Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)

Paul Pop, Michael Lander Raagaard, Marina Gutiérrez, and Wilfried Steiner

Abstract

In this article, we advocate for the use of IEEE 802.1 Time-Sensitive Networking (TSN) as deterministic transport for the network layer of fog computing in industrial automation. We give an overview of the relevant TSN protocol services and motivate the use of TSN. We propose a configuration agent architecture based on IEEE 802.1Qcc and OPC Unified Architecture (OPC UA), capable of performing runtime network configuration. We briefly present the configuration challenges for scheduled networks (considering a subset of TSN mechanisms), and illustrate one problem: the configuration of schedule tables of such networks for hard real-time control applications. We propose a list scheduling-based heuristic to solve this problem. Our evaluation and comparison to previous work demonstrate the feasibility of reconfiguring the scheduled network at runtime for industrial applications within the fog.

Introduction

We are at the beginning of a new industrial revolution, i.e., Industry 4.0, which is underpinned by a digital transformation that will affect all industries. Industry 4.0 will bring increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations and new business models. However, Industry 4.0 will only become a reality through the convergence of Operational and Information Technologies (OT & IT), which use different computation and communication technologies. OT consists of cyber-physical systems that monitor and control physical processes that manage, e.g., automated manufacturing, critical infrastructures, smart buildings and smart cities. These application areas are typically safety-critical and real-time, requiring guaranteed extra-functional properties, such as real-time behavior, reliability, availability, industry-specific safety standards, and security. OT uses proprietary solutions imposing severe restrictions on the information flow.

IT such as cloud computing and service oriented architecture (SOA) cannot be applied to the bottom levels, at the edge of the network, where industrial machines are located, and where very stringent extra-functional properties have to be guaranteed [1]. Instead, a new paradigm, called *fog computing*, is envisioned as an architectural means to realize the IT/OT convergence. According to the OpenFog consortium, fog computing is a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things." With fog computing, communication devices such as switches and routers are extended with computational and storage resources to enable a variety of communication and computation options. This is illustrated in Fig. 1. Fog computing will enable a powerful convergence, unification and standardization at the networking, security, data, computing, and control levels. It will lead to improved interoperability, security, more efficient and rich control, and higher manufacturing efficiency and flexibility [2]. The vision is to virtualize control and achieve the same levels of dependability as the ones taken for granted in Operational Technology.[1]

In this article, we advocate for the use of TSN for fog computing in industrial automation, and we present the relevant protocol services that motivate the use of TSN. We propose a configuration agent architecture capable of performing runtime network configuration, and we discuss the configuration challenges. To illustrate a configuration case study, we identify the configuration of schedule tables of TSN for hard real-time control applications as a challenging problem. Then, we propose a scheduling heuristic to solve this problem, and our experimental evaluation demonstrates the feasibility of reconfiguring TSN at runtime for industrial applications within the fog.

Fog Computing for Industry 4.0

The integration of computational and storage resources into communication devices is realized in the *fog node*. In many applications, including industrial automation and robotics, several layers of fog nodes with differing computation, communication and storage capabilities will evolve, from powerful high-end fog nodes to low-end fog nodes with limited resources. Companies have started to bring computing and storage closer to the edge of the network (called edge computing). However, edge computing does not provide the dependability and real-time properties required for demanding industrial applications. Research

[1] See the FORA project, http://fora-etn.eu.

Paul Pop and Michael Lander Raagaard are with the Technical University of Denmark; Marina Gutiérrez and Wilfried Steiner are with TTTech Computertechnik AG.

Digital Object Identifier: 10.1109/MCOMSTD.2018.1700057

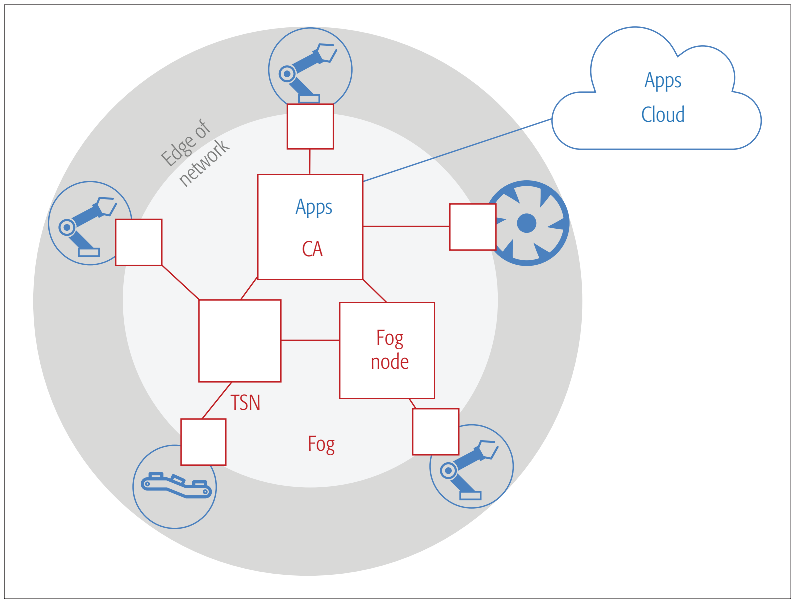


Figure 1. Fog Computing platform. Boxes represent fog nodes, connected with each other and to the Cloud; the thick lines are the network. Applications (Apps) run in the fog and Cloud. Fog nodes may contain a Configuration Agent (CA).

works have started to propose solutions for the implementation of fog nodes [2] including via extensions of the networking layer [3, 4], and fog node solutions have started to be developed by companies. Ongoing work in this area is reported in conferences such as the Fog World Congress, and efforts for standardization are performed within the OpenFog consortium and the Industrial Internet Consortium.

The defining characteristics of a fog node are:

- A fog node is equipped with computational resources that allow the execution of applications.
- The fog node is connected to a larger data processing facility like a cloud environment through a "northbound" connection.
- The fog node is connected to its environment, e.g., machines in the industrial automation use case, through a "southbound" connection.
- A fog node has the ability to configure the communication and computations reachable on its southbound connection.
- The fog node itself must be configurable in terms of communication and computation through its northbound connectivity.

These five characteristics allow the realization of the following examples of generic fog node use cases:

- Computation tasks can be moved from end devices (e.g., drives) to the fog node and to the cloud.
- Updates and patches can be centrally planned in the cloud and automatically rolled out.
- Statistics can be gathered on an end device, fog node, and cloud level as well as any combination thereof.
- Information from the lowest levels, e.g., sensor reads, are seamlessly accessible everywhere in the system on demand, and ideally without a need for protocol gateways.

IEEE 802.1 Time-Sensitive Networking

From fog node characteristics and use cases it becomes clear that the technical capabilities and commercial success of the fog node depends on the technical characteristics of the overall infrastructure that embeds the fog node. In particular, the choice of technology for the southbound connection is crucial in the industrial automation area. Today, the industry uses mostly proprietary protocols [5] that lock customers into the product portfolio of individual product vendors, impairing interoperability.

The initial goal of the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group [6] was to provide timing guarantees for demanding applications such as those in the automotive area. This will support the development of an ecosystem consisting of component and machine builders, system integrators, and hardware and software vendors. Thus, IEEE 802.1 TSN is the ideal technology choice for the fog node's southbound connection, and is most effective by integrating a TSN switch (i.e., IEEE 802.1 Bridge). The vision with TSN is to provide a superior technical solution based on open standards.

TSN guarantees bounded latency communication between the fog node and its environment. This guarantee enables the re-location of real-time critical tasks from the machine to the fog node. Sensor data can be made accessible anywhere in the system by the appropriate configuration of the switch's forwarding table without the need for data or protocol conversion. Furthermore, TSN defines a central network configurator (CNC) that can be operated from a fog node as well. Together with higher-layer protocols like Open Platform Communications Unified Architecture (OPC UA),[2] which currently standardizes OPC UA over TSN, and an OPC UA publish-subscribe protocol suite, TSN not only enables mix-and-match of products, but also plug-and-play capabilities.

There is a strong interest in the industry in adaptive networks that can support safety-critical real-time applications [7]. For example, industrial applications require dynamic reconfiguration to meet new business demands, allowing computation and communication services to evolve over time with minimal disruption. Hence, we focus on solutions that can be used to perform runtime reconfiguration.

TSN and the Integration of IT/OT

The southbound connection of a fog node connects it to its environment, e.g., sensors and actuators, machines, machine components, or to other fog nodes. Thus, the quality of the communication on the southbound interface determines how tightly a fog node integrates in the automation and control processes. While industry, today, deploys various real-time Ethernet variants as a communication means on these low levels, TSN offers an alternative based on open standards of which we review some in this section.

TSN consists to a large extent of amendments to IEEE 802.1Q[3]. It provides the description of basic capabilities like queue-based switching in each outgoing port. A switch that receives a mes-

[2] https://opcfoundation.org

[3] We will not provide references for all standards, but these can be easily found based on their names via IEEE Xplore.

sage will decide on which of the physical output ports the message is forwarded, as well as into which queue on each output port the message is added. Further functions in the IEEE 802.1Q standard include prioritization of messages and the definition of virtual LANs. It also incorporates features that allow bounded latency communication (IEEE 802.1Qav, Credit-Based Shaper) and bandwidth reservation (IEEE 802.1Qat) for Audio/Video Bridging (AVB).

More recently also a synchronous shaping mechanism has been added to the standard, IEEE 801.Qbv, the Enhancements for Scheduled Traffic. IEEE 801.Qbv implements a time-triggered communication paradigm: message transmissions are planned at design time of the system (or through explicit reconfiguration actions). This plan implicitly defines transmission and forwarding points in time for the messages, and each end system and switch in the system locally stores portions of this communication plan (schedule). By reference to this plan, each end system and switch knows when to enable the transmission selection from which queues on the output ports.

A prerequisite for time-triggered communication is the presence of a network-wide reference time, such as the IEEE 802.1AS synchronization protocol that allows local clocks in the end stations and switches to synchronize to each other. IEEE 802.1AS is a profile of the IEEE 1588 standard and is currently under revision. The updated IEEE 802.1AS-rev will provide further synchronization mechanisms, such as improved fault-tolerance by means of hot-standby configurations. The synchronized time in the network enables, e.g., time-triggered communication, coordinated scheduling of tasks and messages, and the implementation of fault-tolerance services.

Furthermore, with TSN critical control tasks can now be migrated to the fog node as the real-time communication mechanisms (IEEE 802.1Qbv, IEEE 802.1Qav, IEEE 802.1Qch) of TSN guarantee the timely response. Even more, such real-time communication mechanisms are currently under standardization, e.g., IEEE 802.1Qcr (asynchronous traffic shaping). Frame preemption (IEEE 802.1Qbu) benefits the integration of real-time and non real-time traffic: critical frames can be configured to interrupt non-critical frames. The transmission of the non-critical frames is resumed once the critical transmission is completed. TSN also standardizes configuration options in IEEE 802.1Qcc, as discussed later.

Configuration Challenges and Related Work

Industrial applications are typically safety-critical and real-time. There has been a lot of work in the area of analysis and optimization of real-time systems. In the context of "Deterministic Ethernet," researchers have addressed the topology design problem, the introduction of new traffic types and the assignment of traffic types to messages; they have proposed solutions to typical communication synthesis problems, such as routing, scheduling, frame packing and fragmenting. A common constraint that needs to be satisfied is the schedulability of messages, and researchers have worked on simulation and timing analysis. The communication synthesis problems have also been addressed in conjunction with task-level scheduling. For a brief survey of the typical configuration problems related to Deterministic Ethernet systems, including TSN, the reader is directed to [8].

In this article we showcase the configuration capabilities of TSN using the problem of scheduling time-sensitive traffic, which is critical for providing guarantees for industrial applications requiring lowest latency. Researchers have shown how to derive at *design time* the schedule tables for both tasks and messages such that deadlines are satisfied [9], and how to incrementally add time-triggered flows at *runtime* in a Time-sensitive Software-defined Network [10].

We have proposed an Integer Linear Programming (ILP)-based formulation for the design-time scheduling problem for TSN [8]. In this article, we propose a configuration agent architecture that uses the capabilities of TSN and OPC UA. We also present an approach to the reconfiguration of schedules at runtime, and we compare the results with the related work on design-time configuration of TSN.

Configuration Agent Architecture

In Time-Sensitive Networking (TSN), the runtime reconfiguration is supported by the extension IEEE 802.1Qcc. It defines a user network interface (UNI), which enables the user to specify stream requirements without knowledge of the network, thereby making the network configuration transparent to the user.

This is achieved via one of three configuration architectures:

- *Fully distributed model*, where stream requirements propagate through the network. The UNI is between an end station and its access switch.
- *Centralized network/distributed user model*, which introduces an entity, called the centralized network configurator (CNC), with complete knowledge of all streams in the network, and all configuration messages originate in the CNC. The UNI is still between the end station and access switch, but in this architecture the access switch communicates directly with the CNC.
- Finally, the *fully centralized model* allows a central user configurator (CUC) entity to retrieve end station capabilities and configure TSN features in end stations. Here, the UNI is between the CUC and the CNC.

Highly critical applications in industrial automation require guaranteed end-to-end delay and minimal delay variation. In other words, predictable, deterministic communication. To this end, we choose a centralized configuration architecture with global knowledge of the network to ensure that all deployed configurations meet the dependability requirements of critical applications. Deriving the schedules is computationally complex and should be centralized into one entity (end system or switch) with complete knowledge of all the streams in the network, hence, this architecture is insufficient for scheduled traffic.

An entity, the configuration agent (CA), is inserted in the network to perform the configuration

In TSN the runtime reconfiguration is supported by the extension IEEE 802.1Qcc, which enables the user to specify stream requirements without knowledge of the network, thereby making the network configuration transparent to the user. To support highly-critical applications, we consider a fully centralized architecture that allows a central user configurator (CUC) entity to retrieve end station capabilities and configure TSN features in end stations.

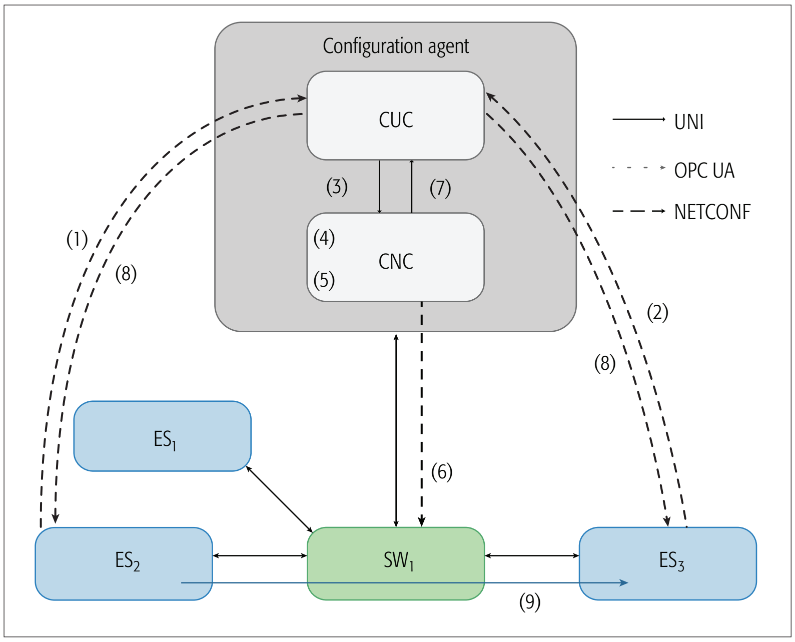


FIGURE 2. Configuration agent consisting of CUC and CNC.

and reconfiguration of the network at runtime. See Fig. 2, where CA is added to a network consisting of three end systems (ES) interconnected by a switch (SW). We consider that both the CNC and the CUC are consolidated into the CA as shown in Fig. 2. End systems send stream requests to the CUC in which the requests are transformed into traffic parameters such as sender, receiver, data size, period, and deadline. The traffic parameters are communicated to the CNC via the UNI. There are several ways to implement the CA (see [11] for a discussion of the solution space):

- Figure 2 shows the conceptual architecture of the CA, which can be implemented both monolithically, in a single entity, or using several entities (end systems and switches) for the CUC and CNC. The fog nodes can be used for the implementation of the CA.
- For the exchange of *network configuration information*, IEEE 802.1Qcc mentions that a variety of protocols can be used, e.g., Simple Network Management Protocol (SNMP), NETCONF (RFC 6241) and RESTCONF (RFC 8040) and suggests YANG (IETF RFC 7950) or Type-Length-Value (TLV) for modeling the data. In this article, we propose the use of NETCONF and YANG.
- Regarding the *discovery* of the time-sensitive streams that have to be considered for rescheduling, IEEE 802.1Qcc mentions that "The protocols that the CUC uses [...] are specific to the user application, outside the scope of this standard." We propose the use of OPC UA over TSN publish/subscribe requests to communicate this information between the CUC and the end systems.

The CNC has knowledge of the network topology, the capabilities of the individual devices in the network, and transmission rates of the links. It runs a scheduling application that maps stream requests to physical routes in the network, assigns streams to output port queues, and schedules the transmission/forwarding time of individual frames of a stream, on every link on the route from sender to receiver. If the stream requests are successfully scheduled, the updated configuration is distributed to the network devices. Once the configuration is updated, the CUC is informed of the stream IDs through the UNI, in order for the end systems to start transmitting via the scheduled streams (an example is discussed in the following section).

OPC UA

OPC Unified Architecture (OPC UA) is a communication protocol for industrial automation developed and maintained by the OPC Foundation. It was originally based on a client/server mechanism in which the client requests information and receives a response from a server. The Unified Architecture Working Group (WG) of the OPC Foundation is working on extending OPC UA with a publish-subscribe model (PubSub[4]) which enables multicast communication, where network devices subscribe to data produced by publishers.

CA introduces two types of traffic in the network:

- Notification messages when new streams are requested between end systems or existing streams are no longer needed.
- Configuration messages, i.e., transportation of configurations to relevant switches.

Notification messages are input to CA and configuration messages are output. The configuration messages are discussed in the next section.

With OPC UA PubSub the end systems do not directly exchange requests and responses, but interact via a *Message Oriented Middleware* that connects the end systems to the CUC. In particular, we propose that the discovery *notification* messages are implemented on top of OPC UA PubSub requests/responses to the message oriented middleware. Thus, the CUC is the central entity that accepts and responds to notification messages with configuration data for the end systems, while the actual data communication between a publishing end system and a subscribing end system is executed directly over the TSN network without a need to involve the OPC UA stack. The implementation details are outside the scope of this article.

NETCONF

To reconfigure each device, TSN has the notion of *managed objects*. Managed objects can be configured to achieve different features of the standards, e.g., managed objects enable setting and receiving the transmission schedules in individual output ports of the network devices. We propose to implement the managed objects using YANG, a data modeling language designed to be used with the Network Configuration Protocol (NETCONF). In a NETCONF architecture there are servers and clients. The CNC component of the CA, with global knowledge of the network, distributes new configurations to all the NETCONF servers sitting in the network switches. We assume that the reconfiguration is realized with NETCONF.

Reconfiguration Case Study

In this case study we consider the configuration of the synchronized transmission schedule as defined in IEEE 802.1Qbv, where time-triggered traffic is implemented by configuring gate-control lists (GCLs) inside network switches. For each

[4] OPC UA Part 14, PubSub Specification, Version 1.04, OPC Foundation.

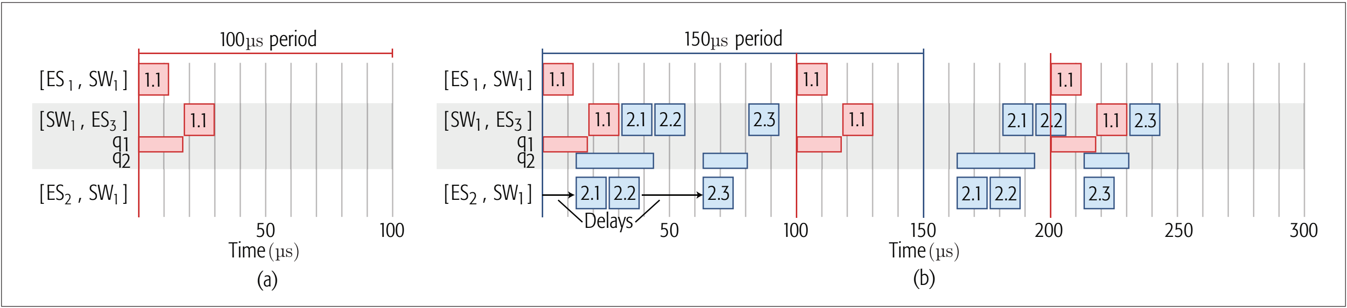


FIGURE 3. Schedule reconfiguration example illustrated as a Gantt chart. A box "i.j" is the *j*th frame of *i*th stream transmitted on the respective link. The thin rows next to the queues illustrate when the frames are in the respective queues: a) initial schedule containing stream 1; b) reconfigured to incorporate stream 2.

outgoing port a GCL specifies which queue is allowed to transmit at precise points in time. In this way, frames are forwarded in the network in a time-triggered manner. For simplicity, in this article, we refer to GCLs as a *schedule*. The schedule is distributed in the output ports of the network devices and must be guaranteed to provide deterministic paths in the network for critical applications. Figure 3a shows such a schedule for transmitting a stream (stream 1) from ES_1 to ES_3 via the switch in Fig. 2.

A window is periodically reserved every 100 µs in the output port of ES_1 for transmitting the (only) frame of stream 1 to the SW_1. Similarly, a window is reserved in the first output port queue (q_1) of SW_1 for forwarding the frame to ES_3 (its destination). The two transmission windows are scheduled such that the frame is forwarded out of SW_1 just after it has arrived in the queue, in order to minimize end-to-end latency. The dedicated transmission windows prevent best-effort traffic from introducing nondeterministic queuing delays.

Suppose that ES_2 is a sensor node that has just been connected to the network in Fig. 2. We assume that all links have transmission rates of 1 Gbps. A distributed control application requires the sensor data to be transmitted to a processing node (ES_3) every 150 µs. The sensor data has a size of 4.5 kilobytes. The steps to establish a stream from ES_2 to ES_3 are illustrated in Fig. 2 and are as follows:

1 ES_2 sends an OPC UA publish request to CUC.
2 ES_3 sends an OPC UA subscribe request to CUC.
3 CUC communicates to the CNC via the UNI that a time-sensitive stream with period 150 µs and data size 4.5 kilobytes is needed from ES_2 to ES_3.
4 The message data is placed into three maximum sized Ethernet frames of 1542 bytes each (including overhead). Considering the transmission rate of the links, we have a transmission duration of 12.336 µs for each of the three frames.
5 The scheduling application routes the stream through the only available route, and extends the schedule with three new frames such that they do not interfere with the existing frames (Fig. 3b).
6 From the updated schedule, a new configuration is derived for SW_1, which is communicated via NETCONF.
7 Once the reconfiguration is completed, CNC notifies CUC that the stream was successfully scheduled with stream ID 2, and of the transmission schedule for ES_2.
8 Via OPC UA, CUC passes this information on to the publisher ES_2 and subscriber ES_3.
9 The distributed application is executed, i.e., ES_2 starts to transmit the sensor data periodically according to the schedule. In this way, ES_3 receives sensor data every 150 µs with guaranteed latency and minimal jitter.

SCHEDULING HEURISTIC

We present a scheduling heuristic for reconfiguring the transmission schedule at runtime (see [12] for details). In order to minimize the impact of the reconfiguration, the heuristic incrementally adds new streams to the existing schedule. If the incremental approach is unsuccessful, the entire schedule is rebuilt using the heuristic. If this also fails, the scheduler resorts to design-time approaches such as [3, 4]. While these design-time approaches are searching for a feasible schedule, the network operates with the current configuration.

We consider that the CUC periodically communicates changes in stream requirements to the CNC. When the scheduler is executed in the CNC, some streams have disappeared since the previous reconfiguration and are to be removed from the current configuration, whereas other streams have appeared and wait to be scheduled. Figure 4 illustrates the flow of the scheduling heuristic.

An important step in the algorithm is prioritizing the appeared streams. The priority determines the order in which streams are incrementally added to the schedule. As the number of streams grows in the network, it becomes increasingly difficult for the heuristic to schedule a specific stream. Hence, important streams should be scheduled first. For instance, critical streams and streams with early deadlines should have high priority to motivate that they are scheduled in time.

The period of a stream also affects when it should be scheduled. To minimize jitter, streams are sent at the same period offset in every repetition. In addition, to account for all the possible scenarios where scheduled streams may interfere with each other, the width of the schedule must equal the *least common multiple* of all stream periods. We refer to this as the *hyperperiod*. This is the reason why the width of the schedule in Fig. 3 is increased from 100 µs to 300 µs when a 150 µs-period stream is added. The frames of a short-period stream are repeated more frequently

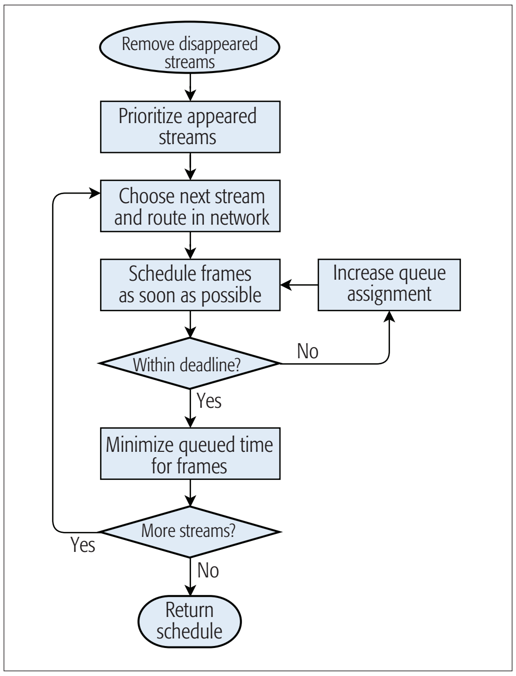

Remove disappeared streams
Prioritize appeared streams
Choose next stream and route in network
Schedule frames as soon as possible
Increase queue assignment
Within deadline?
No
Yes
Minimize queued time for frames
More streams?
Yes
No
Return schedule

Figure 4. Flowchart of scheduling heuristic. If a stream cannot be scheduled, the heuristic attempts to reschedule all streams.

in the entire schedule, and hence its schedule is in general more restricted so it would benefit from being scheduled early.

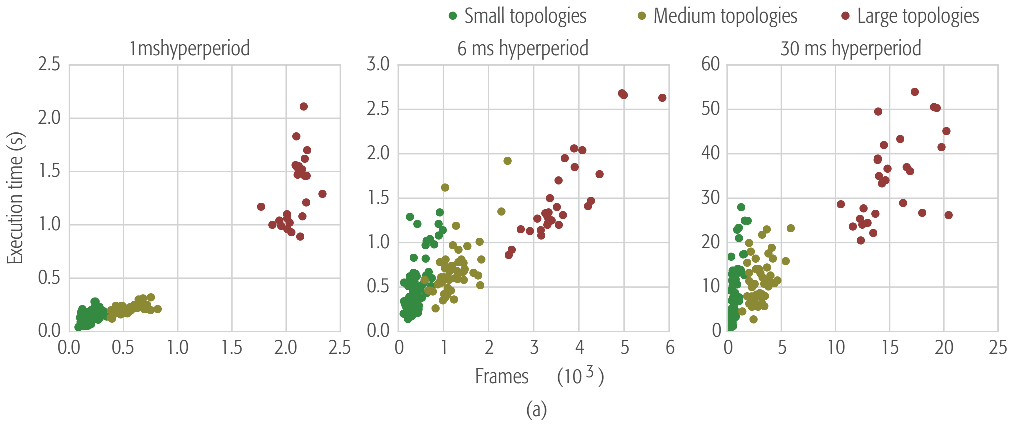
Streams are scheduled individually sorted by priority. Initially, streams are assigned to the first output port queue in each hop to minimize the number of queues dedicated to scheduled traffic. In an attempt to meet the deadline of a stream, its frames are scheduled in an as-soon-as-possible fashion, i.e., each frame is scheduled at the earliest point in time such that the schedule remains feasible. In case the frames fail to meet the deadline, the current queue assignment is too restrictive and must be reevaluated. This is done by incrementing the queue assignment in the first hop which allows a frame to start earlier than with the current assignment.

To prevent scheduled streams from interfering with each other, two streams cannot wait in the same queue simultaneously. As a result, it is not possible to schedule stream 2 such that it shares queue q_1 with stream 1. Instead, it occupies its own queue q_2, as shown in Fig. 3b. Due to this restriction, a post-processing step is introduced after a stream has been scheduled. It reduces the time duration that frames are queued in switches by delaying the transmission on the incoming link as much as possible. The introduced delays are marked in Fig. 3b. In this way, it is more likely that future streams will be able to share existing scheduled queues, thereby leaving more queues available for best-effort traffic.

Experimental Evaluation

In the worst case, the reconfiguration scheduler has to destroy the current schedule and reschedule all streams. Hence, we experimentally evaluate the worst-case execution time of the scheduler on a set of synthetic benchmarks by letting it schedule all streams. The scheduler is evaluated on 440 synthetic test cases on three different network sizes: small (4–7 devices, e.g., 4 ES and 3 SW), medium (50–76 devices, e.g., 48 ES and 28 SW) and large (402 devices, e.g., 288 ES and 114 SW), using a star topology. The topologies are based on industrial requirements and are derived from [13]. The test cases have three different hyperperiods, 1 ms, 6 ms, and 30 ms, with high link utilizations to provoke multi-queue scenarios. The average link utilization is 41 percent, 13 percent, and 8 percent, for the small, medium, and large topologies, respectively. The small test cases have an average of 15 streams fragmented into 430 frames. Medium test cases have 55 streams and 1500 frames on average and large test cases have an average of 290 streams and 7300 frames.

Figure 5a shows the execution times of the test cases for different hyperperiods and topology sizes. On average, 1300 frames are scheduled per second across all test cases. All test cases with hyperperiod 1 ms and 6 ms are schedulable within 3 seconds, whereas large topologies with hyperperiod 30 ms may take up to 1 minute. The execution times assume a high-end fog node where the scheduler runs on a 2.5 GHz Intel Core i5 processor. The distribution of scheduled queues across all output ports is shown in Fig. 5b. It shows that the queue minimization strategy of the heuristic schedules more than 95 percent of output ports using only a single queue for scheduled traffic. Only a small fraction of output ports (0.5 percent) require more than two scheduled queues. Once the schedule is determined, and the fog nodes are ready to



Scheduled queues	Percentage
1	95.2
2	4.3
3+	0.5

(b)

Figure 5. Experimental evaluation of scheduling heuristic: a) execution time as a function of the frames for different hyperperiods and topologies; b) scheduled queues in output ports.

receive it (according to NETCONF), the reconfiguration takes a few seconds or less, considering current TSN network switch prototypes.

We were also interested to compare the proposed heuristic to the related work. Thus, we have implemented the OMT approach from [13] and the ILP approach from [8]. Both Optimization Modulo Theories (OMT) and Integer Linear Programming (ILP) are mathematical formulations that rely on solvers to find the optimal solution, in this case the optimal number of scheduled queues (K). ILP formulates the problem as a linear program where all variables take integer values. OMT augments Satisfiability Modulo Theories (SMT) solvers with optimization capabilities. SMT determine the satisfiability of first-order logical formulas. The values of K are presented in the table, including the lower $\underline{K}$ and upper bound $\overline{K}$. The lower bound is assuming a single scheduled queue in all output ports forwarding scheduled traffic, and the upper bound assumes the minimum of the available queues in the output port and the number of scheduled streams forwarded through that output port. We have compared the three approaches on the test cases from [8], and our scheduler has been able to obtain the same optimal solutions in a fraction of a second for all test cases, as shown in Table 1. However, note that our heuristic is not guaranteed to find the optimal solution in all cases.

Conclusions

We have presented the main features of a fog node, and argued that IEEE 802.1 TSN is the right solution for the networking layer of fog computing in industrial automation. However, TSN presents several configuration challenges, and configuration is also desirable at runtime. Hence, we have proposed a configuration agent architecture and we have used the derivation of schedule tables as a case study to illustrate the runtime configuration challenges. We have proposed a scheduling heuristic for the synthesis of the schedule tables in TSN. We have extensively evaluated our proposed heuristic, and the conclusion is that it is able to handle large runtime reconfiguration problems, leading to good quality solutions.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA–Fog Computing for Robotics and Industrial Automation.

ID	Arch.		Running time (s)			Queue usage		
	ES	SW	ILP	OMT	Heuristic	K	$\underline{K}$	$\overline{K}$
T01	3	1	0.66	0.81	0.02	2	2	5
T04	3	1	2.49	2.46	0.05	2	2	5
T05	3	1	3.73	3.43	0.10	2	2	3
T10	5	2	4.70	5.12	0.06	4	4	8
T11	5	2	16.54	12.94	0.10	3	3	7
T12	5	2	210.03	34.33	0.09	5	5	9
T14	3	1	39.06	22.87	0.12	2	2	3
T18	3	1	10.98	7.17	0.03	2	2	5

Table 1. Comparison of ILP, OMT, and scheduling heuristic.

References

[1] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in Real-Time Virtualization and Predictable Cloud Computing," *J. Systems Architecture*, vol. 60, no. 9, 2014, pp. 726–40.

[2] F. Bonomi *et al.*, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, pp. 169–86.

[3] D. Henneke, L. Wisniewski, and J. Jasperneite, "Analysis of Realizing a Future Industrial Network by Means of Software-Defined Networking (SDN)," *2016 IEEE World Conf. IEEE Factory Commun. Systems (WFCS)*, 2016, pp. 1–4.

[4] J. Farkas, S. Haddock, and P. Saltsidis, "Software Defined Networking Supported by IEEE 802.1q," CoRR, vol. abs/1405.6953, 2014, http://arxiv.org/abs/1405.6953.

[5] P. Gaj, J. Jasperneite, and M. Felser, "Computer Communication within Industrial Distributed Environment — A Survey," *IEEE Trans. Industrial Informatics*, vol. 9, no. 1, 2013, pp. 182–89.

[6] IEEE, "Official Website of the 802.1 Time-Sensitive Networking Task Group," http://www.ieee802.org/1/pages/tsn.html, 2016.

[7] "ARTEMIS EMC2 project, Internet of Things Living Lab, Open Deterministic Networks," https://www.artemis-emc2.eu/project overview/ll5_internet_of_things/.

[8] P. Pop *et al.*, "Design Optimization of Cyber-Physical Distributed Systems using IEEE Timesensitive Networks (TSN)," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, 2016, pp. 86–94.

[9] S. S. Craciunas and R. S. Oliver, "Combined Task- and Network-Level Scheduling for Distributed Time-Triggered Systems," *Real-Time Systems*, vol. 52, no. 2, 2016, pp. 161–200.

[10] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental Flow Scheduling Routing in Time-Sensitive Software-Defined Networks," *IEEE Trans. Industrial Informatics*, vol. PP, no. 99, 2017, pp. 1–1.

[11] M. Gutiérrez *et al.*, "Self-Configuration of IEEE 802.1 TSN Networks," *IEEE Int'l. Conf. Emerging Technologies and Factory Automation*, 2017, pp. 1–8.

[12] M. L. Raagaard and P. Pop, "Optimization Algorithms for the Scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)," Technical University of Denmark, Tech. Rep., Jan. 2017.

[13] S. S. Craciunas *et al.*, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," *Proc. 24th Int'l. Conf. Real-Time Networks and Systems*, 2016, pp. 183–92.

[14] F. Dürr and N. G. Nayak, "No-Wait Packet Scheduling for IEEE Timesensitive Networks (TSN)," *Proc. 24th Int'l. Conf. Real-Time Networks and Systems*, 2016, pp. 203–12.

Biographies

Paul Pop [SM] (paupo@dtu.dk) is a professor at DTU Compute, Technical University of Denmark (DTU), and the director of DTU's IoT Research Center. He received his Ph.D. degree in computer systems from Linköping University in 2003. His main research interests are in the area of system-level design of cyber-physical systems. He has published extensively in this area, and has received the best paper award at the DATE 2005, RTiS 2007, CASES 2009, MECO 2013 and DSD 2016 conferences and the EDAA Outstanding Dissertation Award (co-supervisor) in 2011.

Michael Lander Raagaard (michael@raagaard) received the M.Sc. degree in computer science and engineering from the Technical University of Denmark (2017). His main research interest is combinatorial optimization in embedded systems design. He has applied his research to the areas of safety-critical networks and microfluidic biochip synthesis.

Marina Gutiérrez [M] (marina.gutierrez@tttech-automotive.com) is a project engineer at TTTech Computertechnik AG. She is a voting member of the IEEE 802.1 TSN working group and the editor of the P802.1Qcw, a project that is standardizing YANG models for TSN features. She holds a BS in physics and an MS in computer science from the University of Cantabria, and she is currently enrolled in Mlardalen University as a Ph.D. student. Her research is focused on the configuration and management of deterministic communications for cyber-physical systems.

Wilfried Steiner (wilfried.steiner@tttech.com) is a corporate scientist at TTTech Computertechnik AG and Leader of the research team TTTech Labs. He holds a degree of Doctor of Technical Sciences from the Vienna University of Technology, Austria. His research is focused on the design of systems and network protocols with real-time, dependability, and security requirements. Target areas are automotive, space, aerospace, and more recently the Industrial Internet of Things.