A protocol for mixed-criticality management in switched Ethernet networks

Olivier CROS, Laurent GEORGE
Université Paris-Est, LIGM / ESIEE, France
cros@ece.fr,lgeorge@ieee.org

Xiaoting LI
ECE Paris / LACSC, France
xiaoting.li@ece.fr

***Abstract*—In real-time industrial networks, providing timing guarantees for applications of different criticalities often results in building separate physical infrastructures for each type of network at the price of cost, weight and energy consumption. Mixed-Criticality (MC) is a solution first introduced in embedded systems to execute applications of different criticality on the same platform. In order to apply MC scheduling to off-the-shelves Switched Ethernet networks, the key issue is to manage the criticality change information at the network level. The objective of this work is to propose a criticality change protocol for MC applications communicating over Switched Ethernet networks. The protocol relies on a global clock synchronization, as provided by the IEEE-1588v2 protocol, and a real-time multicast based upon it, to preserve the consistency of the criticality level information stored in all the Ethernet switches. We characterize the worst-case delay of a criticality change in the network. Simulation results show how the criticality change delay evolves as a function of the network load.**

I. Introduction

Nowadays, highly-constrained industrial systems found in defense, public transports or home automation have increasing needs in terms of reliability and performance. It is a common situation for such systems to integrate several independent network architectures in order to transmit, in each network, messages of different criticality (e.g. in bus: passenger information, mechanical control information, speed, etc.) such that each system can be certified in isolation. This solution is very expensive in terms of cost, weight and hence in terms of energy consumption, as each network must have its own infrastructures, materials and wires. For example, the mechanical functions, trajectory control and the passenger information are often treated in separated infrastructures inside a public bus, with different dedicated materials.

One solution to this problem is the mixed-criticality (MC) scheduling approach first proposed in the context of uniprocessor and multiprocessor systems [1]. It executes several applications of different criticalities on the same platform by adapting certification effort to the level of assurance needed at a given criticality level. Since a networked system is an interconnect system for applications of different criticalities, the objective of this work is to study how to manage criticality level information in networked systems.

With MC scheduling, each task is characterized by the maximum criticality level it is allowed to execute. A task can be non-critical, critical for the mission (mission-critical), critical for the safety of the vehicle (mechanical-critical), or for the safety of its occupants (safety-critical). Each criticality level must provide guarantees on end-to-end transmission delays, especially for high critical tasks. The more critical a task is, the more reliable the guarantee should be.

In the real-time network context, we focus on how to integrate criticality management in networked systems. The first point is to bound the number of criticality levels we use. Baruah [2] showed that the complexity of MC scheduling problems is NP-hard in the strong sense. In order to limit the complexity of our architectures, we focus on a two-level criticality network, as presented in [3]. These two criticality levels are called low-critical (LO) and high-critical (HI) levels: only a set of predefined messages can be transmitted in HI criticality level, whereas all messages can be transmitted in LO criticality level.

The main goal of the criticality management in networked systems consists in providing Quality of Service (QoS) guarantees (in terms of worst case end-to-end transmission delays), specially for high critical messages. In this context, we focus on a method to grant the consistency of the criticality level information in a Switched Ethernet network, to ensure bounds on end-to-end transmission delays of messages as a function of the criticality of information sent.

In uniprocessor and multiprocessor systems, the problem of characterizing the impact of low-criticality tasks when a criticality change from LO to HI has been considered by bounding the demand of carry-in jobs.

Assuring deterministic communications in networked systems implies to be able to bound the end-to-end delay of all messages. In [4], we proposed a tool to evaluate the worst case end-to-end delay of any message sent on a Switched Ethernet network relying on a global clock-synchronization protocol. In this paper, we also consider a clock synchronized network, synchronized with the IEEE-1588v2 synchronization protocol and its implementation in Precision Time Protocol (PTP). On top of the clock synchronization, we build a reliable multicast to consistently switch the criticality level on all the nodes of the network.

In this paper, first we present in II the network model studied in this work. In III, we illustrate a link utilization problem based on an example, and then we present the importance of managing MC in network context. We propose a MC management protocol in IV. Finally, we show by simulation the performances of this protocol in V, and conclude this paper

as well as future perspectives of this paper in VI.

II. Network Model

A. Mixed-Criticality

In this paper, we consider a tree-based topology as those found in application domains like avionics systems and recent and futur public transport systems [5]. The network is composed of a set of interconnected nodes, all organized according to a tree-based structure with one final collecting node denoted the sink node. An example of such topology is the one showed in figure 1, with S_4 as the sink node.

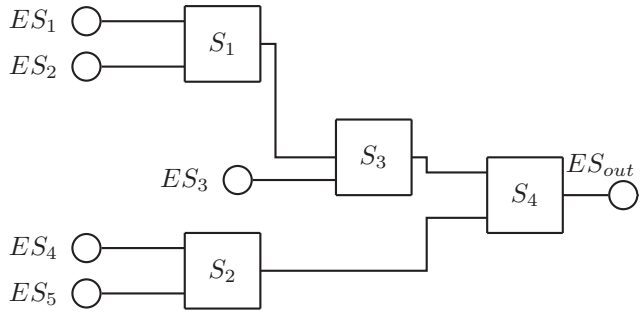

Fig. 1. Centralized Network architecture

Our goal is to propose a criticality level management protocol in a tree-based Switched Ethernet network topology. The sink node is in charge of stroring the network criticality information. All the nodes of the network have a local copy of the network criticality level information. The protocol we propose should maintain the consistency of the criticality level information in all the nodes of the network in the case of a criticality switch.

We then have two cases:

- If the network criticality level is LO, the first node sending a request to the sink-node, to change the criticality level to HI will result in a multicast sent by the sink node the all the nodes of the network with the new network criticality level. All the nodes receiving this message should update their local criticality level information so as to keep it consistent after a criticality mode switch.
- If the network criticality is HI, a switch to LO criticality level can happen only if the sink node has received from all the other nodes a request to switch to LO mode.

A simple multicast is not sufficient to guarantee the consistency of the network criticality level information in all the nodes. We introduce a real-time reliable multicast protocol in section IV-C as one solution to this consistency problem. We caracterize the maximum time needed to switch the network criticality level from LO to HI, from the request of the first node willing to change the criticality level to HI, to the time all nodes are allowed to change their criticality level (after receiving the reliable multicast from the sink node).

B. Notations and main hypothesis

In MC systems, representing different levels of criticality inside a system is mostly based on a choice between two different hypotheses : either a message has a dedicated worst-case transmission time (WCTT) for each criticality level, which means that the flow of data sent in the entry points are longer in the case of HI modes. For example, as a plane is landing, it might need more precise evaluation of the altitude. It means that the altitude sensors will send more complete, and so longer, data values.

The second hypothese is not based on longer WCTT, but on a more frequent messages. Each message has now two different periods, one for LO level and one for HI level. It corresponds to increasing the number of measures during a critical phase : for example, during landing, the measure of speed or altitude could have to be more frequent. We consider this case in our analysis.

A network is a set of interconnected switches communicating through full-duplex links connecting end-systems. On each link, we can send one or several flows v_i, and each flow produces several messages. Each flow v_i is represented as a 3-tuple $v_i = \{\mathcal{P}_i, C_i, \vec{T_i}\}$ where:

- $\mathcal{P}_i$ is the path of nodes followed by any message of v_i, starting from a source node to a destination node. We consider this path as statically defined by the designer.
- C_i defines the WCTT of any message of v_i sent in LO or HI network criticality level.
- Its period is defined as $\vec{T_i} = \{T_i^{LO}, T_i^{HI}\}$. It is a vector of different periods of the flow, corresponding to LO-critical and HI-critical period (we assume two network criticality levels in this paper).
- In the case where a flow can only be sent in LO criticality level, that means that the flow will not be sent by a switch when the network criticality level is HI.

Furthermore, we suppose that:

- Each message is independent from each other
- All message transmissions are non-preemptive
- All the switchs use a Fixed Priority (FP) scheduling with FIFO scheduling in a specific FP queue, denoted as FP/FIFO scheduling.

III. Problem statement

A. An example

We consider the case of a simple network composed of one switch S (denoted S_M is it support MC), scheduling flows with FIFO scheduling and having three entry ports $ES1$, $ES2$ and $ES3$ respectively receiving flows v_1, v_2 and v_3, with the following parameters:

Flow	T_i^{LO} (μs)	T_i^{HI} (μs)	C_i (μs)	u_i^{LO}	u_i^{HI}
v_1	500	250	100	0.2	0.4
v_2	500	250	100	0.2	0.4
v_3	300	-	100	0.33	-

Imagine a scenario where all the three flows are transmitted in the LO criticality level. The LO-utilization (u^{LO}) of the network at the most loaded node S_4 is then $u^{LO} = u_1^{LO} + u_2^{LO} + u_3^{LO} = 0.73$. Then flow v_1 and flow v_2 increase their workloads by reducing the periods of messages due to certain emergencies. Then flow v_1 and flow v_2 are transmitted

in HI criticality level. Supposing that there is no criticality management, now the utilization at the node S_4 should be $u_1^{HI} + u_2^{HI} + u_3^{LO} = 1.13$. It means that S_4 is overloaded in a mixed mode with both criticality levels.

We focus on the impact of such an overloaded link on transmission delays when criticality levem is not managed by the nodes. We suppose that, at $t = 100$ μs, the system becomes high-critical : v_1 and v_2 start emitting messages according to T_i^{HI} and no more to T_i^{LO}. Basically, this results in a strong increase in the transmission delay for the frames of v_1 (see S in figure 2).

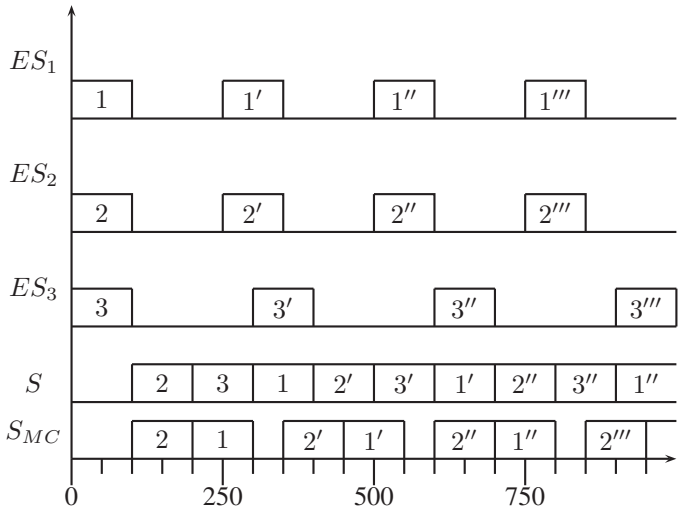

Fig. 2. Transmission delay with criticality management

We can observe that the transmission delay of messages from v_1 increases drastically with time. In fact, we can easily compute that the waiting delay of each message from v_1 in the entry point of S_4 increases of 50 ms at each new emission. Thus, in classical Ethernet context, switches do not have input buffers of infinite size, so a too high waiting delay can result in dropping out a message, and then loss of data.

If the network supports MC management (see S_{MC} in figure 2), we can observe that the transmission delay of HI-critical messages is constant and that criticality management allowed us to fix the overload problem. We show next a protocol to implement MC in network context.

IV. A CRITICALITY-CHANGE PROTOCOL

A. A two-phase protocol

Transmitting and managing criticality level inside network topology implies two different conditions. First, we need to assure that all nodes in the network have the same criticality level. Secondly, in case of criticality level switch, we have to be sure that all nodes change their local criticality level information preserving the consistency the network criticality level information.

Like we showed in II, criticality level is managed by a central entity (the sink node). It means that, even if each node has its own criticality level, it must be synchronized at all times with the one of the sink node. To assure this condition, we propose our MC managing protocol. It consists in assuring the consistent change of criticality level in all the nodes of a network topology. For this, we use a reliable multicast to ensure a total order (updates are scheduled in the same order in all nodes) for the update of criticality level information in all nodes. This preserves the serialisability of the criticality updates hence the consistency of the criticality level information [6].

This MC managing protocol works in two phases for a LO to HI criticality change. The node n in charge of initiating this criticality level change sends a criticality change request to the sink node. We name it the switch-criticality call (SCC) message, the delay needed to send this message from node n to the sink is denoted I^n_{delay}. Next, we need to send a criticality switch message with the new criticality level to all the nodes (except the sink node) of the network. This is the reliable multicast phase initiated by the sink node that send a timestamp of his local clock in the multicast message (recall that we assume a global clock synchronization). The delay needed to send and receive this message from the sink node to node n is denoted M^n_{delay}.

The total criticality switch delay S_{delay} in the network $\mathcal{N}$ can be computed by :

$$S_{delay} = \max_{n \in \mathcal{N}} (I^n_{delay} + M^n_{delay}) \quad (1)$$

Upon reception of a criticality switch message, each node will have to localy determine when to to the criticality switch. We now how yo characterize the criticality switch delay in the case of a LO to HI criticality switch.

B. Switch-criticality call

Suppose that the network criticality is LO. Calling for a criticality level change to HI consists in sending a specific message from a node n in the network requiring this criticality level change, and transmitting it to the central node responsible of criticality management. This SCC message is transmitted with the highest priority (except PTP messages), dedicated for configuration messages. Nevertheless, it can be delayed by other messages in the network: either by PTP synchronization messages, or by other messages due to the non-preemptive effect (a message even with the lowest priority cannot be stopped once its transmission has started).

The SCC message is considered as a new message in the network. It means that it is defined by its own path $\mathcal{P}_c$ from n to the central node S_h and its WCTT C_c. We consider that only one SCC message is sent by a switch (the first one received). If another nodes initiate another SCC message with the same criticality level, the switch receiving it will discard the message.

In other words, computing the delay needed to transmit the SCC consists in evaluating the delay needed for the message to be tranmitted from one node n to the sink node. To do this, we use the trajectory approach, presented in [7]. The SCC

transmission delay (noted as I^n_{delay}) can be computed by the general trajectory approach expression. Given that the call is done by a node n in the network, we apply the trajectory approach computation method to the switch-criticality call message c. This gives us the following result :

$$I^n_{delay} = \sum_{\substack{j \in hp_c \\ \mathcal{P}_c \cap \mathcal{P}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{W^{last_{c,j}}_{c,t} - M^{first_{c,j}}_i + A_{c,j}}{T^{LO}_j} \right\rfloor\right)^+ \cdot C_j \quad (2)$$

$$+ \sum_{\substack{j \in sp_c \\ \mathcal{P}_c \cap \mathcal{P}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{t + S^{first_{c,j}}_{max_c} - M^{first_{c,j}}_i + A_{c,j}}{T^{LO}_j} \right\rfloor\right)^+ \cdot C_j \quad (3)$$

$$+ \sum_{h \in \mathcal{P}_c \setminus \{n\}} \left(\max_{\substack{j \in sp_c \cup hp_c \\ h \in \mathcal{P}_j}} (C_j) \right) \quad (4)$$

$$+ (|\mathcal{P}_c| - 1) \cdot sl \quad (5)$$

$$+ \sum_{h \in \mathcal{P}_c} \delta^h_c \quad (6)$$

$$- C_c \quad (7)$$

- (2) is the delay induced by messages with higher priorities than the one of message c. These messages can delay c if they arrive at one shared output port with c during the maximized interval. For a higher-priority flow j , its maximized arrival jitter is $A_{c,j} = S^{first_{c,j}}_{max_c} - S^{first_{c,j}}_{min_c}$, where $S^h_{max_c}$ (resp. $S^h_{min_c}$) is the maximum (resp. minimum) delay of the message c from its source node $first_c$ till the output port h.
- (3) is the delay induced by messages with the same priority as message c. They are scheduled by FIFO policy. According to FIFO, messages arriving at the output port where $first_{c,j}$ (the first output port where they meet message c) during the maximized temporal interval $[M_{first_{c,j}}, t + S^{first_{c,j}}_{max_c}]$ can delay message c.
- (4) indicates the transmission delay of a message sequence including message c. This delay is maximized by considering the transmission time of the largest frame in the sequence at each output port along $\mathcal{P}_c$.
- (5) is the electronical latency induced by the transmission through wires (with sl, the electronical latency induced by the transmission between two nodes)
- (6) represents the delay induced by the non-preemptive effect (see [8]) of flows with a lower priority
- (7) finally, we substract C_c because the global transmission delay $W^{last_c}_{c,t}$ corresponds to the delay between the emission of c and its starting instant of emission in the final sink node

As SCC message has a high priority (dedicated for configuration messages), the only higher priority messages that will delay us are the PTP synchronization messages. We consider that PTP messages and switch-criticality call messages are the only one to be sent in this level of priority and higher, so there is no other message with the same priority as SCC. It means that we just need to compute the number of PTP messages generated in each node encountered along the path of the switch-criticality call.

Moreover, we suppose that PTP messages have their specific worst case transmission time, noted as C_{PTP}. But, for the sake of readability, we consider that C_c and C_{PTP} have the same value (the highest one of the two). Even if it is a pessimistic assumption, we can consider it true as PTP and SCC messages are both configuration messages, defined with a small and close number of bytes in Ethernet protocol. Considering these hypotheses, we obtain (with F_{PTP}, the PTP synchronization frequency) :

$$I^n_{delay} = F_{PTP} * \sum_{\substack{j \in hp_c \\ \mathcal{P}_c \cap \mathcal{P}_j \neq \emptyset}} \left(S^{first_{c,j}}_{max_c} - M^{first_{c,j}}_c + A_{c,j} \right) + \sum_{h \in \mathcal{P}_c} \delta^h_c + (|\mathcal{P}_n| - 1) * (sl + 2 * C_c) \quad (8)$$

C. Reliable multicast

In order to update the criticality level information from the central node to all the nodes in the network, we must be sure that the criticality switch order is received by all the nodes and is executed preserving the consistency of the criticality level information. Thus, in order to preserve the consistency of the current criticality level in all the nodes, we need to guarantee a total order in the criticality updates: two consecutive criticality switches have to be executed in the same order in all the nodes. A reliable real-time multicast is a method to send the same information to all nodes in a network providing total order for the update of the criticality level in all nodes. In [6], the authors show how to build a real-time reliable multicast provided that worst case messages end-to-end delays can be bounded from above. In this paper, we adapt their solution to the context of mixed criticality management with the trajectory approach to compute worst case end-to-end message delays.

To implement this, we need a deterministic computation of the transmission delay of the information. Since we can assure a bounded transmission delay for information in each physical link of the network, we will be able to provide guarantees on transmission delays in the whole network. That builds the determinism of the delay.

Suppose a network $\mathcal{N}$, composed of a set of nodes $\mathcal{S} = \{S_1, S_2, ...S_n, ES_1, ES_2, ...ES_m\}$. We note M^n_{delay} the delay needed by node n to receive the reliable real-time multicast information from the central node.

Now, we can compute the transmission delay of the criticality-switch order (which is a message) from the central node to all the nodes in the network. This multicast delay is noted as M_{delay}.

Even if we implemented PTP, we need to consider clock accuracy ϵ_i for each node i, as it impacts the reliable multicast protocol [6]. The multicast delay of the whole network can then be deduced by taking the maximum value of accuracy on any node. We have $\epsilon = \max_{n \in \mathcal{N}}(\epsilon_n)$. We obtain [6]:

$$M_{delay} = \max_{n \in \mathcal{N}}(M^n_{delay}) + \epsilon \quad (9)$$

M^n_{delay}, for a node n, is then computed by the addition of different elements: the switching latency sl induced by

electronical transmission between two nodes, and the WCTT of the criticality-switch message, noted as C_o^i. For clarity purposes, we consider that the WCTT of the switch-criticality message is the same in each node: $\forall n \in \mathcal{N}, C_o = C_o^n = C_c$.

For a node i in the network with a central node (sink node) S_h , the delay needed to receive the order directly depends on the distance between i and S_h. We note this distance d_h. Furthermore, we make the hypothesis that the switching latency sl is the same for each physical link (like in IV-C), and so that $sl = 0$ms : the electronical delay generated by the distance between each node is null.

We then obtain the following expression of M_{delay}:

$$\begin{aligned} M_{delay}^n &= d_n * (C_c + sl) + \epsilon_n \\ M_{delay} &= \max_{n \in \mathcal{N}}(d_n) * (C_c + sl) + \epsilon \end{aligned} \tag{10}$$

As we are computing the multicast delay, we are computing the worst case delay needed for the farthest node from the sink node to receive the switch criticality order. At the end of this multicast delay M_{delay}, we are sure that all the nodes in the network received the criticality change information.

The criticality switch occurs on any node at its local time $t_m + M_{delay}$, where t_m (respectively M_{delay}) is the timestamp (the switching delay) sent by the sink node in the criticality switch multicast message. Hence when all nodes have received the criticality switch request, hence preserving the consistency of the criticality level information in all nodes. All the nodes switch almost at the same time, with a time difference bounded by ϵ the clock synchronization accuracy.

D. Criticality-switch message

Given the expression of I_{delay}^n(8) and M_{delay}(**??**), we obtain the global expression of the criticality-switch delay S_{delay} in the network. Given the hypotheses that the switching latency is constant, and that PTP, SCC and the reliable multcast message have the same WCTT (noted as C_c), we obtain:

$$\begin{aligned} S_{delay} &= F_{PTP} * \sum_{\substack{j \in hp_c \\ \mathcal{P}_c \cap \mathcal{P}_j \neq \emptyset}} \left(S_{max_c}^{first_{c,j}} - M_c^{first_{c,j}} + A_{c,j} \right) \\ &+ \sum_{h \in \mathcal{P}_c} \delta_c^h \\ &+ (2 * \max_{n \in \mathcal{N}}(d_n) - 1) * (C_c + sl) + C_c(\max_{n \in \mathcal{N}}(d_n) - 1) \\ &+ \epsilon \end{aligned} \tag{11}$$

When we compute the total delay needed to operate a criticality switch in the network, we then compute the delay represented by PTP synchronization messages, the non-preemptive effect induced by all messages in the network, the WCTT of criticality switch messages and finally the delay induced by electronical latency and clock jitter.

V. Simulation

In order to provide estimations of the transmission delay of criticality information inside a network, we provide simulation results using ARTEMIS [4]. To provide these results, we based our approach on the topology described on figure 1, and on randomly-generated tasksets to simulate traffic load in the topology. To generate these tasksets, we used the UUnifast generation algorithm presented in [9]. But as this method was designed for processor-context simulation, we first adapted it to network context.

A. UUnifast

The UUnifast method is a random tasksets generation method, first presented in [9] and used to generated tasksets in mono and multicore contexts. It consists in three steps :

- First, we generate random periods in a configurable time interval $[\epsilon; T]$, where ϵ is the clock accuracy(see IV) and T is the global simulation time. These periods can be of any size.
- On a second point, we generate a random value in $[0, 1]$ according to a uniform law, called the utilisation of a frame. It represents the individual load represented by the frame.
- Then, based on the generated period and utilisation, we compute the WCET of the task.

When it comes to adapt UUnifast method to a network context, it results in two different problems to solve. First, when we generate a random flowset, we have to specify a targetted load for the whole system (to compute the utilisation). But in the network, the load is different in each node. To solve this, we decided to focus on the load on the sink node of the topology : as it is the central node, we assume this is the one with the highest load, or at least with the most important one to focus on.

Secondly, we had to modify the UUnifast method in order to generate LO and HI critical messages. So we introduced a critical rate in the method which defines, randomly and according to this rate, the average number of critical messages inside the whole generated flowset. As we based our approach on critical periods change, each frame was defined either with just one LO-critical period, or with one LO and one HI critical period.

B. Impact of the load

We did generate different flowsets, each one representing a different scheduling scenario. We computed the utilization of each flow with a uniform distribution based on the network load, and finally deduce the WCTT of the flow. As we are working with ethernet(IEEE 802.3), the size of each frame in constrained in size between 64 and 1518 bytes. With a 100 Mb/s bandwith, all the WCTT in our network are bound between 4.9μs and 115,8μs.

We made a scenario with a set of 50 different flows (corresponding to a classical context of use). In our evaluation, we fixed a global simulation time of $t = 500\mu$s, which is enough to observe and bound the different delays we want to focus on. We made the load represented by the flows increasing from 0.4 to 1. The computed load was the one in the central node (S_4) which receives all the flows. With

these generated flows, we wanted to evaluate the impact of the network load on the criticality switch transmission delay.

Assigning the highest priority for mixed criticality management messages (dedicated for configuration messages) allows the criticality-switch messages to not be delayed more than once by messages with a lower priority (non-preemptive effect). We verified this hypothese by evaluating the MC delay switch as a function of the network load. Thus, we need to add to this delay the one due to PTP synchronization messages, considered with a higher priority (see IV).

We can observe in figure 3 that, basically, the delay of the criticality-switch as a function of the network load is linear.

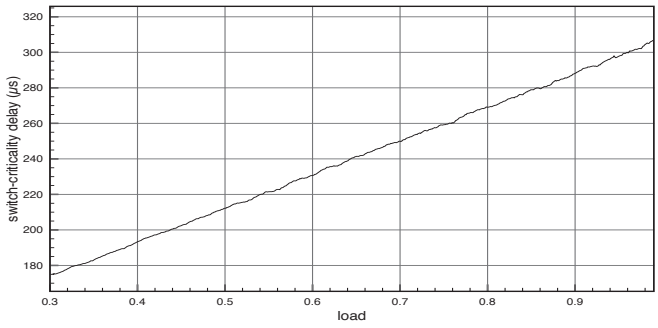

Fig. 3. Switch-criticality delay/load

We now analyse the impact of the non-preemptive delay for flows sent with a smaller priority than those of MC messages.

C. Non-preemptive effect

We picked the same parameters as for the previous scenario, but we also decided to put a limit on the highest WCTT in the network than for MC messages. We simulated 4 different cases with different limits in the highest WCTT.

The results obtained (see figure 4) shows that the criticality-switch delay is strongly influenced by the highest WCTT in the network. To evaluate this influence, we limited the highest WCTT to small sizes: 20 μs (262B), 30 μs (393B) and 40 μs (524B). We can observe that, at the highest loads, the transmission time is nearly constant.

The point is: in the UUnifast task generation method we presented in V-A , the task model bases the WCTT computation on the load. It means that the highest WCTT increases with the load. That explains why, in the first scenario without any particular limit, we obtained an increasing transmission time with the increasing load. On the contrary, when limiting the highest WCTT in the network, we obtain an constant switching delay (impacted from 1% to 6% in our examples at the highest network load, due to error margins we tolerated in the load computation).

As explained in IV, attributing to the switch-criticality and reliable multicast messages the highest priority allows us to make the criticality messages independant from the network load (the impact is very limited). No matter the traffic in the network, we can then compute the criticality switch delay in the network. As this switch delay is bound, the transmission of HI-critical messages can be assured in our topology in a bound and known time.

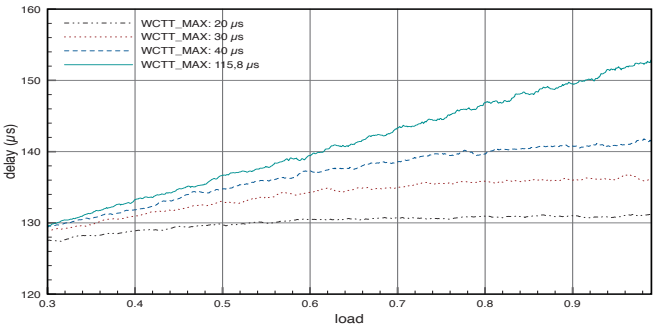

Fig. 4. Switch-criticality delay/load with limited highest WCTT

VI. Conclusion

A. Conclusion

In this paper, we present a criticality-change protocol in a clock synchronized Switched Ethernet network, in the case of two criticality levels. This criticality change protocol is based on a reliable real-time multicast used update the criticality information in all the nodes of the network while preserving its consistency. The real-time multicast builds a total order for the updates of the critality information. It relies on the based IEEE1588v2 global clock synchronization. We characterize the worst case delay for a criticality change with the trajectory approach. Through simulation, we generate random scenarios to test the time needed for a criticality change. We show that the criticality switch delay is harldy impacted by the network load. As a further work, we will show how to characterize the end to end response time of HI messages in the case of a LO to HI criticality switch.

References

[1] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed-criticality systems," in *RTSS 2011*.

[2] S. Baruah, "Mixed criticality schedulability analysis is highly intractable," 2015.

[3] A. Burns and R. Davis, *Mixed criticality systems: A review*. Department of Computer Science, University of York, 2013, vol. Tech. Rep.

[4] O. Cros, F. Fauberteau, L. George, and X. Li, "Simulating real-time and embedded networks scheduling scenarios with artemis," in *WATERS*, 2014.

[5] H.-T. Lim, L. Völker, and D. Herrscher, "Challenges in a future ip/ethernet-based in-car network for real-time applications," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024727

[6] L. George and P. Minet, "A fifo worst case analysis for a hard real-time distributed problem with consistency constraints," in *Proceedings of the 17th International Conference on Distributed Computing Systems*, 1997.

[7] S. Martin, P. Minet, and L. George, "End-to-end response time with fixed priority scheduling: trajectory approach versus holistic approach," in *International Journal of Communication Systems*, vol. 18, no. 1. John Wiley & Sons, Ltd., 2005, pp. 37–56.

[8] X. Li, O. Cros, and L. George, "The trajectory approach for afdx fifo networks revisited and corrected," in *RTCSA'14*, 2014.

[9] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Journal of Real-Time Systems*, pp. 129–154, 2005.