Software-defined Flow Reservation: Configuring IEEE 802.1Q Time-Sensitive Networks by the Use of Software-Defined Networking

Tim Gerhard
Institute of Telematics
Karlsruhe Institute of Technology
Karlsruhe, Germany
tim.gerhard@kit.edu

Thomas Kobzan
Fraunhofer IOSB-INA
Lemgo, Germany
thomas.kobzan@iosb-ina.fraunhofer.de

Immanuel Blöcher, Maximilian Hendel
Innovation Technology
Hilscher Gesellschaft für Systemautomation mbH
Hattersheim, Germany
{ibloecher,mhendel}@hilscher.com

***Abstract*—Communication systems are core elements of future cyber-physical systems (Industrial Internet, Industry 4.0, Internet of Things) which form the basis for applications such as smart production, smart grid, smart city, and the like. These applications will comprise data streams with very different communication requirements regarding data rate, latency, jitter, and reliability that may also vary over time. This calls for communication solutions that on one hand can adapt flexibly and on-demand to these needs and on the other hand can seamlessly support data streams with different quality-of-service requirements.**

Within this paper we present Software-defined Flow Reservation (SDFR), an approach that combines Software-defined Networking (SDN) technologies with Time-Sensitive Networking. Special focus is on the configuration of the time-sensitive network. SDFR implements the IEEE 802.1Qcc standard in the logically centralized SDN controller and, thus, allows for a flexible software-driven configuration of the underlying network.

This flexibility enables operators to support multiple configuration interfaces simultaneously and to easily modify traffic classes, scheduling approaches, and network behavior. SDFR allows to re-use existing solutions for well-understood SDN use cases while simultaneously supporting novel TSN-specific solutions.

***Index Terms*—Software-Defined Networking, Time-Sensitive Networking, Network Management, Industrial Internet**

I. Introduction

With the integration of Internet of Things (IoT) and cyber-physical systems (CPS) into future production facilities, flexibility is a key driver for innovation. One important aspect for this is the self-configuration of networked end devices as well as the network itself: applications on end devices announce their communications needs to the network's control plane, which provides quality-of-service (QoS) communication for the given streams. Furthermore, the network must perform fast reconfiguration, i.e., it needs to react to changed configurations with a reasonably short delay during runtime. While these are no new requirements for state-of-the-art management-level networks, it is a more demanding task on process control and field levels of a network.

German Federal Ministry of Education and Research within the scope of the Project FlexSi-Pro, grant number: 16KIS0650K.

Our vision is an Ethernet-based network which can provide low-latency, low-jitter, and no-loss guarantees to individual streams, while also supporting best-effort and other kinds of traffic. Existing Ethernet-based solutions for real-time capable networks are usually incompatible to standard Ethernet, incompatible to each other [1], and often subject to proprietary licenses, which makes them unfit for a vision of an open network infrastructure. With *Time-Sensitive Networking* (TSN), the IEEE has extended its already existing effort for QoS traffic in order to support real-time or deterministic data streams – however, their latest standards require before-hand configuration. The recent IEEE 802.1Qcc-2018 standard [2] specifies on-demand TSN configuration on an architectural level and provides a data model for stream registration; however, it only gives recommendations and does not provide concrete specifications regarding actual techniques for configuration, e.g., protocols. More precisely, in the models with centralized network controllers, the open questions of the 802.1Qcc standard are: Which protocol should be used for signaling stream reservations from end devices to the network? What algorithms should be used to calculate routes and schedules based on this information? And which protocol can be used for deploying the resulting routes and schedules?

We argue that OpenFlow is a fitting solution for the last question, and that using it enables a high degree of flexibility in the other two questions. While much research has been done on scheduling algorithms for time-triggered traffic inside an SDN controller, we are not aware of an architecture that actually provides the components specified in the IEEE 802.1Qcc standard in an SDN-based network. Our key contribution is the *Software-defined Flow Reservation* (SDFR) architecture, which fills this gap while allowing an easy integration and simultaneous support of any number of scheduling or registration approaches over well-defined interfaces. We use existing SDN controller software and SDN southbound protocols to define a fully functioning TSN configuration infrastructure in accordance with 802.1Qcc. Furthermore, we present our first working demonstrator of this architecture.

This paper is organized as follows: sec. II presents the basis

978-1-7281-0303-7/19/$31.00 ©2019 IEEE

our work is built on. Sec. III describes the SDFR architecture, and sec. IV presents our proof-of-concept implementation for time-triggered traffic, which is evaluated in sec. V. Sec. VI discusses related work, and sec. VII concludes this paper.

II. Basics

A. TSN Stream Registration

IEEE's TSN task group is working on new standards or extensions of previous ones which specify certain enhancements for networks to enable deterministic guarantees like low packet loss, bounded latency, or low jitter. This set of standards is also called *TSN*. TSN specifications can mostly be found in the IEEE 802.1Q [3] standard. At the time of writing, TSN consists of 7 published base standards, plus some more standards which are currently in specification. The current standards are undergoing revisioning, and amendments are being added.

1) Time-Sensitive Streams: A *time-sensitive stream* is a stream of real-time communication data frames sent from one *talker* to one or multiple *listeners*. In general, time-sensitive streams are considered to be multicast streams. To be transmitted with real-time guarantees, time-sensitive streams must be registered a-priori on the network.

When registering, a time-sensitive stream is identified by a *Stream ID* as known from the *Stream Reservation Protocol* (SRP) [3]. Frames of a certain time-sensitive stream can be identified by a *data frame specification* which specifies information like MAC addresses, IP addresses, or Transport protocol ports. The traffic of a time-sensitive stream is formed as specified in the *traffic specification*. Time-sensitive streams are subject to *user-to-network requirements* like bounded latency or bounded jitter.

2) Network Configuration Architecture: The application that uses time-sensitive streams is called the *user application*. In 802.1Qcc, it is assumed that it can provide the so-called *user/network configuration information*. The purpose of TSN configuration is to generate and deploy a *bridge configuration* based on this information. For this to happen, the user/network configuration information must be transmitted from user applications to the network by using a *User/Network Interface* (UNI). 802.1Qcc defines three different architectural models for realizing this.

In the *fully distributed model*, configuration happens in a decentralized fashion, with user applications on talkers and listeners transmitting information via end-to-end signaling protocols, which serve as UNI. From this, the network's bridges configure themselves appropriately and may change the content of UNI messages to provide feedback to end devices. This is how SRP was specified originally.

The *centralized network/distributed user model* defines a component called *Centralized Network Configuration* (CNC) which configures bridges via a management protocol. The user/network configuration information is transmitted from talkers and listeners to their neighboring bridges via the same UNI protocols, but in this case, the first bridge will forward the frames to the CNC. The CNC will process the gathered information and configure the network's bridges accordingly, and has frames sent to other end devices according to the rules of the UNI protocol.

The *fully centralized model* additionally defines a *Centralized User Configuration* (CUC) which contains the user/network configuration information and transmits it to the CNC. It is up to the user application to ensure that the CUC has access to all required information. Communication between the CUC and other components may be used for this purpose, but is not considered to be an issue of TSN configuration.

3) User/Network Configuration Information Data Model: User/network configuration information is encoded in *Talker groups* or *Listener groups*. A Talker group is a group of values that describes the intent of a single talker to transmit a single time-sensitive stream, a Listener group is the intent of a single listener to receive a single time-sensitive stream. Talker groups and Listener groups are loosely coupled and matched only by the Stream ID. Together, they contain the user/network configuration information for their respective time-sensitive stream.

Talker groups are usually transmitted in *Talker join messages*; Listener groups in *Listener join messages*. With join messages, talkers and listeners can announce their intent to start sending or receiving a time-sensitive stream, respectively. Talker leave messages and Listener leave messages can be used to leave a time-sensitive stream.

Once a Talker group or Listener group has been registered, the network can respond with *Status messages*, including *Status groups*, which describe the current state of the time-sensitive stream from the network's perspective. Status groups contain information on the status of talker and listeners, i.e., whether there is a talker and at least one listener present, and whether the network can fulfill the user-to-network requirements for at least one listener. Furthermore, Status groups contain configuration information for *stream transformation*. This means that the talker's network interfaces may be required to add a VLAN Tag or an 802.1CB label to the frames before transmission, which needs to be removed later by listeners' network interfaces.

B. Software-Defined Networking

Conventional network devices like switches contain two different planes located on the same device: the *data plane* and the *control plane*. The control plane is responsible for instructing the data plane how to forward datagrams, i.e., writing a forwarding table for usage by the data plane. The data plane forwards incoming datagrams to an egress interface based on that forwarding table.

1) SDN Architecture: Having the control plane distributed over all network devices has drawbacks regarding the complexity of network management as well as compatibility issues between network devices. *Software-defined networking* (SDN) tackles this issue by separating the control plane from the data plane, so that one logically centralized *SDN controller* sets the forwarding tables for all *SDN switches* in its *SDN domain*. Since most SDN switches can match link layer, network layer,

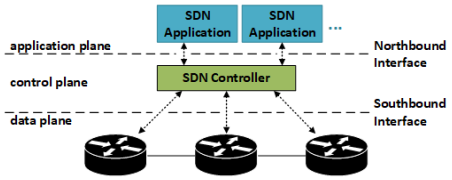

Figure 1. Basic SDN architecture

and transport layer headers of a single frame, there is no distinction between bridges, switches, and routers – there are only SDN switches.

SDN switches connect to their SDN controller over a network via a *southbound interface*. *OpenFlow* [4] is the dominant southbound interface protocol supported by most switches. Therefore, the remainder of this section shall describe SDN based on OpenFlow. Fig. 1 illustrates the SDN architecture.

The behavior of SDN controllers is defined by so-called *SDN applications*. SDN applications connect to the SDN controller via the controller's *northbound interface*. This can be realized as a network API, or a plug-in system for the controller, or both, depending on the controller's architecture. Usually, SDN applications for common features like topology discovery or keeping track of known hosts are included in the SDN controller. The boundaries of different SDN applications become blurry as apps depend on each other, similar to dependencies between UNIX programs.

2) Flow Tables: SDN operates on basis of so-called *flows*, which are a sequence of frames sent from a source to a destination and do not need to be distinguished further by forwarding devices.

The forwarding tables in SDN switches are called *flow tables*. The entries, *flow rules*, usually consist of tuples of matching rules and instructions. An incoming datagram is matched to the highest-priority flow rule to which the matching rules apply. Once a flow rule has been found, the switch executes the flow rule's instruction for the datagram, like selecting an egress port. The set of instructions is only limited by the capabilites of the switch, and may include features like adding/removing VLAN tags or MPLS labels, or setting a new IP destination address.

Usually, SDN switches are configured to forward unmatchable frames (or parts of it) to the SDN controller for examination. The SDN controller examines these frames, and sends own instruction to the switch. Usually, the SDN controller also alters the flow table in this case so that the switch can handle future frames without this redirection. This mode of operation is called *reactive flow programming*.

Using reactive flow programming results in an increased network load and additional latency for the first frame of a flow due to the redirection. To avoid this, *proactive flow programming* can be used by ensuring that switches' flow tables always contain the required entries for processing frames of a new flow. However, this means that new flows must be known a-priori.

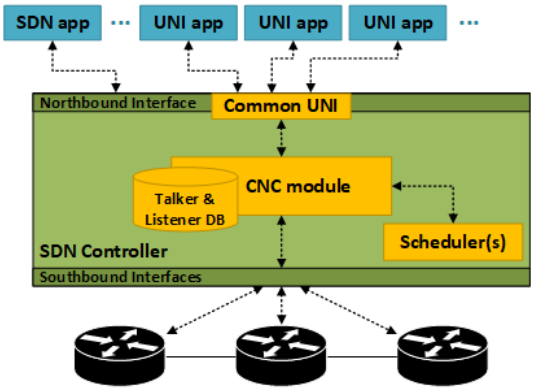

Figure 2. SDFR architecture

III. Software-defined Flow Reservation Architecture

Software-defined Flow Reservation (SDFR) aims at realizing TSN configuration as specified in the IEEE 802.1Qcc standard in an SDN-based architecture.

Having a centralized SDN controller, SDFR only uses the centralized network configuration models of 802.1Qcc, i.e., a CNC is always present. Moreover, the CNC, which is responsible for managing the network's real-time resources and the registration of time-sensitive streams, has a kind of functionality that should be included in the SDN controller. Therefore, in SDFR, the CNC is a module of the SDN controller, called the *CNC module*. Since time-sensitive streams are realized as flows in SDN, we use the term *time-sensitive flows* in the context of SDFR.

Fig. 2 shows the basic architecture of SDFR. In the following, all parts of this architecture will be discussed.

A. CNC Module

UNI messages are exchanged with SDN apps via the so-called *common UNI*, which will be discussed later. From incoming UNI messages, the CNC module creates and maintains a database of active Talker and Listener groups and matches them to time-sensitive flows. Once a change of a time-sensitive flow is detected, or when other conditions, e.g. a network link failure, require recalculation, a scheduler is used to calculate a schedule for all time-sensitive flows. This schedule is then converted into a network configuration, which is deployed via the controller's southbound interface. Afterwards, Status messages are communicated back over the common UNI.

The CNC may support many different traffic classes for time-sensitive flows. Schedulers, network configuration, and configuration deployment strategies may be different per traffic class. It is advised to have a clear separation of concerns between traffic classes, allowing network operators to add or remove support for traffic classes, and allowing an easy exchange the control and scheduling strategies for individual traffic classes. Furthermore, the CNC module needs to have policies for assigning network resources to different traffic classes.

Traffic that is not part of time-sensitive streams can be handled according to well-known SDN mechanisms, separated from time-sensitive flows. The priority of time-sensitive flows' flow rules needs to be set higher than that of others, and the data path may be required to process frames of time-sensitive flows, including flow table matching, in real-time. Again, this is a clear separation of concerns between traditional network traffic and TSN-specific network traffic.

B. Common UNI and UNI Apps

While an SDN controller usually provides only basic helper services, actual network behavior is defined by SDN apps. SDN controllers provide a northbound API for SDN apps to access the controller's functionalities and services. SDN apps are, for example, responsible for handling network signalling protocols or making routing decisions. SDFR's common UNI is an interface that allows SDN applications to exchange Talker, Listener and Status messages with the CNC module according to the UNI specification in 802.1Qcc. The northbound interface is the place in an SDN architecture to put this functionality.

In SDFR, SDN apps which access the common UNI to exchange user/network configuration information with the CNC module are called *UNI apps*. The CNC module allows to use any number of UNI apps at the same time. Status messages are always semantically associated with a Talker message or a Listener message, and the common UNI needs to select the correct UNI app to forward Status messages to. We expect most UNI apps to be protocol handlers that forward and translate messages between a UNI protocol used in the network and the common UNI. UNI apps can adhere to the centralized user model, to the distributed user model, or act as a hybrid thereof. In fact, there is no need to distinguish between these user configuration models from a network-architectural perspective in SDFR. UNI apps may provide their own APIs, connect to other APIs, or use the SDN controller's northbound interface.

In the following, we will describe how the two user configuration models of 802.1Qcc can be realized by UNI apps in SDFR.

1) Distributed user model: A common pattern in SDN is forwarding frames of network signaling protocols over the controller to SDN apps. For this, the SDN app installs flow rules for matching the UNI protocol on all switches, containing actions to forward the frame to the controller. The frame is not forwarded any further on the data path if no such action is present in the flow rule. Once the frame has arrived on the controller, the SDN app is notified about this event and may inspect the content of the frame and do actions based on this. For communication back to the end stations, the SDN app can insert any data frames to the data path, and provide instructions about where to forward this data to. In the SDFR architecture, the distributed user model can be realized with UNI apps that implement this SDN pattern.

2) Centralized user model: In the SDFR architecture, a CUC can be realized as a UNI app. Since SDFR supports multiple UNI apps, a CUC may not contain all user/network configuration information.

3) User model independence: Since SDFR moves UNI functionality to UNI apps, it is independent of how these UNI apps receive user configuration. Therefore, there is no need to distinguish between these two user models, and UNI apps may use either the SDN controller or own network interfaces to communicate with network devices, or both of these simultaneously. This means that UNI apps do not need to strictly adhere to either the centralized or the distributed user model.

C. Routing, Traffic Shaping, and Scheduling

When adding or modifying a time-sensitive flow, the CNC will do *routing* as a first step, i.e., selecting a path through the network from the talker to listeners. In the case of time-sensitive flows, the selection of paths is restricted to paths that can fulfill the flow's user-to-network requirements.

Traffic shaping is a feature aiming to shape the rate at which frames are transmitted. Oftentimes, this means that bursts of traffic are shaped into a constant bit rate by limiting the rate at which frames of a flow can be transmitted. *Scheduling* is a special kind of traffic shaping where the transmission of frames is planned thoroughly. This requires a complex calculation, but results in a schedule which selects the point in time for the transmission of each individual frame, thus reducing jitter to a minimum if the flow is subject to a regular transmission interval.

Since these steps may be intertwined, especially scheduling and routing, we call the sum of these calculations *scheduling*, the respective architectural component is a *scheduler*, and the result of this process is a *schedule*. This decision gives us the most flexibility, since a scheduler component can also realize the more simplistic kinds of calculations, but a more simplistic component may struggle to do a full scheduling.

Schedulers can be stateful in a way that they can re-use old configurations or partial results of earlier calculations for speeding up computation. Since the scheduling process may involve computationally complex algorithms, the scheduler may be realized as an external service on high-performance infrastructure. We use a black-box view on the scheduler as we consider the actual scheduling approach out-of-scope of the SDFR architecure.

D. Southbound Interface and Data Path

The fact that the controller has knowledge of all time-sensitive flows a-priori due to registration over the UNI, and in order to keep latency and jitter as low as possible, the CNC module operates only in proactive flow programming mode for time-sensitive traffic. When a schedule is converted into data path configuration, the CNC module can decide about control strategies like matching criteria, adding VLAN tags, and flow aggregation.

The common data plane operation in TSN is that frames are matched to their time-sensitive stream by matching the data frame specification from the respective Talker group.

Once matched, the switch needs to apply ingress policing, i.e., verifying that the frame does not violate the traffic description of its stream. Then, the frame is queued to the respective egress queue on the target egress interface. After this point, the association of a frame to the flow becomes irrelevant. Once in a queue, the frame is queued until it reaches the head of its queue, and the traffic shaper on the egress interface selects this queue for transmission.

In SDFR, we can do the same data plane process. Tab. I shows which parts of this process can be realized with OpenFlow. Frames can be matched to flow rules based on matching criteria, and the selection of egress queue and egress port are well-known OpenFlow actions. With the exception of 802.1CB-specific headers, OpenFlow supports all possible protocol fields that can be selected in 802.1Qcc data frame specifications, so there is no need to require stream transformation on talkers and listeners. For ingress policing, SDFR uses OpenFlow metering. Everything happening after the selection of egress interface and egress queue, i.e., traffic shaping and transmission selection, is out-of-scope of OpenFlow. What is missing is the possiblity to adequately configure traffic shaping on the target egress queue. For now, shaper configuration can be deployed via an additional southbound protocol with the only downside being the need to use two different southbound protocols. For the long term, we suggest integrating traffic shaper configuration into the OpenFlow specification.

IV. Time-Triggered Traffic in SDFR

To show that the SDFR architecture can indeed solve TSN configuration, we have implemented a prototype. This section will introduce time-triggered traffic as our use case, discuss changes to the architecture presented before, and present the used hardware.

A. Use Case: Time-Triggered Traffic

Time-triggered traffic is traffic which is sent from talkers periodically in a fixed interval. Commonly used for motion control, the goal is to reduce jitter to a minimum so that frames arrive in nearly the same fixed intervals as they have been sent. Low latency is usually given as a strong upper-bound requirement, but not as an optimization criterion. With the introduction of the *time-aware shaper*, TSN is able to achieve this kind of deterministic latency. The time-aware shaper limits the number of open queues at a given point in a periodical time cycle, which allow frames from high-priority queues to be transmitted at exactly selected points in time [5].

In SDFR, this means that the scheduler is required to schedule time slots for the transmission of frames of a given time-sensitive flow on each switch. The schedule is translated into a configuration in which frames are assigned to FIFO egress queues which are managed by a time-aware shaper. In order for this to work properly, all devices on the frame's path require tick synchronization.

There has been done much work on approaches to calculate such schedules, which is an NP-hard problem. For our proof-of-concept implementation, we have used a rather primitive approach: Our scheduler calculates a shortest path via the Dijkstra algorithm for each new time-sensitive flow, costs equal to the number of time-sensitive flows routed over a given link at the time of adding. Then, it selects the earliest available and feasible time slot in a cycle for the transmission of a flow's frame on each switch. Our scheduler does not support different transmission intervals for multiple streams, and we do not yet have multicast support. However, we kept the scheduler exchangable so we can easily substitute it for a better-working alternative – we consider the scheduling problem as a different problem from the configuration architecture, so it is out-of-scope of this work. With equal transmission periods on all streams and sufficient overprovisioning of network resources, our scheduler can be used in our evaluation.

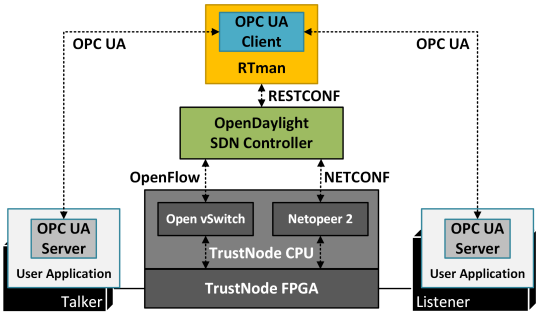

Figure 3. Implementation architecture

B. Testbed

In our testbed, we use two *Hilscher netPI* systems as talker and listener and the *InnoRoute TrustNode* [6] as an SDN/TSN capable switch. The SDN controller, the CNC module, and the UNI app are run on a regular PC.

The TrustNode switch supports the gPTP protocol for time synchronization as well as the TSN time-aware shaper. It has up to eight configurable queues on up to eight Ethernet ports, which are subject to a time-aware shaper. The TSN implementation is done in an FPGA design, which is managed by an Intel Atom board[1].

C. SDN Controller and CNC module

We chose the *OpenDaylight* SDN controller (ver. 0.8.4) as a fitting solution for our needs, because it is open source and the dominant vendor-independent controller framework [7]. Additionally, it comes with features like OpenFlow 1.3, NETCONF and RESTCONF interfaces that are needed by our implementation. For best-effort traffic, we use OpenDaylight's layer-2 switch app.

The implementation architecture is depicted in fig. 3. Although the CNC module should be located on the SDN

[1]CPU: 1.9GHz Quad-Core; RAM: 4GB. OS: OpenWRT 18.06; Switch Software: Open vSwitch 2.10.0 with modifications for FPGA support; NETCONF server: Sysrepo data store 0.7.4 and Netopeer 2 0.5.26; Additional Sysrepo yang models for time-aware shaper configuration: ieee802-dot1q-types@2018-03-07, ieee802-dot1q-bridge@2018-03-07, and ieee802-dot1q-sched@2017-03-16

Table I
AVAILABILITY OF REQUIRED TSN FEATURES IN OPENFLOW.

TSN feature	Configuration	Data Plane	Comment
Stream Transformation	OF Match and frame modification actions in flow rules	Basic Flow Table Processing	Optional, since OpenFlow supports direct matching of any of the DataFrameSpecification tuples in 802.1Qcc.
Queue Management	set-queue action in flow rules	Basic Flow Table Processing	OpenFlow cannot add or remove queues, just discover and use available queues of the switch.
Traffic Shaping	**unavailable in OpenFlow**	Use queue management.	Traffic shaping is invisible for OpenFlow, but may be set up for each queue of the switch
Ingress Policing	OpenFlow Metering	Basic Flow Table Processing	OpenFlow Metering has a low accuracy for small amounts of traffic per flow
Time Synchronization	**unavailable in OpenFlow**	unavailable in OpenFlow	Needs to be turned on, then it can operate autonomously and provide the required configuration.

controller, we decided to implement it as an SDN app in our proof-of-concept implementation. This solution works as well, the only downside is that the common UNI is not a part of the controller's northbound interface. We plan to integrate the CNC module into the SDN controller in the future.

The CNC module is a standalone Python application called *RTman* which connects to OpenDaylight's RESTCONF interface, i.e., a RESTful API providing direct control over switches' OpenFlow flows and NETCONF attributes and management of the controller's NETCONF sessions.

D. common UNI and UNI apps

With the placement of our proof-of-concept CNC module in the SDN application layer, the common UNI is not part of the controller's northbound interface. Therefore, we designed a custom API for this purpose. Since RTman is a standalone application, we decided to implement UNI apps as modules that can be inserted into the process during set-up. The common UNI itself is realized as an object-oriented interface.

As our UNI app, we implemented an OPC UA-based UNI which acts as an OPC UA client which retrieves Talker and Listener groups from OPC UA servers located on all end devices, and stores the values of the associated Status groups.

E. Southbound API: OpenFlow and NETCONF

We create one flow rule for each time-sensitive flow on each switch on that flow's route. Matching rules are created based on the time-sensitive flow's data frame specification, and instructions select an egress queue on an egress interface.

As shown in tab. I and mentioned in sec. III-D, we are not able to configure time synchronization and traffic shaping with OpenFlow. While time synchronization can be statically configured, traffic shaping for time-triggered traffic needs to be reconfigured dynamically during runtime in our use-case. As suggested, traffic shaper configuration can be done via a different southbound protocol to circumvent this shortcoming of OpenFlow. For this, we implemented a NETCONF application for the TrustNode platform, which can be accessed via the *ieee802-dot1q-sched* YANG module to read and write time-aware shaper configuration from or to the TrustNode's FPGA.

F. OPC UA UNI

A promising solution candidate for the configuration interface between end systems and CUC is OPC UA [8], [9]. OPC UA allows to build an information model, which can then be accesses via a standardized network protocol. This satisfies the requirements for an implementation according to 802.1Qcc, and it is a technology of industrial usage. We use OPC UA to realize a CUC UNI app in the SDFR architecture.

The UNI app is connected to the data plane network. It contains an OPC UA client which connects to end devices through the data plane. In our implementation, we used the Python package *freeopcua* (ver. 0.90.6). End devices contain OPC UA servers based on *open62541* (ver. 0.3)

We implemented an OPC UA information model which defines Talker groups, Listener groups, and Status groups on the end devices. User applications can put any number of Talker and Listener groups into the OPC UA server. The UNI app's OPC UA client can fetch Talker and Listener groups from the end devices regularly and hand them over to the common UNI. A writable associated Status group is available on OPC UA servers for every Talker or Listener group, so that a Status group from the common UNI can be transmitted from the UNI app by modifying the values in the associated Status group on the target OPC UA server. User applications can react to changes of Status group values.

V. EVALUATION

A. Topology

Fig. 4 shows our topology. The PC with RTman and OpenDaylight has connections to the TrustNode's Southbound Interface (TrustNode USB port, accessed using a USB-to-Ethernet Adapter), an unmanaged switch and a SIEMENS Bus Analyzer. We connected two Raspberry Pis to the TrustNode as dummy devices. The need for this is described in sec. V-B. A Network TAP is located between TrustNode and the Dummy Listener to mirror the traffic one-way to the Dummy Listener for the Bus Analyzer. The netPis are connected as Talker and Listener to the unmanaged switch.

SSH connections to the two dummy devices, a USB-based serial connection to the TrustNode, as well as details of the miscellaneous network are not explicitely shown in fig. 4.

B. TrustNode Issues

The TAS implementation of the TrustNode, which is developed by InnoRoute and independent of the architecture presented in this paper, is in an experimental stage. At the

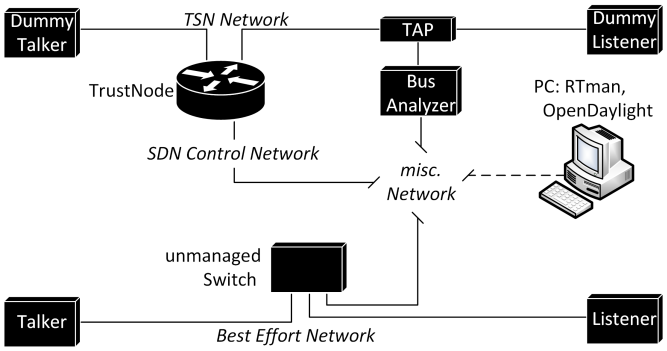

Figure 4. Experiment topology

time of writing, there were issues related to the data plane traffic. It was not possible to forward OPC UA traffic over the TrustNode because this kind of traffic caused the FPGA to crash frequently. Because of this, we decided to split the network into two parts, one for the exchange of UNI messages via an unmanaged switch (best effort network), and one for the examination of the TSN configuration and the resulting scheduled traffic (TSN network). In the TSN network, we used dummy devices that generated UDP traffic used as time-sensitive flows.

C. Evaluation Goals

We show that our architecture works based on three observations: (1) End devices specify their time-sensitive flows via the OPC UA-based UNI and the CNC module calculates a schedule for this. We use RTman's visualization features which show the reserved streams, their paths through the network topology (TSN network) and their schedules to verify this. (2) The CNC module generates a switch configuration based on this schedule and deploys this configuration, and traffic is forwarded accordingly. This can be verified by examining the switch's configuration by hand (serial connection), and by examining traffic dumps on the link between TrustNode and dummy listener (Bus Analyzer). For this examination, we introduce artificial jitter on the Dummy Talker application and verify that traffic between the TrustNode and the Dummy Listener is transmitted according to the schedule. (3) The end devices receive Status messages over the UNI and start sending their flows. To verify this, we examine the respective traffic between the end devices and use the OPC UA client and GUI "UaExpert 1.5.1" to check the OPC UA servers' Status groups.

D. Evaluation Results

We were able to observe all the effects we have mentioned in our evaluation goals. The schedule was calculated according to the specifications of the end devices which were transmitted over the best effort network and took effect on the TSN network (goal 1). The effect of this schedule (goal 2) was shown by an observable lower jitter after the Dummy Talker traffic was forwarded through the TrustNode. Goal 3 was verified by observing the status of the end devices and the Talker behavior. If the Stream IDs of Talker and Listener were equal, the status of both end devices was set as specified and the Talker started to transmit data.

VI. Related Work

A. Real-Time Ethernet Networks

The current state-of-the-art is a wildly spreaded field of different real-time Ethernet networks such as Ethernet/IP, PROFINET, POWERLINK, SERCOS III or EtherCAT. In order to get a real-time Ethernet network to work, the configuration of the network's controller has to be done manually. This configuration is often created by proprietary engineering tools. This also requires a-priori exact knowledge about all connected devices, their application needs, network configuration, and the network topology. While there are approaches for automatic configuration, these are not yet ready for use in production.

B. Real-time Capable Southbound Interfaces

As we haved discussed, OpenFlow does not provide out-of-the-box support for configuring TSN or other real-time systems in switches. Silva et al. [10] implemented an approach using OpenFlow experimenter messages to store information about assigned real-time streams on the switches, thus allowing the use of a custom dispatcher to transmit the stream, once matched in the regular flow table. This approach does not consider TSN and is thus not usable for our work.

C. SDN-based Real-time Stream Reservation

In 2016, Du and Herlich [11] investigated the combination of SDN and Real-Time (RT) Ethernet regarding dynamic configuration. They stated that, apart from the single point of failure issue, an SDN approach is advantageous for dynamic (re)configuration of RT networks and further research may focus on the integration of OpenFlow control functionality in RT Ethernet devices leaving this as an open issue. Kumar et al. [12] came to a similar conclusion in 2017, and they stated that future RT Ethernet systems can also benefit from higher scalability, lower costs, etc. that come with commercial off-the-shelf SDN components. Nayak et al. [13], [14] provided related research outcomes in 2016 and 2018, when they described time-sensitive SDN approaches. For this, they utilized SDN for a now possible centralized configuration of the time-triggered flows. In 2017, Schneider et al. [15] investigated the benefits of SDN for distributed industrial automation systems regarding deterministic communication and stated that a future evaluation of SDN and TSN is profitable for understanding possible synergies. They left the investigation of these synergies as an open issue. Also in 2017, Guttiérrez et al. [8] introduced a framework with learning capabilities for self-configuration of RT networks. For this, they favored the IEEE 802.1Qcc fully centralized model which resembles the SDN architecture and described the usage of OPC UA PubSub requests for the configuration of deterministic paths. In 2018, Ehrlich et al. [16] provided an overview on how SDN resembles the IEEE 802.1Qcc fully centralized model and missing links between SDN and TSN as a solution for

the configuration of RT networks. Schriegel et al. [17] have gotten more concrete regarding heterogeneous time-sensitive networks and described the utilization of a distributed control plane. Also in 2018, Kobzan et al. [18] provided a proposal how time-sensitive communication can become possible for remote monitoring and process control utilizing TSN together with an SDN architecture concept. They stated that an open issue is the configuration of the RT traffic leaving this to future research.

Most of the recent publications (e.g. [15], [16]) recognize the combination of SDN and TSN especially regarding the configuration of time-sensitive networks, describing this as an open issue for research. Related publications (e.g. [8], [17], [18]) provide partial solutions for this, but none is providing a convincing holistic approach plus implementation which is conform to IEEE 802.1Qcc. Implementations for the configuration of time-sensitive networks consisting of end systems and infrastructure devices are either not described or in an early stage.

VII. Conclusion

With SDFR, we have proposed an architecture that can be integrated with existing SDN solutions, provided the switches in the network support SDN features that can be configured via a southbound interface protocol.

As a proof-of-concept, we have shown that this architecture can be used for configuring deterministic transmission of time-triggered traffic, thus providing a solution for an open question in the current efforts for time-sensitive networking. However, the architecture is not limited to this and can support any kind of time-sensitive streams.

Not only have we kept a strict separation of concern between different traffic classes – SDFR is independent from actual UNI protocols, even allowing multiple UNI protocols of different configuration models to coexist in the same network and thus achieving the well-known flexibility of other SDN-based architectures. Furthermore, we are independent of actual routing or scheduling approaches, keeping these interchangable and independent from the UNI as well. As a result, SDFR can be a foundation for TSN configuration in a network, on which scheduling and stream registration can be built on.

References

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, March 2017.

[2] Ieee standard for local and metropolitan area networks–bridges and bridged networks – amendment 31: Stream reservation protocol (srp) enhancements and performance improvements. *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pages 1–208, Oct 2018.

[3] Ieee standard for local and metropolitan area networks–bridges and bridged networks. *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pages 1–1832, Dec 2014.

[4] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[5] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 183–192, New York, NY, USA, 2016. ACM.

[6] C. Liß, M. Ulbricht, U. F. Zia, and H. Müller. Architecture of a synchronized low-latency network node targeted to research and education. In *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–7, June 2017.

[7] SDNCentral LLC. The future of network virtualization and sdn controllers. Technical report, SDNCentral LLC., 2016.

[8] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat. Self-configuration of ieee 802.1 tsn networks. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Cyprus, Sep 2017.

[9] T. Kobzan, A. Boschmann, S. Althoff, I. Blöcher, S. Schriegel, J. S. Michels, and J. Jasperneite. Plug&produce durch software-defined networking. In *9. Jahreskolloquium "Kommunikation in der Automation" (KommA)*, Lemgo, Germany, Nov 2018.

[10] L. Silva, P. Gonçalves, R. Marau, P. Pedreiras, and L. Almeida. Extending openflow with flexible time-triggered real-time communication services. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sept 2017.

[11] J. L. Du and M. Herlich. Software-defined networking for real-time ethernet. In *13th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Lisbon, Portugal, Jul 2016.

[12] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba. End-to-end network delay guarantees for real-time systems using sdn. In *IEEE Real-Time Systems Symposium (RTSS)*, Paris, France, Dec 2017.

[13] N. G. Nayak, F. Dürr, and K. Rothermel. Time-sensitive software-defined network (tssdn) for real-time applications. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*, RTNS '16, pages 193–202, New York, NY, USA, 2016. ACM.

[14] N. G. Nayak, F. Dürr, and K. Rothermel. Incremental flow scheduling and routing in time-sensitive software-defined networks. *IEEE Transactions on Industrial Informatics*, 14(5):2066–2075, May 2018.

[15] B. Schneider, A. Zoitl, M. Wenger, and J. O. Blech. Evaluating software-defined networking for deterministic communication in distributed industrial automation systems. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Cyprus, Sep 2017.

[16] M. Ehrlich, D. Krummacker, C. Fischer, R. Guillaume, S. S. Perez Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, H. D. Schotten, and J. Jasperneite. Software- defined networking as an enabler for future industrial network management. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1109–1112, Turin, Italy, Sep. 2018.

[17] S. Schriegel, T. Kobzan, and J. Jasperneite. Investigation on a distributed sdn control plane architecture for heterogeneous time sensitive networks. In *14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Imperia, Italy, Jun 2018.

[18] T. Kobzan, S. Schriegel, S. Althoff, A. Boschmann, J. Otto, and J. Jasperneite. Secure and time-sensitive communication for remote process control and monitoring. In *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, Sep 2018.