2} + \text{TA} + \text{SW}_{i+2} + \text{AM}$$

as detailed on Figure 1. The message arrives during the end of EC_{i-1}, at the beginning of the ϵ period, just after the last instant allowing it to be transmitted in the consecutive EC. EC_i is thus lost before the queuing status can be sent to the master in EC_{i+1}. The message is eventually scheduled in EC_{i+2} after $\text{TM}_{i+2} + \text{TA} + \text{SW}_{i+2} + \text{AM}$. The improvement brought by FTT-OpenFlow concerns the sporadic traffic handling and is based on two key elements hereafter detailed.

3.2.1 Arming mechanism

In FTT-SEwATS, when a sporadic sender has a message to transmit, he sends the master his queue state. This is the same as a transmission request. One will therefore observe a series of events like: request - transmission - request - transmission... Another way to proceed is to use an arming mechanism and allow the sender to transmit only when he is armed. After transmission, the sender must thus be rearmed before being allowed to transmit again. If the sender is initially armed, one will observe a series of events like: transmission - rearming - transmission - rearming... The second series also allows to constrain transmission, with the advantage of being able to offer a better latency. Furthermore, in bus based networks like shared Ethernet or CAN, the stations know when their messages have been transmitted with success. This knowledge is lost in switched Ethernet as the packet may be queued or dropped. Transmitting a rearming message during transmission of a given message returns this information to the sender, like on a bus based system. This mechanism will be used for sporadic flows in the overcoming proposition.

3.2.2 Packet-in, packet-out and buffering

In the OpenFlow specification [1], there is a possibility to set rules ordering the switch to send some packets to the controller by means of a packet-in message. It may be specified if the entire packet is to be transmitted or only an arbitrary number of starting bytes, usually covering the header. In case of partial transmission, the switch places the packet in a buffer and transmits the relevant buffer identifier in the packet-in message. This is usually the case when no rules specify the switch how to treat the packet, allowing the controller to add one. In the same way, the controller may

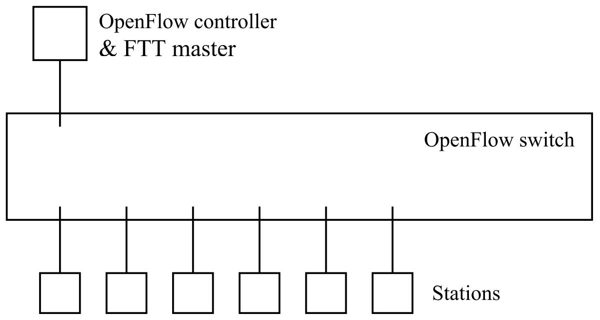

Figure 2: FTT-OpenFlow architecture

ask the switch to send a packet with a packet-out message. The controller may send the entire message to the switch or refer to a buffer identifier from a previous packet-in message. In the overcoming proposition, sporadic messages will be buffered and their headers transmitted to the master. The master will be in charge of their scheduling and order their transmission by means of a packet-out message.

3.3 Protocol description

A diagram of the architecture is given on Figure 2. For the sake of simplicity, in this first proposition of FTT-OpenFlow, multi-cast possibilities are neglected and a bus behavior with real-time traffic broadcasted to every station is assumed. With this approach, a sporadic message is sent back to the initial sender upon transmission on the network, playing the role of the rearming message. The transmission of periodic messages follows exactly the same rules as in FTT-SE. For sporadic transmissions, an armed sender is allowed to directly transmit his entire message during the time slice composed by $\mathrm{TM} + \mathrm{TA}$ where TA is the turn-around time. Upon reception, the switch identifies it as being part of a sporadic flow. It's set of rules instruct it to buffer the message and transmit the header and relevant buffer identifier to the controller. At that moment, the controller who also plays the role of the FTT master is in charge of scheduling this particular message. Following the arbitrary scheduling policy in use, when the effective transmission has to occur in the asynchronous window, the controller sends the transmission order to the switch. All the sporadic traffic is thus buffered by the switch and sent over the network when the master orders to do so. If the OpenFlow processing delay of a TM message differs from the one of a sporadic message transmission order, the timeliness of the forecasted schedule may not be respected. In case of a difference between those two delays, the induced shift shall be compensated. This can be done by sending the order in advance to enable its transmission and consecutive switch processing to end just before the instant of scheduled message transmission.

3.3.1 *Transmission order latency*

The latency between start of transmission of a `packet_out` message by the controller and beginning of the related sporadic message transmission by the switch is made of these components:

- $\Delta t_{\mathrm{trans}}$ transmission delay: amount of time spent transmitting data on the link;
- Δt_{prop} propagation delay: delay of the signal in the physical medium;
- $\Delta t_{\mathrm{queue}}$ queuing delay: amount of time the packet will wait in switch's queue;
- Δt_{proc} processing delay: amount of time required for the switch to process the packet.

The transmission delay depends on packet size s and data rate r: $\Delta t_{\mathrm{trans}} = \frac{s}{r}$.

The propagation delay depends on the medium used and it's length.

For the queuing delay, one of the key elements of FTT-SE and FTT-OF to manage hard real-time constraints is to avoid queuing or restrict it to precise boundaries in the case of periodic message serialization. The messages going trough the controller/master upload link are:

- TM: trigger messages;
- `packet_out` messages: orders to transmit buffered sporadic messages;
- answers of online admission control mechanism;
- OpenFlow controller-to-switch messages like configuration modifications.

The TM and `packet_out` messages shall not suffer any queuing in any case as it would jeopardize the precise schedule enabling to meet hard real-time constraints. Online admission control and OpenFlow configuration messages shall therefore be sent in periods where they would not compete with previous ones. For OpenFlow configuration messages, one such period starts just after the TM message transmission with a sufficient delay before the asynchronous window. For online admission control, idle periods of the network can be used. With such cautions, the queuing delay can be assessed as zero.

The processing delay is considered in Section 3.4. This delay will depend on the particular switch being used. If it presents variations, a guarding window considering the worst case shall be used at the end of the asynchronous window to compensate. This will ensure that the consecutive TM message transmission does not suffer any delay at the price of a negative impact on bandwidth usage.

3.3.2 *Transmission timeline*

The timeline of transmission of a unique sporadic message not competing with other ones is given on Figure 1 for FTT-SEwATS and FTT-OF. In such case, the FTT-OF worst case delay between message arrival and end of transmission is given by:

$$\epsilon + \mathrm{EC}_i + \mathrm{TM}_{i+1} + \mathrm{TA} + \mathrm{SW}_{i+1} + \mathrm{AM}$$

an improvement in latency of one EC compared to FTT-SEwATS [10].

3.3.3 *Schedulability impact*

In the proposed design, every sporadic message is first buffered by the switch. Effective transmission occurs afterwards following an order to do so emitted by the controller. When several sporadic messages have to be transmitted consecutively, the controller has to emit the transmission orders serially. If the minimum length sporadic message has a smaller size than the transmission order, a series of minimal length sporadic messages can not be transmitted at full speed if the controller uses the same connection speed

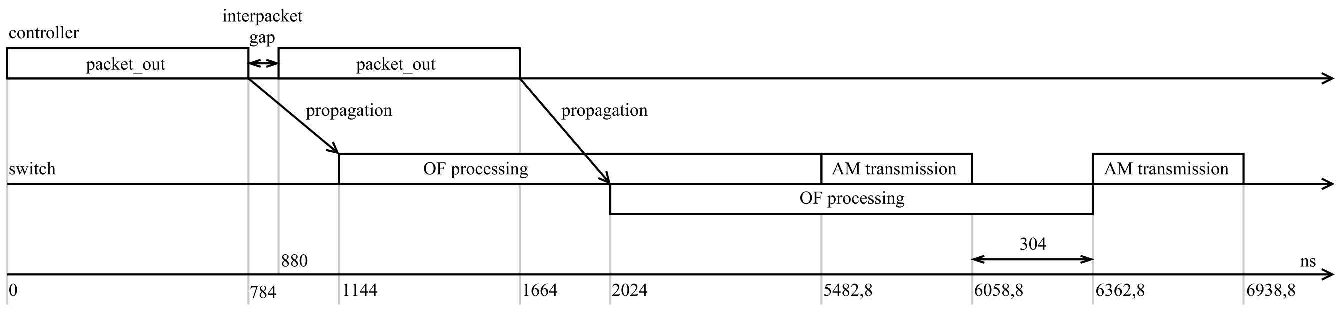

Figure 3: Two consecutive sporadic messages transmission

than the stations. The sporadic messages will be spaced by a gap equalling the difference between transmission order and minimum sporadic message length. Provided that the switch is capable of processing multiple messages simultaneously, the impact can be incorporated into schedulability analysis considering sporadic messages as having a minimum size equalling the transmission order size.

It should be noted that the amount of sporadic senders is limited to the same extent as in FTT-SEwATS. Research on this field will be conducted in future work.

3.3.4 Predictability

Periodic message handling follows exactly the one developed in FTT-SE. Therefore, the same outcome applies: message inversions are possible within one EC. This is the price paid for reduced time constraints on periodic senders. With a time scale resolution of one EC, the timeliness of the schedule is however respected. Sporadic traffic does not suffer this outcome. Their transmissions are ordered by the master and the queuing mechanism of the switch is not used. The elements that may jeopardize the timeliness of the schedule are avoided by construction. Those are:

- Answers of online admission control mechanism by the master;
- OpenFlow master/controller-to-switch messages like configuration modifications;
- OpenFlow processing time variability.

The two first ones are emitted by the master and periods avoiding contention are used. For the last one, a guarding window at the end of the asynchronous window is used.

3.4 Example analysis

The variables cited in Section 3.3.1 are here after fixed on an example. This is an opportunity to have some highlights on the order of magnitude of the considered delays and on some OpenFlow details. The example is constituted by an OpenFlow switch with gigabit ports and a 72 meters long (Airbus A380 length) connexion between the controller and switch. This connexion is made of a twisted pair cable.

The transmission delay of an order of sporadic message transmission depends on it's size. This one can be inferred from OpenFlow specification as follows. Messages are exchanged between controller and switch through an OpenFlow channel. This channel is usually instantiated using TLS or plain TCP. Focusing on time constraints, encryption will be avoided and plain TCP/IP over Ethernet chosen. The `packet_out` message will therefore be encapsulated in a TCP/IP frame itself encapsulated in an Ethernet frame. The layer 1 Ethernet packet size from preamble to frame check sequence is 26 bytes excluding payload. As the minimum payload of 46 bytes will be satisfied, no padding is necessary. Minimum IP and TCP headers without any options are 20 bytes each, leading to a size of 66 bytes + `packet_out` size. The variable size elements of this `packet_out` are:

- `ofp_match`: packet pipeline fields;
- `ofp_action_header`: action list to apply;
- packet data.

As the packet is referred to with a `buffer_id` rather than being transmitted, packet data will be void. The action list is composed of a unique `OFPAT_GROUP` action: process packet through a given group. Group processing allows to send the packet back to the initial sender for rearming, a behavior that is not possible another way. This leads to an action list size of 8 bytes. Packet pipeline fields are fields like ingress port, required for OpenFlow processing but that can not be inferred from the packet headers. As they can't be inferred, those fields must be transmitted along in the case of a `packet_in` or `packet_out` event. In this case, the `OXM_OF_IN_PORT` (ingress port) is mandatory. With only one field, the packet pipeline fields need 8 bytes. The non varying elements of `packet_out` count for 16 bytes, giving a `packet_out` size of 40 bytes and a total frame size of 98 bytes. With a gigabit connexion, $\Delta t_{\text{trans}} = 784$ ns.

The propagation delay depends on the medium used and the connexion's length. Using twisted pair copper wire and a propagation speed of two thirds the speed of light ($\frac{2}{3} \times 3 \times 10^8\ \mathrm{m\,s^{-1}}$), with a 72 meters cable, $\Delta t_{\text{prop}} = 360\,\text{ns}$.

The OpenFlow processing delay is analyzed in [5] for a NEC PF5240 switch. The studied case is not exactly the same as this is the particular case of `packet_out` processing delay which is here considered. However, the study brings interesting highlights on the matter. The study infers processing delay from a packet's round-trip time. For this particular switch, it concludes that the processing delay is independent of frame-size and amount of flow table entries configured in the switch. For a 1526 bytes long packet, the mean processing delay is 4338.8 ns with a standard deviation of 160.43 ns of the round-trip time.

Using the processing delay from the above referred study and the parameters of the considered example, the latency between start of `packet_out` and start of transmission of referred sporadic message is estimated being 5482.8 ns.

Concerning the minimum sporadic messages length to consider in schedulability analysis. The messages being buffered, the size of the orders is independent of the sporadic message's size. `packet_out` layer 1 frames with adequate group action does have a size of 98 bytes. Ethernet implies a 12 bytes interpacket gap and the minimum Ethernet layer 1 frame size is 72 bytes long. A minimum layer 1 size of 98 bytes rather than 72 bytes shall therefore be considered in schedulability analysis. Figure 3 gives the transmission timeline of 2 consecutive smallest size sporadic messages with the parameters of the example and a 4338.8 ns OpenFlow processing time.

4. CONCLUSIONS AND FUTURE WORKS

FTT paradigm allows one to directly use non modified OpenFlow in a hard real-time context. In the proposed protocol, worst case and average latencies of sporadic traffic are improved compared to FTT-SEwATS [10]. Non real-time traffic also benefits the improvement since it can be handled as sporadic one with a minimum priority. Some drawbacks addressed by HaRTES like non complying or non FTT nodes can easily be handled with OpenFlow. Prototype experimentation is the next step. OpenFlow possibilities lead the way to new versions e.g. with relaxed timeframe for sporadic emissions by the nodes or novel scheme for non real-time traffic. Schedulability analysis developed for FTT-SE [9] remains valid for periodic traffic as it is handled the same way while sporadic traffic schedulability analysis is to be conducted in further work. The proposal of new OpenFlow extensions allowing sporadic traffic handling with reduced controller intervention also constitutes a field for further research.

5. ACKNOWLEDGEMENTS

Thanks go to Luís Almeida for his availability and the discussions we had, and to Gary Farrelly for reviewing the spelling of this paper.

6. REFERENCES

[1] Openflow switch specification, version 1.5.1. *Open Networking Foundation*, 2015.

[2] L. Almeida, P. Pedreiras, and J. A. G. Fonseca. The ftt-can protocol: Why and how. *Industrial Electronics, IEEE Transactions on*, 49(6):1189–1201, 2002.

[3] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 147–155. IEEE, 2008.

[4] J.-D. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6):1102–1117, 2005.

[5] F. Dürr and T. Kohler. Comparing the forwarding latency of open-flow hardware and software switches. Technical report, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Institute of Parallel and Distributed Systems, Germany, 2014.

[6] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ACM SIGPLAN Notices*, volume 46, pages 279–291. ACM, 2011.

[8] R. Marau, L. Almeida, and P. Pedreiras. Enhancing real-time communication over cots ethernet switches. In *WFCS'06: IEEE International Workshop on Factory Communication Systems*, pages 295–302, 2006.

[9] R. Marau, L. Almeida, P. Pedreiras, K. Lakshmanan, and R. Rajkumar. Utilization-based schedulability analysis for switched ethernet aiming dynamic qos management. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–10. IEEE, 2010.

[10] R. Marau, P. Pedreiras, and L. Almeida. Asynchronous traffic signaling over master-slave switched ethernet protocols. In *6th International Workshop on Real Time Networks (RTN'07)*, 2007.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[12] T. Mizrahi and Y. Moses. Software defined networks: It's about time. In *IEEE INFOCOM*, 2016.

[13] P. Pedreiras, L. Almeida, and P. Gai. The ftt-ethernet protocol: Merging flexibility, timeliness and efficiency. In *Proc. 14th Euromicro Conf. Real-Time Systems*, pages 134–142. IEEE Press, 2002.

[14] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-time systems*, 26(2):161–197, 2004.

[15] R. Santos, R. Marau, A. Vieira, P. Pedreiras, A. Oliveira, and L. Almeida. A synthesizable ethernet switch with enhanced real-time features. In *Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE*, pages 2817–2824. IEEE, 2009.

[16] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.