Formal Analysis Based Evaluation of Software Defined Networking for Time-Sensitive Ethernet

Daniel Thiele and Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig, Germany
{thiele,ernst}@ida.ing.tu-bs.de

***Abstract*—Software defined networking (SDN) aims to standardize the control and configuration of network infrastructure. It consolidates network control by moving the network's control plane to a (logically) centralized controller and downgrading switches to simple forwarding devices. This offers huge advantages for future automotive Ethernet networks, including admission control (e.g. to prevent/limit congestion) or network reconfiguration (e.g. in case of faults), both based on a centralized view of the current network state. SDN's centralized architecture, however, requires additional communication, which entails a certain overhead. If SDN is used in safety-critical real-time networks, this communication is subject to strict timing requirements. In this paper, we present a formal analysis based evaluation of the general suitability of SDN for time-sensitive networks including overhead, scalability, and timing guarantees by using a realistic automotive setup.**

I. Introduction

Ethernet is considered to become the communication backbone for future automotive networks, as traditional buses such as CAN or FlexRay cannot keep pace with the increasing bandwidth and scalability requirements of advanced driver assistance systems and infotainment systems. The introduction of Ethernet enables access to a large set of concepts and protocols from other Ethernet domains, such as SDN [1].

The main idea of SDN is the separation of a network's control and data planes. The data plane is responsible for frame forwarding, e.g. the switch fabric and flow (forwarding) table, while the control plane makes the actual routing decisions and configures the data plane accordingly. In SDN, the network's control plane (red components in Figure 1) is consolidated in a (logically) centralized SDN controller and only the data plane (blue components in Figure 1) remains in the switches. The SDN controller runs a network operating system, which manages and controls the entire network by configuring the switches. Inside an SDN switch, there is an SDN agent. This agent implements the communication with the SDN controller and updates the flow table inside the switch. The content of this flow table defines the forwarding operations performed by the switch's data plane, i.e. which incoming traffic stream (identified for example by source and destination addresses and/or port numbers) is forwarded to which output port.

Its centralized architecture allows SDN to realize globally optimal network management based on the current network state. As each switch communicates directly with the SDN controller, there is no need for complex (and potentially lengthly) distributed network control protocols (e.g. (rapid) spanning tree protocol or shortest path bridging), which first have to reach a common understanding of the network state before they can take appropriate actions. Applications of centralized network management include admission control (e.g. to prevent/limit network congestion by blocking non- or less-critical traffic) and run-time reconfiguration (e.g. to recover from faults in safety-critical networks by rerouting critical traffic around a failed component or link). The latter becomes increasingly important with the advent of highly automated and autonomous driving, where current fail-safe safety strategies must be replaced by fail-operational ones.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

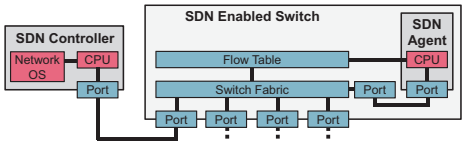


Fig. 1. SDN components

If SDN is used to manage real-time networks, it is subject to strict timing requirements. Admission control must be able to decide in bounded time if a traffic stream is allowed to enter the network. Safety-critical systems usually demand an upper bound on the fault recovery time. This is usually defined to be the time from the occurrence of a fault until it has been resolved. It comprises the time to detect the fault and the time to execute counter measures such as network reconfiguration. The AVnu Alliance suggests a fault recovery time of less than 100ms for critical automotive control applications [2]. Other sources suggest less than 50ms [3]. Real-time systems require the verification of these timing requirements. Here, formal worst-case analysis methods are typically chosen over simulation-based approaches, as simulation usually does not guarantee to expose all corner cases, whereas a formal analysis gives safe upper bounds on a system's worst-case behavior.

The **contribution of this paper** is the evaluation of the general suitability of SDN for real-time Ethernet. For that purpose, we discuss two protocols for basic SDN-based network configuration. We show how these protocols can be modeled and formally analyzed in a compositional performance analysis (CPA) framework [4]. Finally, we evaluate each protocol regarding its timing guarantees, impact on other traffic streams, and scalability by using a realistic automotive Ethernet setup.

II. Related Work

SDN is an emerging topic in time-sensitive Ethernet. The upcoming Ethernet TSN standards propose the introduction of SDN concepts. IEEE P802.1Qca [5] defines the explicit computation of (potentially redundant) paths by a (central) path computation element (PCE). In SDN terms, the PCE acts the SDN controller. IEEE P802.1Qcc [6] proposes centralized network configuration for time-sensitive networks. As these standards are still in their draft phase, we focus on the general suitability of SDN for Ethernet with real-time constraints.

Ethernet AVB's stream reservation protocol (SRP) [7] defines a distributed admission control scheme. SRP allows the reservation of network resources at the switches along a path from a sender to its receiver(s). However, the decision if a stream is allowed to enter the network is made locally at the involved switches. Particularly, there is no central control over when a sender is allowed to advertise a traffic stream or when a receiver is allowed to subscribe. Also, as there is no central controller with global network view in SRP, fault recovery cannot be implemented as efficiently as in SDN. Unlike SDN, SRP does not allow to (re)configure flow tables.

An extensive survey of SDN is given in [1]. The authors give a comprehensive overview of SDN, covering current standardization activities, interfaces, controllers, network programming languages, and applications. SDN interfaces define the communication between the SDN controller and the switches. The most prominent SDN interface is OpenFlow [8]. It defines a communication protocol (including the frame format) along with the flow table layout in the switches. OpenFlow, however, relies on TCP-based communication, which is less suitable for latency-critical real-time traffic because its complex handshake and flow control mechanisms contribute significantly to the overall latency. Real-time traffic is typically UDP-based.

In [3] a fast fault recovery mechanism for SDN is presented. This mechanism uses link-based bidirectional forwarding detection to detect link faults and OpenFlow's Fast Failover Group feature to reroute traffic. This feature allows a switch, in case of a link failure, to autonomously alter its flow table to a preconfigured backup route. As this failover mechanism is based on a purely local decision, the backup route might be suboptimal and typically involves crankback routing, i.e. returning traffic towards its source while trying alternative paths to reach the destination [3]. Because this recovery process does not involve communication with the SDN controller, it enables fast fault recovery (the authors measured 3.3ms in a simulation-based evaluation). Communication with the SDN controller might still be required to replace the preliminary backup route with a (better) final one. Naturally, this method cannot be applied to admission control. Also, a simulation-based evaluation cannot replace a formal timing verification.

A network calculus based formal analysis of the communication between an SDN controller and SDN switches focusing on SDN controlled (i.e. admission controlled) traffic is presented in [9]. The authors derive upper bounds on the frame delay in switches as well as the buffer sizes in the SDN controller. Compared to their analysis model, which models SDN controller and SDN switches (each) as single resources (network calculus servers), we present a much more elaborate analysis model. Our model considers that switches have, in fact, multiple ports, which, in turn, have multiple traffic queues (one per Ethernet priority), and that all ports of a switch share a single SDN agent (see Figure 1). We also explicitly model that the SDN controller and SDN agents have limited processing resources (CPUs), which, as we will see, contribute significantly to the timing guarantees of SDN traffic.

Specifically, we derive worst-case timing guarantees for two basic SDN network configuration protocols. In contrast to [9], we take into account that, as part of these protocols, the SDN controller might have to reconfigure the network by distributing new forwarding rules to individual switches and wait for their acknowledgment. This reconfiguration takes time and contributes significantly to a protocol's overall latency. Furthermore, we evaluate the overhead of introducing SDN, i.e. we evaluate the impact of SDN traffic on non-SDN traffic.

SDN performance has also been studied by using queueing theory (e.g. [10]). The probabilistic performance guarantees given by queueing theory, however, are unsuitable to derive worst-case metrics for real-time networks.

III. Compositional Performance Analysis

We use CPA [4] to analyze the worst-case behavior of SDN communication. The CPA system model comprises three components: resources, tasks, and event models. Resources model processing or network resources (e.g. CPUs or switch ports) and provide service according to a scheduling policy. Tasks are mapped to resources and compete for their service. Per activation, a task consumes service varying between its best- and worst-case execution time. Task activations are abstracted by event models. An event model is defined by a tuple of event arrival functions $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which give the lower and upper bounds on the number of events (task activations) in any half-open time interval $[t,t+\Delta t)$. In contrast to a specific event trace, an event model captures all possible event arrival scenarios within its bounds. A system is modeled as a directed graph, where tasks correspond to nodes and task dependencies are modeled by edges. Whenever a task finishes its execution, it propagates an event to its dependent task(s), i.e. its output event model becomes the input event model of its dependent task(s). If a task activation depends on events from multiple tasks, these events must first be joined by a junction with AND semantic [4]. Tasks without predecessors must be stimulated by an event model from an external source.

CPA is an iterative approach. Resources are analyzed by a *local analysis* based on the busy period approach [11]. In this approach, new output event models for each task are derived from a critical instant scenario. This scenario maximizes the response time of the currently analyzed task by activating interfering tasks with their worst-case activation (and execution) pattern. The response time jitter (maximum minus minimum response times) can be used to derive new output event models [4]. A *global analysis* loop propagates event models between dependent tasks. The analysis ends, if all event models reach a fixed-point (do not change anymore). The analysis also ends if predefined constraints (e.g. number of analysis iterations or end-to-end latencies etc.) are violated, in which case the system is considered to be unscheduable.

IV. Modeling and Formal Analysis of Software Defined Networking

The focus of this paper is the evaluation of the general suitability of the SDN concept for real-time networks. As SDN (exemplified by OpenFlow) was not designed with real-time requirements in mind, we propose a simplified (but analyzable) SDN scheme. We differ from OpenFlow in two key points: (a) As discussed in Section II, UDP is typically chosen over TCP to transport latency-critical traffic. Hence, we present two UDP-compatible protocols to realize SDN-based network control. (b) In Section V, we argue that certain messages are assumed to be smaller than in OpenFlow.

A. SDN-based Network Configuration

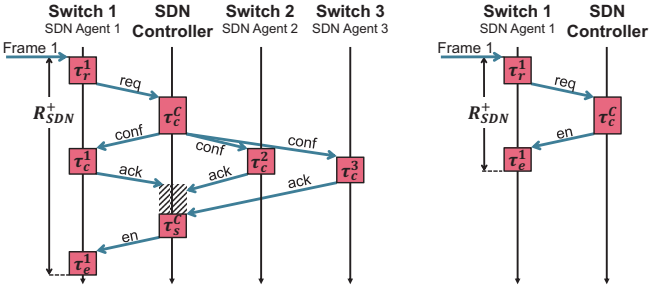
SDN network control is stream-based (also called *flow*-based in SDN literature). A traffic stream is defined to be a sequence of frames between a source and a destination, which receive identical service policies in the network [1]. In this

paper, we assume that the data plane (inside the switches) is controlled by flow tables similar to OpenFlow's [8]. Each flow table entry comprises three fields: rule, action, and statistics (frame counters etc.). Incoming frames are matched against the rules to determine the traffic stream they belong to, e.g. by their MAC and/or IP addresses, VLAN ID, or TCP/UDP ports. Rules are implicitly prioritized by their order of appearance in the flow table to resolve ambiguities. Actions define how a frame should be treated. Possible actions include: *forward* the frame to a specific switch port, *drop* the frame, and *request* further instructions from the SDN controller on how to handle the frame. Network configuration in SDN is the process of creating and distributing these flow table entries, e.g. when a new *request* has been received by the SDN controller.

We discuss the *explicit flow configuration* protocol in Figure 2a, exemplified by admission control. In an admission control scenario, traffic streams are either allowed to enter the network (*forward* action), blocked (*drop* action), or must request admission before entering the network (*request* action). We are interested in the worst-case timing behavior of the *request* action. When the first frame (Frame 1 in Figure 2a) of an incoming traffic stream arrives at Switch 1, this switch determines (based on its current flow table) that this stream requires admission control and generates a request (*req*) addressed to the SDN controller. The SDN controller receives the request and checks if this stream is allowed to enter the network. In case the stream is denied network access, the controller notifies Switch 1 immediately (not shown in Figure 2a). If the controller decides to grant network access to the stream, the network must be (re)configured to support this new traffic stream. For this, the SDN controller sends configuration messages (*conf*) to all involved switches to update their flow tables. These configuration messages can carry multiple flow table updates, so that, for example, new forwarding rules for the requesting traffic stream and dropping rules for other (now obsolete) traffic streams can be distributed concurrently. Each switch must confirm this update via an acknowledgment (*ack*) message to the SDN controller. The controller must wait for all acknowledgments (hatched time interval in Figure 2a) before it finally sends the confirmation (*en*) to Switch 1 to enable the flow table entries, which allow the requesting traffic stream to enter the network.

Each step of this protocol introduces a certain delay. The processing delay on the CPUs of the SDN agents and the SDN controller (the step's actual execution time plus interference from other requests) is (symbolically) indicated by the red boxes. Communication is done via Ethernet and experiences delay (own frame transmission times plus interference from other traffic streams) in the network, which is indicated by the blue arrows. The entire network configuration process from the arrival of the first frame of a traffic stream to its final admission has a certain *SDN configuration latency* R^+_{SDN}.

From the perspective of the requesting traffic stream, the network configuration latency R^+_{SDN} of the explicit flow configuration protocol can be *hidden* by configuring the network before this stream enters the network. However, this does not reduce the actual configuration latency and only works if the network change is known beforehand. In automotive systems, however, communication scenarios are typically predefined at design time. So, alternatively, all switches could be preconfigured to support all possible traffic streams. Admission control for a given traffic stream can then be realized by enabling



(a) Explicit flow configuration (b) Predefined flows

Fig. 2. SDN network (re)configuration protocols

or disabling a higher-priority dropping rule in the flow table of the switch at which this traffic stream enters the network. This reduces admission control to a simple *req*/*en* handshake, as shown by the *predefined flows* protocol in Figure 2b.

Fault recovery can be realized by the explicit flow configuration protocol, e.g. by switching to a set of preconfigured alternative flows for a particular fault. Instead of an arriving frame, the network reconfiguration process is triggered by the detection of a fault. Here, the switch sending the *req* message might be different from the one receiving the *en* message.

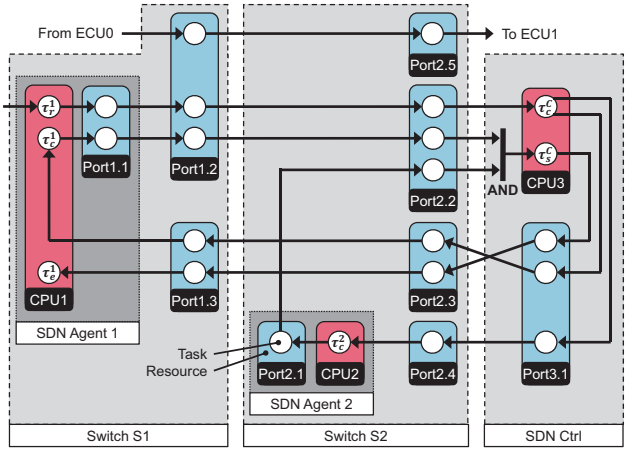
B. Modeling SDN in Compositional Performance Analysis

In this section, we explain how the previously discussed network configuration protocols can be modeled in CPA by using the example network in Figure 3a. There are two switches, two ECUs, and one SDN controller. We assume that only switch *S1* sends requests to the SDN controller. Additionally, there is one non-SDN traffic stream from *ECU0* to *ECU1*.

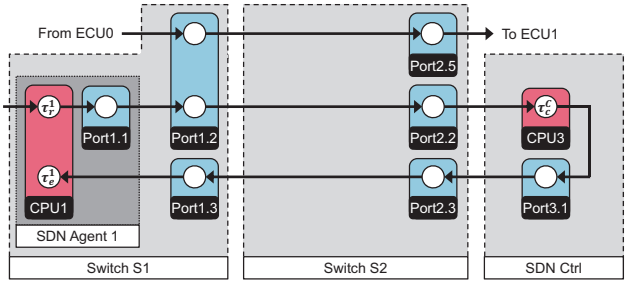
Figure 3b shows how the explicit flow configuration protocol from Figure 2a can be mapped to CPA primitives. We model the processing resources (CPUs) of the SDN agents in the switches and of the SDN controller as CPA resources. Each processing step of the protocol is modeled as a task executing on these resources, e.g. the generation of a request to the SDN controller is modeled by task τ_r^1 on resource *CPU1*. The execution times of these processing tasks depend on complexity of the work to be performed. The Ethernet communication follows the CPA model proposed in [12]: We assume that the arbitration inside switches takes place at the output ports and, hence, model only these output ports as CPA resources. Analogously, we model the Ethernet output ports of ECUs and SDN agents and controllers as CPA resources. *Port1.1*, for example, is the output port of SDN Agent 1 towards Switch 1. The transmission of a frame at an output port is modeled as a task executing on the output port's resource. The execution times of these communication tasks correspond to their frame transmission time, which, in turn, depends on the frame's size. The dependencies between individual protocol steps are modeled as task dependencies. Hence, the protocol can be modeled as a chain of tasks mapped to the corresponding processing and communication resources. The synchronization process at the SDN controller (hatched time interval in Figure 2a) is modeled by an AND-junction (i.e. wait for all inputs before proceeding) of the corresponding traffic streams in the CPA model [4]. The traffic stream from *ECU0* to *ECU1* shares the *Port1.2* resource with SDN traffic. Hence, there will be interference between SDN and non-SDN traffic.

CPA derives (among other metrics) worst-case execution times for each task, based on the interference from other tasks on the same resource [4], [12]. The SDN configuration

(a) Example network



(b) CPA model for the explicit flow configuration protocol from Figure 2a



(c) CPA model for the predefined flow protocol from Figure 2b

Fig. 3. CPA models for the SDN (re)configuration protocols from Figure 2

latency R^+_{SDN} can be derived by computing the end-to-end path latency from task τ_r^1 at SDN Agent 1 to τ_e^1. This latency is upper bounded by the sum of the worst-case execution times of all tasks along the path. Task τ_c^C forks the SDN request to trigger the reconfiguration of the individual switches. The forked paths are rejoined by the AND-junction preceding τ_s^C to generate the enable message. Hence, to compute an upper bound on the worst-case end-to-end path latency (i.e. a worst-case latency guarantee) from τ_r^1 to τ_e^1, we take the path latency from task τ_r^1 to τ_c^C, the maximum path latency over all forked paths from τ_c^C to τ_s^C, and the path latency from τ_s^C to τ_e^1.

Figure 3c shows how the predefined flow protocol from Figure 2b can be modeled in CPA by using the same principle. As expected, this model is less complex.

V. Evaluation

In this section, we evaluate the general suitability of the SDN network control concept for real-time Ethernet. As SDN's centralized network control introduces additional traffic, we evaluate both the SDN configuration latency, to investigate if the network can be configured in a timely manner, and the impact of the additional SDN traffic on non-SDN traffic (i.e. the traffic that is controlled by SDN), to quantify SDN's overhead on existing traffic. We explore, for both protocols, different parameters: the number of SDN requests per switch, SDN frame sizes, SDN processing times, and SDN traffic priorities. Our analysis is based on the CPA models introduced in Section IV-B. We use the worst-case latency guarantees derived from CPA as a comparison metric.

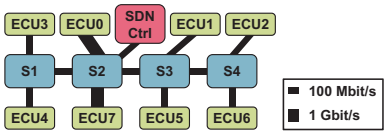


Fig. 4. Quad star topology

Our evaluation is based on the quad star topology in Figure 4. Eight ECUs are distributed throughout the network. Highly loaded links are connected via 1GBit/s. All other links are 100MBit/s links. The SDN controller is located near the network's center. Table I summarizes the normal (non-SDN) traffic in the network, which has been provided by Daimler AG. The traffic is categorized into control and camera traffic. Control traffic is assumed to be latency-critical and is mapped to a higher Ethernet priority than camera traffic, which we assume to have more relaxed latency requirements. There are different kinds of traffic streams in the network: unicast, multicast (the notation $n(d)$ specifies that there are n multicast streams with d destinations), and broadcast. For each traffic class, Table I gives its minimum, maximum, and average payloads and periods. We assume that IPv4/UDP is used to route data, so that an overhead of 28bytes must be added to the payload. Although traffic is considered to be periodic, we use a *periodic with jitter* event model [4] for traffic streams entering the network. To allow, in the worst-case, an occasional burst of two frames, we set the jitter to the period.

The SDN configuration latency depends on various parameters, which we explore in the following. Specifically, we investigate: (a) *The number of SDN requests per switch:* In our evaluation, each switch sends a certain number of independent SDN requests to the controller. We assume that, in the worst-case, these requests are generated and sent concurrently. Each request is modeled by a periodic event model with a period of 1s, e.g. for a periodic 5ms stream every 200th frame would cause an SDN request. We also assume that each request requires all switches in the network be updated with new forwarding rules. (b) *The size of SDN messages:* OpenFlow, for example, defines different message types (e.g. to send requests to the controller or to reconfigure flow tables), which can be of variable size (depending, e.g., on the number of flow table entries to modify or the number of actions). Thus, we also evaluate different SDN message types and sizes. We assume that there are two different types of SDN messages. In OpenFlow, the requesting frame (e.g. Frame 1 in Figure 2) can be sent as part of the request message (PacketIn message in OpenFlow) payload to the SDN controller to aid decision making. This, however, leads to larger SDN overhead, as larger frames potentially cause more/longer interference in the network. In this evaluation, which is in the context of real-time systems, we try to reduce the SDN overhead and assume that request, acknowledge, and enable messages only carry enough information to unambiguously identify a traffic stream,

TABLE I
Traffic description

	Control	Camera
Unicast Multicast Broadcast	26 13(2), 4(3), 1(4) 6	4 1(2) 0
Payload (bytes)	[1, 250]	[875, 1400]
Average (bytes)	54	1160
Period	[5ms, 1s]	[100us, 1ms]
Average	182ms	372us

e.g. MAC and/or IP addresses, VLAN IDs, port numbers, etc. and not the entire requesting frame. We evaluate these small SDN messages for sizes of 16 and 32bytes. SDN configuration messages, in contrast, must provide enough room to carry new switch configuration data, e.g. flow table entries (rules to match traffic streams (including wildcards) and actions), and are typically larger. We evaluate configuration messages for sizes of 64, 128, and 256bytes (OpenFlow's FlowMod messages are in this range). Again, we assume IPv4/UDP encapsulation, which introduces an additional overhead of 28bytes. (c) *The execution time on SDN processing resources:* We assume that there are two different types of execution times. At the SDN agent side the tasks to generate the request (τ_r^1), update the flow table (τ_c^1), and enable forwarding (τ_e^1), as well as the synchronization task on the SDN controller (τ_s^C) are assumed to require less processing time than the task on the SDN controller that decides whether a request is allowed to enter the network or not (τ_c^C). We assume that the execution times for the former tasks are in the range of [20us,30us] and for the latter task in the range of [30us,50us]. As the execution times highly depend on the implementation complexity and the underlying processor architecture and speed, we additionally evaluate the SDN configuration latency if these execution time ranges are cut in half, i.e. [10us,15us] and [15us,25us], respectively. (d) *The priority of SDN traffic:* In Ethernet, SDN communication uses the same network infrastructure as non-SDN traffic. Consequently, the Ethernet priority of SDN traffic has a decisive impact on the SDN configuration latency, as it determines how much interference it can experience. Likewise, the impact of SDN traffic on non-SDN traffic also depends on this priority. We investigate the impact of SDN's Ethernet priority by comparing setups where SDN traffic is mapped to the highest Ethernet priority (higher than control traffic), where SDN traffic shares a priority with control, and where it shares a priority with camera traffic.

We use standard Ethernet (IEEE 802.1Q) to arbitrate frames in the Ethernet network [13]. We also implemented the optimization proposed by [14], which considers Ethernet links as (physical) traffic shapers. The scheduling on the CPUs of the SDN agents and the controller is assumed to be static-priority preemptive scheduling, where tasks of equal priority are processed in their activation order. While the priorities of these tasks can be used to prioritize requests, we set their priorities to the Ethernet priorities of their SDN messages.

In Figures 5, 6, and 7, we present our evaluation results as boxplots (even though this is not a random experiment). *Observe the different scale of the y-axes*. Each box summarizes the worst-case latency guarantees of all analyzed traffic streams (i.e. SDN traffic streams in Figures 5 and 6, and non-SDN traffic streams in Figure 7). The actual box covers 50% of the latency guarantees. The lower and upper edges are the 25% and 75% quartiles and the antennas mark the worst-case guarantees of the streams with the lowest and highest latency guarantees. The median guarantees are marked by a black line inside the box. The x-axis labels follow the scheme *X,N/M*. *X* specifies whether full (*F*) or half (*H*) execution times are used for SDN processing tasks. $N \in \{16,32\}$ and $M \in \{64,128,256\}$ specify the frame size of SDN messages. Due to space constraints, we focus on the explicit flow configuration protocol and only highlight the differences to the predefined flow protocol, which, in general, shows similar behavior, but with significantly lower guaranteed latencies.

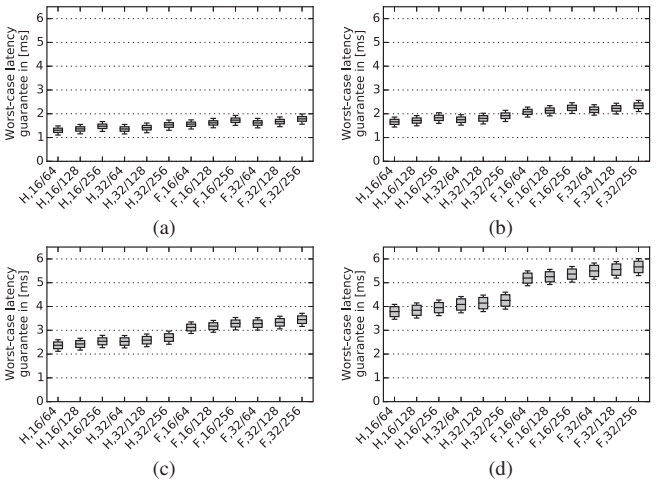


Fig. 5. **Worst-case SDN configuration latency guarantees** for the explicit flow configuration protocol with SDN traffic mapped to the highest priority for (a) 1, (b) 2, (c) 4, and (d) 8 SDN requests per switch.

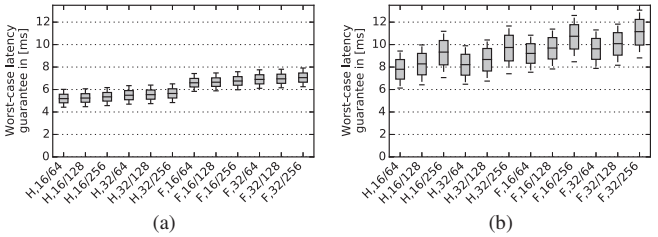


Fig. 6. **Worst-case SDN configuration latency guarantees** for the explicit flow configuration protocol with 8 SDN requests per switch. SDN traffic is mapped to the same priority as (a) control and (b) camera traffic (cf Figure 5d).

Figure 5 shows the *worst-case SDN configuration latency guarantees* for the explicit flow configuration protocol with SDN traffic mapped to the highest Ethernet priority for an increasing number of SDN requests per switch. As expected, the SDN configuration latency guarantees grow as the number of SDN requests per switch increases. However, in our setup, even with 8 SDN requests per switch, i.e. many concurring SDN requests, they always stay below 6.1ms. The impact of the SDN message sizes on the SDN configuration latency guarantees is visible, but relatively small. This is because the SDN configuration latency does not solely depend on the transmission times of SDN Ethernet messages, but also on the amount of blocking by lower-priority Ethernet traffic. In the worst-case, the largest lower-priority Ethernet frame can block high-priority SDN traffic once (per busy period), due to the non-preemptive link access in IEEE 802.1Q [13]. Also, in our setup, the SDN processing requirements at the SDN controller and the SDN agents are independent of the SDN message size and contribute about 40% (half execution times) and 60% (full execution times) to the SDN configuration latency guarantees. Reducing the execution times of SDN's processing requirements (*F* vs. *H* setups in Figure 5), e.g. by using faster processors, can significantly lower the SDN configuration latency guarantees, especially for an increased number of SDN request per switch. The predefined flow protocol generally shows similar behavior, but with lower worst-case latency guarantees. For 1, 2, 4, and 8 SDN request streams per switch, the configuration latency bounds are 0.6ms, 0.8ms, 1ms, and 1.3ms, respectively. It is independent of the *conf* message size.

From a safety perspective, transmitting all SDN traffic on the highest priority might not be desirable, as, this way, lower-

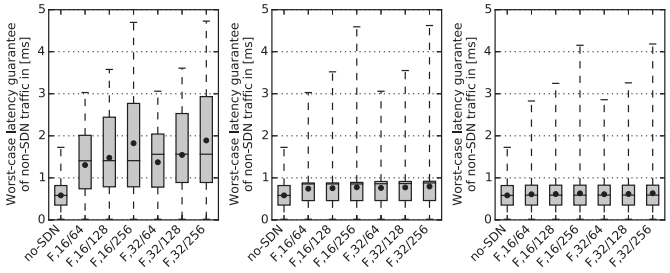


Fig. 7. Impact of SDN traffic with 8 SDN requests per switch in the explicit flow configuration protocol on the **worst-case latency guarantees of non-SDN traffic**. SDN traffic is mapped to (a) the highest Ethernet priority, (b) the priority of control traffic, and (c) the priority of camera traffic.

priority non-SDN traffic could flood the network with high-priority SDN requests. Mapping SDN traffic to the priority of the requesting (non-SDN) traffic stream can mitigate this problem, but entails longer SDN configuration latencies. Figure 6 shows the impact of the SDN traffic priority on the *worst-case SDN configuration latency guarantees* for the explicit flow configuration protocol with 8 SDN requests per switch. Unsurprisingly, the lower the priority of SDN traffic, the higher the worst-case SDN configuration latency guarantees. The increase of the SDN configuration latency guarantee depends highly on the non-SDN (camera and/or control) traffic. In Figure 6a SDN traffic experiences same-priority interference from itself and control traffic as well as (limited) lower-priority blocking from camera traffic. In contrast, in Figure 6b it experiences same-priority interference from itself and camera traffic, while also experiencing control traffic as higher-priority interference. For the explicit flow configuration protocol the worst-case SDN configuration latency guarantees are 8ms and 13ms for SDN traffic on control and camera priority, respectively. For the predefined flow protocol, the bounds are 2.3ms and 2.8ms.

Figure 7 shows the impact of SDN traffic with 8 SDN requests per switch in the explicit flow configuration protocol on the *worst-case latency guarantees of non-SDN traffic*, i.e. control and camera traffic, for different SDN traffic priorities. We only show the setups with full execution times. The worst-case end-to-end latency guarantees of camera and control traffic in a network without interference from SDN traffic, i.e. without SDN control, are summarized by the *no-SDN* box. Compared to the no-SDN case, if SDN traffic is on the highest priority, the worst-case end-to-end latency guarantees of non-SDN traffic increase on average (average values are marked with a black dot) by 1 to 1.5ms, while the peak values increase by at most 3ms. In Figure 7a, the impact of the different SDN *conf* message sizes, which are transmitted on the highest priority, on the lower-priority traffic is clearly visible. Mapping SDN traffic on the same priority as control traffic significantly reduces the worst-case latency guarantees of most non-SDN traffic streams (see Figure 7b). Recall that most non-SDN traffic streams are control streams (see Table I), which now benefit from the reduced priority of SDN traffic. This is because the optimization proposed by [14] reduces the overestimation of same-priority interference. The peak latency guarantees, however, are not reduced. These guarantees are from camera traffic, which, in this setup, still experiences SDN traffic as higher-priority interference. Figure 7c shows that camera traffic does benefit from mapping all SDN traffic to its priority level, but the improvement is comparatively small. This is because the period of camera traffic streams is close to the transmission times of SDN traffic frames. The transmission time of a single 256byte SDN *conf* message (about 26us) is about a quarter of the shortest camera traffic period (see Table I). Hence, the presence of multiple SDN traffic streams (8 per switch in Figure 7) increases the camera traffic backlog (i.e. the number of frames waiting for service at a switch port), which causes large worst-case latency guarantees. Note that the control traffic in Figure 7b is not affected by this backlog build up, as control traffic periods are large compared to the transmission times of SDN messages. In Figure 7c, control traffic, which makes up most of the traffic streams, benefits slightly, as can be concluded from the subtly lower median worst-case latency guarantees (compared to Figure 7b). For the predefined flow protocol, all worst-case latency guarantees of control and camera traffic are upper bounded by 2ms, i.e. the overhead of this protocol on non-SDN traffic is very small.

VI. Conclusion

Software defined networking (SDN) is an emerging topic in real-time Ethernet and SDN concepts can be found in the upcoming Ethernet TSN standards. SDN decouples a network's control and data planes and introduces centralized network control to unify network configuration. In this paper, we investigated the general suitability of SDN for real-time Ethernet. We showed how SDN communication can be modeled and analyzed in a compositional formal analysis framework to derive worst-case bounds for network configuration latencies. We used a typical automotive setup to evaluate SDN's scalability and its overhead on non-SDN traffic. Our results show that worst-case network configuration latency guarantees well below 50ms are possible, which allows to use SDN for admission control and fault recovery in Ethernet.

References

[1] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, Jan 2015.

[2] J. Takeuchi *et al.*, "Requirements for Automotive AVB System Profiles," in *AVnu Alliance Whitepaper*, March 2013.

[3] N. van Adrichem, B. van Asten, and F. Kuipers, "Fast Recovery in Software-Defined Networks," in *European Workshop on Software Defined Networking (EWSDN)*, Budapest, Hungary, September 2014.

[4] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - the SymTA/S Approach," in *IEE Proceedings Computers and Digital Techniques*, 2005.

[5] IEEE Time-Sensitive Networking Task Group, "P802.1Qca (Draft 2.1) - Path Control and Reservation," http://www.ieee802.org/1/pages/802.1ca.html.

[6] ——, "P802.1Qcc (Draft 0.4) - Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," http://www.ieee802.org/1/pages/802.1cc.html.

[7] IEEE Audio Video Bridging Task Group, "802.1Qat - Stream Reservation Protocol," http://www.ieee802.org/1/pages/802.1at.html.

[8] Open Networking Foundation, "OpenFlow." [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

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

[10] K. Mahmood, A. Chilwan, O. sterb, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, 2015.

[11] K. W. Tindell, A. Burns, and A. J. Wellings, "An extensible approach for analysing fixed priority hard real-time tasks," *Real-Time Systems Journal*, vol. 6, pp. 133–151, 1994.

[12] J. Diemer, J. Rox, and R. Ernst, "Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis," in *MATHMOD - Vienna International Conference on Mathematical Modelling*, Vienna, Austria, 2012.

[13] J. Diemer, D. Thiele, and R. Ernst, "Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching," in *IEEE International Symposium on Industrial Embedded Systems*, 2012.

[14] P. Axer, D. Thiele, R. Ernst, and J. Diemer, "Exploiting Shaper Context to Improve Performance Bounds of Ethernet AVB Networks," in *Proceedings of DAC*, San Francisco, 2014.