Towards Industrial/Multimedia TSN Network Slice Management: A Bottom-Up Approach

Gilson Miranda Jr.*†, Nina Slamnik-Kriještorac*, Johann M. Marquez-Barja*, Daniel F. Macedo†
*IDLab - imec, University of Antwerp, Belgium
†Universidade Federal de Minas Gerais - Computer Science Department, Brazil
{gilson.miranda, nina.slamnikkrijestorac, johann.marquez-barja}@uantwerpen.be
damacedo@dcc.ufmg.br

***Abstract*—Network slicing enables multiple virtual networks to share physical resources, allowing network operators to deliver highly customizable and efficient networking solutions that meet the diverse requirements of modern applications. The automated management of network slices has been studied in the last years to make such solutions more flexible, ready to support new applications, and capable of optimizing network resource utilization. Many works in the literature give a top-down approach, focusing on the high-level decision processes, and relying on abstracted infrastructure managers and simulation tools to apply/execute such decisions. In this work, we leverage components that we previously developed for network monitoring, flexible traffic shaping, and Software-Defined Time-Sensitive Networking control, to create a bottom-up approach toward automated slice management. We describe the intricate coordination of elements required for an automated control loop and present the results achieved with a proof-of-concept executed in a real testbed of wired and Wi-Fi nodes. The results show the capability of the system to correctly identify the bottleneck of a flow and apply corrective actions to reestablish its intended performance level.**

***Index Terms*—TSN, Network Programmability, SDN**

I. Introduction

Network slicing allows the deployment of independent virtual end-to-end networks running over a shared physical infrastructure. This technology largely relies on virtualization mechanisms to provide resource sharing and isolation. In industries, different applications like process control, video surveillance, alarm propagation, and object tracking often coexist, each with its own set of requirements. There are many challenges to be addressed when managing network slices. First, the life-cycle management of slices and the assurance of their performance requirements must be automated to avoid the constant need for human intervention [1]. With multiple slices sharing the physical infrastructure, the adaptation of resource allocation is necessary to ensure the performance requirements of applications. At the same time, optimized resource multiplexing can improve the number of served slices and increase revenues [2]. Dealing with multiple applications with different characteristics, and keeping their performance within the expected Key Performance Indicators (KPIs) is a challenging task [3].

Deployments with coexisting Operational Technology (OT) and Information Technology (IT) traffic are becoming more likely with the evolution of Time-Sensitive Networking (TSN) over Ethernet [4]. TSN originated from the multimedia domain with the Audio Video Bridging (AVB) Task Group by IEEE 802.1 in 2007 [5]. The scope of TSNs was widened to support more applications, and its standards have been proven promising to achieve effective network resource allocation [6].

While network slicing has been the subject of several studies in the last years, the vast majority of works focus on higher-level aspects of slicing, relying on abstracted infrastructure managers and simulation tools to evaluate the solutions. Building on top of our recent works on data plane programmability using softwarized TSN functions, fine-grained network telemetry, and Software Defined Networking (SDN) control for TSN networks, address the challenge of slicing management for Industrial and Multimedia TSN networks in a bottom-up perspective [6]–[8]. We detail our data modeling for application and slice management with insights from recent 3GPP and ETSI standards for Zero-touch network and Service Management (ZSM) and Intent-Based Networking (IBN), adapted for operation with our TSN Controller architecture.

The paper is organized as follows: Section II presents the related work. Section III describes the extensions added to our TSN Controller. Section IV details the Control Loop operation. Section V describes the setup used for experimentation and Section VI presents and discusses the Results. Section VII concludes this paper.

II. Related Work

In this section, we first discuss the related works addressing network slicing mechanisms that aim to adapt the allocation of resources towards a user- or application-specified set of goals. In this context, works on IBN aim at managing network infrastructure to satisfy *intents*, usually expressed through a high-level language. Several works address the problem of expressing and interpreting intents and their translation into objective goals, metrics, or actions [9].

For industrial or critical communications, Saha et al. [10] and Mehmood et al. [11] present architectures highlighting the interpretation of intents and extraction of key information from keywords. Lower-level actions are attributed to infrastructure managers such as OpenStack. Cerroni et al. [12] propose a reference architecture for the orchestration of heterogeneous SDN domains, with some details on JavaScript Object Notation (JSON) messages to express intent-based requests to a virtual

infrastructure manager. These infrastructure managers abstract details about which techniques can enforce the decisions on the network and the evaluation is done through emulation or simulation. In this work, we start from the low-level enablers for traffic monitoring and shaping on heterogeneous networks, then incrementally add the building blocks for high-level management. Moreover, we perform an experimental evaluation of our system on a wired/Wi-Fi testbed.

Rothenberg et al. [13] and Perepu et al. [14] propose systems for intent-based multimedia management using Quality of Experience (QoE) feedback. The first paper targets a video-streaming application, using estimated QoE from Quality of Service (QoS) metrics emulated in the network. The second paper considers a Conversational Video application for which a QoE metric between $[1, 5]$ must be maximized, and Ultra-Reliable Low-Latency Communication (URLLC) and mobile IoT applications for which the Packet Loss Rate (PLR) between $[0, 1]$ must be minimized. The paper is focused on a Multi-Agent Reinforcement Learning system to control bitrate and priority for the flows and evaluates the performance using a network emulator. Our work uses objective QoS metrics as target KPIs to be optimized, with QoE-based management envisioned as a future step in this bottom-up approach. Therefore, we address the realization of slice management at a lower level, with a Proof-of-Concept (PoC) testbed implementation.

Network slicing over Wi-Fi is also a relevant subject, as a collaboration between Wi-Fi and cellular (5G/6G) networks has been long envisioned [15]. Richart et al. [16] propose a slicing mechanism for Wi-Fi Access Points (APs) combining a queuing and scheduling mechanism. The solution is evaluated analytically and through simulations. Isolani et al. [17] carry out experimental work with an SDN-based approach for on-the-fly end-to-end slice orchestration. The solution is based on the framework from Coronado et al. [15], enhancing it with an algorithm that periodically adjusts the airtime allocation of the slices on a Wi-Fi AP. The system considers two slices: QoS and best-effort; being unclear whether more slices are supported. In this work, we propose data models and algorithms to manage slices on a higher level, and over heterogeneous wired/Wi-Fi networks.

This work builds on our previous experimental works on SDN-based TSN networking and flexible data plane monitoring and configuration. On top of these elements, we detail a data model based on standards under development and a Control Loop workflow for automated network slice management using TSN functions.

III. TSN Controller and New Extensions

Before describing the TSN Controller (TSNC) extensions for this work, we briefly describe the architecture of TSNC and TSN Agent (TSNA) shown in Figure 1, introduced in our previous work [6]. The architecture uses a Controller/Agent model, with the TSNAs deployed at the Network Elements (NEs) and a centralized NE running the TSNC. In the TSNC, the Central Network Controller (CNC) module interacts with the TSNAs to set NE configuration and receive notifications of important events (e.g., topology changes). Configurations such as schedule updates and flow classification rules are sent to the TSNAs by the CNC via ZeroMQ (ZMQ) messages. The TSNA interprets such messages and issues the appropriate command/API calls on the NE to set the desired configuration. The configuration messages from the CNC can be triggered by commands received from a user operating the network via the User/Network Interface, or from other micro-services in the TSNC such as the CUC and Control Loop, via the Internal Interface. A Monitor micro-service runs a database, dashboard, and ZMQ subscriber to receive telemetry from the NEs.

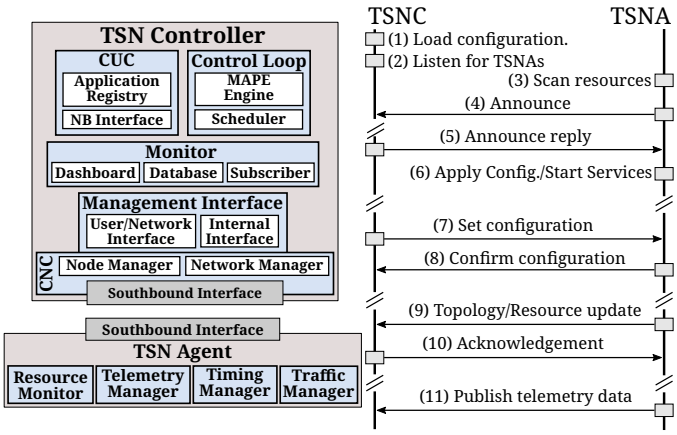

Figure 1: TSNC/TSNA block diagram and communication flow

A. Event System

To properly support the automated creation, management, and termination of network slices, we added new features to the CUC and CNC modules. The first feature is an event system based on ZMQ Publish/Subscribe messaging. The Centralized User Configuration (CUC) and CNC elements of the TSNC open a publishing server socket to which other micro-services like the Control Loop can subscribe to get notifications of relevant events. The events generated by the CNC are *Node Connect*: when a new node (TSNA) connects to the network; *Node Disconnect*: when a node leaves the network; *Topology Update*: when topology updates are detected through the Link Layer Discovery Protocol (LLDP) protocol executed by the TSNAs. The events generated by the CUC are *App Instance Created*: when an application requests an instance and the admission control of the CUC accepts it; *App Instance Removed*: when the application ends or is terminated.

B. CUC Updates

The second improvement is on the CUC module. The CUC is part of a fully centralized TSN network and has no strict specifications, being vendor-specific. The vendor of the TSN devices will supply a CUC [18]. Therefore, we propose a CUC architecture aimed at supporting automated slice management in cooperation with the other components of the overarching architecture. The implementation of our CUC is inspired by

specifications of ZSM and IBN [19], [20]. In this sense, we developed a structure for the specification of performance requirements (in terms of KPIs) and *Attributes* (e.g., packet generation pattern) of applications that will use the network.

Figure 2 describes the elements of an application *Definition*, and their composition to create an application *Instance*. These items are specified in JSON format and loaded by the CUC. An application *Definition* is a generic specification that extends *KPIs* and *Attributes*. An application *Instance* is created when an application is started (or is ready to start). The *Instance* is extended with the *Flow* information describing the network endpoints (addresses, ports, protocols), and the *Slice* information indicating the slice allocation of the flows. With this structure, many instances of applications with similar characteristics and KPIs can exist in the network. The applications can be wrapped to automatically request instances to the CUC, or the instances can be manually created by an operator via the Northbound Interface (NB Interface) of the CUC.

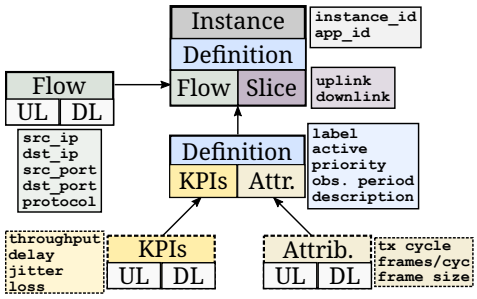

Figure 2: Composition of an application instance

The KPIs are measurable performance metrics of a network flow, equivalent to *intentExpectations* from 3GPP specifications [20]. In this work, we define throughput (Bytes/s), delay (ms), jitter (ms), and PLR (%) as KPIs, which can be measured per-flow/per-hop using an In-band Network Telemetry (INT) framework [7]. For each metric, the KPIs specify a lower and upper bound, for example, $delay : [0, 100]$ defines that the delay must be lower than 100ms; $jitter : [0, 1]$ defines that the jitter must be no higher than 1 ms; $loss : [0, 0]$ defines that no packet loss is tolerated. If a given metric is not relevant for an application, it can be simply set to a wide range, e.g., $delay : [0, 10000]$. Other metrics such as RSSI or RTT can also be used as KPIs, as long as the control loop can interpret them for the traffic analysis. These definitions are specified for Uplink (UL) and Downlink (DL).

Attributes are an optional component of an application definition, specifying the frequency of packet transmission and the size of packets generated by the application. This information is useful for flow scheduling and traffic control. The controller might be able to allocate resources for a flow with small packets, but not for a flow with large packets. Therefore, if this information is known and can be specified for the application, it might facilitate the admission of an application instance.

The UL/DL KPIs and Attributes are combined to form an application *Definition*, equivalent to an *intent* specification from 3GPP [20]. Other fields in a *Definition* are: *description* of the definition; *observation period*: the time in seconds for the verification of the fulfillment of the KPIs; *priority*: the priority that will be given to instances of this definition for admission control and resource allocation; *active*: if this definition can become an instance or not; *label*: a unique identifier of the definition to be used when requesting/creating instances.

Finally, we have the *Instance* specification, which is created on the fly based on requests received by the NB interface of the CUC. An application wrapped to work with this system will request an instance to the CUC before starting transmitting/receiving data. This request informs the Source/Destination IPs, Source/Destination ports, and transport protocols. This information is used to build the *Flow* structure of the instance and generate classification rules to allocate the flows to the appropriate slice. The *Slice* structure specifies to which slice the uplink and downlink flows are allocated, and can be changed in runtime based on decisions of the Control Loop service. Each instance is uniquely identified by an *instance_id*, and a numeric *app_id* (for services that support numeric identification only, such as the INT Framework).

C. Instance Creation and Termination

Figure 3 describes the workflow for creating and terminating an instance. Actions such as setting classification and INT rules are always confirmed by the TSNA to the CNC/CUC but were omitted in the figure. A Definition can be registered in the CUC via the NB interface if a suitable definition for the application is not yet registered in the CUC. Then, an *Instance Request* starts the instance creation process.

First, an admission check is done by the CUC, and the initial slice for the application is defined, depending on the network route, KPIs, and attributes. The CUC makes calls to the CNC to apply flow classification rules in the nodes. In essence, the TSNAs use *iptables* to mark the packets of the flow with a certain Differentiated Services Code Point (DSCP) value, which we later assign to a certain queue for scheduling. The INT rules are applied via API calls to elements of the framework (detailed in reference [7]). Next, there is an optional step of applying a new schedule to the network. This step can be skipped if the current network configuration can accommodate the new flows. We use time-based schedules that follow the format defined by IEEE802.1Qbv [21], also implemented on top of Click Router. The CUC then triggers an event informing other services that a new instance was created and returns the specification and a success or failure code to the requester in the NB Interface. At this moment, the network is ready and the application can start running.

During operation, the TSNAs transmit telemetry reports to the Monitor module of the TSNC. These reports are stored in a database and can be consulted by other micro-services like the Control Loop. During the application life-cycle, the schedule and classification rules can be modified by the Control Loop (e.g., to change the flows to a different slice) through calls to

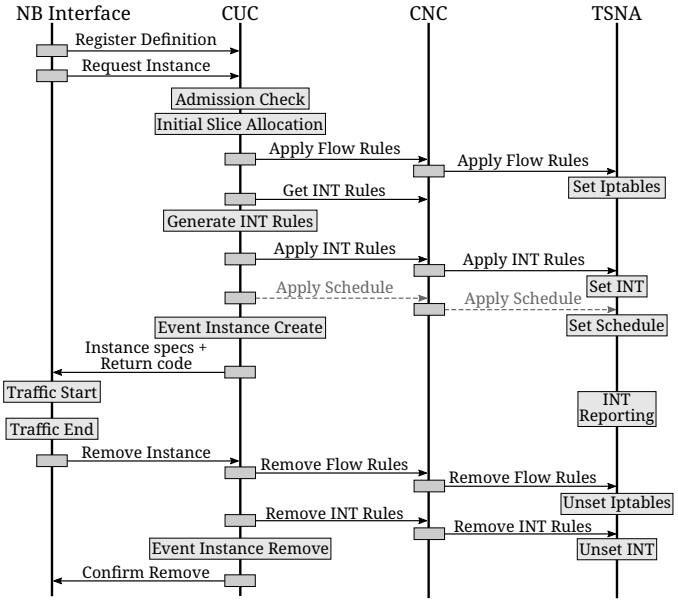

Figure 3: Workflow for application instance creation

the CNC. In case of slice change, it is the responsibility of the Control Loop to make an API call to the CUC to update the *Slice* data of the instance, to keep the information consistent.

When the application terminates, it issues a termination call to the CUC, which triggers calls to remove the classification and monitoring rules. An event is issued announcing the termination of the instance, and a confirmation is sent back to the application. Instances can also be terminated manually through the NB Interface or based on time (e.g., after an amount of minutes/hours).

IV. Control Loop

While an application is running, the network performance of its flows might degrade due to various circumstances, especially when wireless links are part of the end-to-end route. The Control Loop monitors the performance of these applications and adjusts the network slices to guarantee that the requested KPIs of an instance are met. We developed a Control Loop micro-service that automatically reacts to performance degradation events, triggering actions to identify the main bottlenecks and take corrective actions to restore application performance.

Figure 4 gives an overview of the flow of actions and events of the Control Loop when any of the KPIs are not fulfilled. The Control Loop listens to events from the CUC and CNC. Upon detection of an instance creation event, the module requests detailed information about the instance to the CUC and starts a monitoring thread. This thread gets telemetry reports from the database and compares the measurements against the KPIs of the instance definition. If any KPI is measured out of the expected range, a *Fulfillment Report* is generated indicating which *Flow* (UL/DL) and which metric (throughput/delay/jitter/PLR) is not fulfilled, and an internal event to the main thread is triggered.

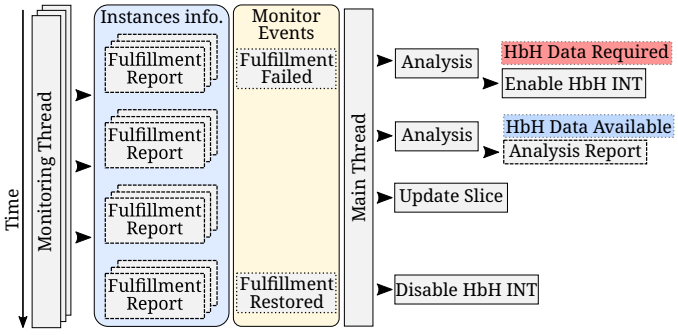

Figure 4: Control Loop flow overview

When the main thread gets a *fulfillment failed* event, it starts the analysis function, which reads the telemetry data to identify which hops are the main contributors to performance degradation. By default, we set the INT Framework to perform only End-to-End (E2E) monitoring to limit data and control plane overhead. Thus, the Hop-by-Hop (HbH) telemetry is usually not immediately available. The analysis function then enables the HbH monitoring for the flows and waits until detailed telemetry becomes available. When HbH telemetry becomes available, the analysis function generates a report identifying the main bottlenecks of the flows. This report has a structure indicating the hop (a tuple of source and destination node IDs), and the telemetry measurements. The hops are sorted according to the "bottleneck magnitude": e.g., if the E2E delay is 20 ms, and the delay on a hop is 15 ms, this will be the first hop of the list in the analysis report. This way, the corrective actions can be planned with a focus on the links that contribute the most to the performance degradation.

After getting a complete analysis report, the main thread schedules a *update slice* function. If more than one analysis for instances with unmet KPIs is currently running, the *update slice* function waits until it generates an analysis report, to perform a network update that encompasses multiple instances. The *update slice* function then acts to address the bottlenecks and reestablish the expected KPIs. When the monitoring thread detects that the KPIs are again fulfilled, it waits $2 \times observation\_period$, then generates a *fulfillment restored* event to the main thread. The main thread disables the HbH monitoring of the flows of the instance. We define this longer observation period to disable the HbH monitoring to ensure that the issue is solved and no further analysis is required.

V. Experimental Setup

To evaluate our PoC implementation we used a testbed with the topology and nodes detailed in Figure 5. The topology has 3 switches based on industrial mini PCs and firewall appliances, one Wi-Fi AP based on Intel NUC, a wired end node (*node0*), a Wi-Fi end node (*node1*) based on Intel NUC, and a Controller. All nodes are time-synchronized using Precision Time Protocol (PTP). To evaluate the operation of this PoC, we use a simple traffic generator sending packets every 1 millisecond from *node0* to *node1*. This traffic generator

is wrapped to request an instance to the CUC and only starts after the request confirmation. We ran tests with the *observation period* of the application definition configured for 1, 2, and 4 seconds. The application KPIs defined that the uplink delay of the application had to be under 10 ms.

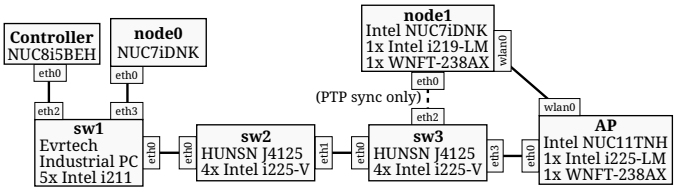

Figure 5: Topology for the experiments

We performed 30 repetitions for a given observation period configuration. After 3 seconds of the start of an experiment repetition, we started another traffic generator on the same slice between a pair of random hops, saturating the slice on one of the interfaces. For example, starting a saturating flow between *sw3* and the *AP*, saturating the slice on interface *eth3* of *sw3*. This event triggers the reconfiguration of the network slices to restore the performance of the monitored flow. For this PoC, we defined a simple action of reconfiguring the class of the monitored flow and migrating it to a different slice with a dedicated slot in the time-based schedule. Our main goal is to validate the complex sequence of events and actions required for the monitoring, analysis, and identification of network bottlenecks to perform effective automated troubleshooting.

VI. Results and Discussion

Figure 6 shows the one-way delay measured by INT between *node0* and *node1*. At the beginning of the test, the delay is under 2 ms. After we start the background traffic, the delay goes over the 50 ms mark. Table I shows the occurrence of Control Loop events after the start of the background traffic. On average, it takes 3 to 5 seconds for the unmet fulfillment to be detected. After that, the process of enabling HbH telemetry, gathering more data, and taking action takes another 3 to 4 seconds on average.

Observation Period (s)	Event time (seconds (Std.Dev.))		
	Detection	Action	Restored
	On-demand HbH Telemetry		
1	3.63 (0.52)	6.73 (0.53)	11.35 (0.94)
2	3.64 (0.80)	7.60 (0.85)	13.71 (0.80)
4	5.04 (1.17)	9.06 (1.17)	17.22 (1.47)
	Always On HbH Telemetry		
1	3.56 (0.46)	3.57 (0.46)	8.13 (0.74)
2	4.06 (0.67)	4.06 (0.67)	10.05 (0.66)
4	4.95 (1.22)	4.95 (1.23)	13.12 (1.51)

Table I: Control Loop reaction time

As expected, the time from detection to time to action increases according to the observation period setting. The fulfillment is restored after 11, 13, and 17 seconds, for observation periods of 1, 2, and 4 seconds, respectively. We performed another set of tests where the HbH telemetry is always on, to evaluate how much benefit this would bring at the cost of the telemetry overhead. The results are shown in Figure 7 and in the second part of Table I. These results show a faster reaction time from the control loop. From the detection of an issue, the system took action in a fraction of a second and recovered from the issue in a shorter time.

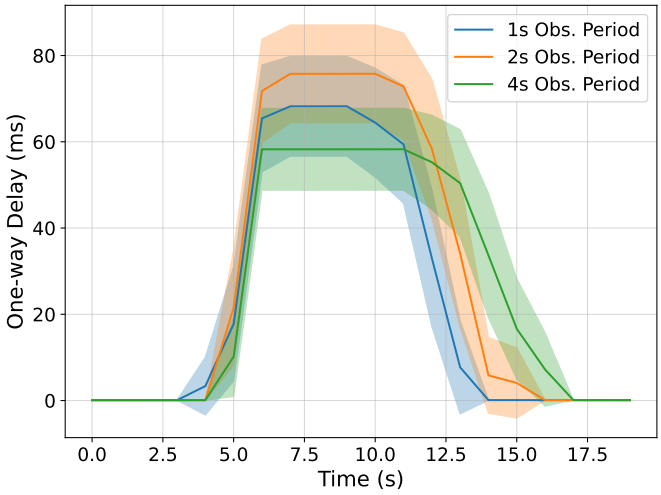

Figure 6: Delay measurements and system recovery according to observation period for on-demand HbH telemetry

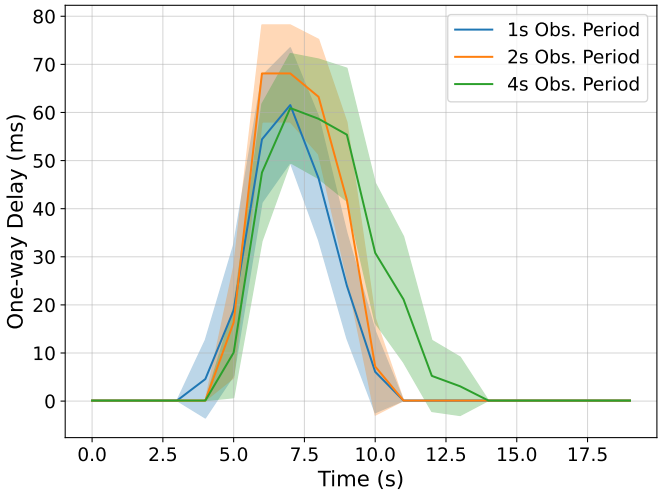

Figure 7: Delay measurements and system recovery according to observation period for always-on HbH telemetry

The time needed to recover from failures in the order of a few seconds can be excessive for many time-sensitive applications. The always-on approach for HbH telemetry significantly improves the action time after an issue is detected. The detection time can be further improved by reducing the telemetry reporting frequency (currently once per second) and supporting observation period configurations under 1 second.

VII. Conclusion and Future Work

This paper presents a solution for network slice management over TSN networks for industrial and multimedia applications.

Building on recent advances in network telemetry and flexible network control enabled by INT and TSN standards, we describe a bottom-up approach for automated slice management inspired by ZSM and Intent-based Networking definitions. We detail the improvements implemented on our TSN Controller architecture, as well as the data model to compose application instances, the set of actions to prepare the network for these applications, and the steps taken by a Control Loop to monitor and take actions on the network upon detecting issues. These specifications can give important insights to other researchers and developers working on automating network operations.

The current implementation takes action on a single slice at a time. In the future, a global optimization function will be used to adjust the network configuration. We have also used a simple control action, migrating the flow to a different slice. In future steps, the actions will involve more complex actions such as adding slots or changing their duration. The results from this PoC show the feasibility of the complete solution to automate the process of identifying and acting on the network to achieve dynamic and application-specific network troubleshooting.

ACKNOWLEDGMENT

This research is partially funded by the imec ICON project VELOCe - VErifiable, LOw-latency audio Communication (Agentschap Innoveren en Ondernemen project nr. HBC.2021.0657). This research is also supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001, São Paulo Research Foundation (FAPESP) with Brazilian Internet Steering Committee (CGI.br), grants 2018/23097-3 and 2020/05182-3, Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERENCES

[1] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, and X. Costa-Pérez, "OVNES: Demonstrating 5G network slicing overbooking on real deployments," in *INFOCOM 2018 - IEEE Conference on Computer Communications Workshops*, 2018, pp. 1–2. [Online]. Available: http://doi.org/10.1109/INFCOMW.2018.8406867

[2] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," pp. 32–39, 2016. [Online]. Available: http://doi.org/10.1109/MCOM.2016.7514161

[3] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proceedings - IEEE INFOCOM*, no. 671584, 2017. [Online]. Available: http://doi.org/10.1109/INFOCOM.2017.8057230

[4] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019. [Online]. Available: https://doi.org/10.1109/JPROC.2019.2905334

[5] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018. [Online]. Available: http://doi.org/10.1109/MCOMSTD.2018.1700076

[6] G. Miranda, E. Municio, J. Haxhibeqiri, J. Hoebeke, I. Moerman, and J. M. Marquez-Barja, "Enabling time-sensitive network management over multi-domain wired/wi-fi networks," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023. [Online]. Available: https://doi.org/10.1109/TNSM.2023.3274590

[7] J. Haxhibeqiri, P. H. Isolani, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, "In-Band Network Monitoring Technique to Support SDN-Based Wireless Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 627–641, 2021. [Online]. Available: https://doi.org/10.1109/TNSM.2020.3044415

[8] G. Miranda, J. Haxhibeqiri, J. Hoebeke, I. Moerman, D. F. Macedo, and J. M. Marquez-Barja, "Flexible wired/wi-fi tsn networking through sdn and soft traffic control," in *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2023, pp. 178–179. [Online]. Available: http://doi.org/10.1109/NFV-SDN59219.2023.10329729

[9] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2023. [Online]. Available: http://doi.org/10.1109/COMST.2022.3215919

[10] B. K. Saha, D. Tandur, L. Haab, and L. Podleski, "Intent-based networks: An industrial perspective," in *Proceedings of the 1st International Workshop on Future Industrial Communication Networks*, ser. FICN '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 35–40. [Online]. Available: https://doi.org/10.1145/3243318.3243324

[11] K. Mehmood, D. Palma, and K. Kralevska, "Mission-critical public safety networking: An intent-driven service orchestration perspective," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 37–42. [Online]. Available: http://doi.org/10.1109/NetSoft54395.2022.9844105

[12] W. Cerroni, C. Buratti, S. Cerboni, G. Davoli, C. Contoli, F. Foresta, F. Callegati, and R. Verdone, "Intent-based management and orchestration of heterogeneous openflow/IoT SDN domains," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–9. [Online]. Available: http://doi.org/10.1109/NETSOFT.2017.8004109

[13] C. E. Rothenberg, D. A. Lachos Perez, N. F. Saraiva de Sousa, R. V. Rosa, R. U. Mustafa, M. T. Islam, and P. H. Gomes, "Intent-based Control Loop for DASH Video Service Assurance using ML-based Edge QoE Estimation," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 353–355. [Online]. Available: http://doi.org/10.1109/NetSoft48620.2020.9165375

[14] S. K. Perepu, J. P. Martins, R. S. S, and K. Dey, "Intent-based multi-agent reinforcement learning for service assurance in cellular networks," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 2879–2884. [Online]. Available: http://doi.org/10.1109/GLOBECOM48099.2022.10001426

[15] E. Coronado, S. N. Khan, and R. Riggio, "5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, 2019. [Online]. Available: http://doi.org/10.1109/tnsm.2019.2908675

[16] M. Richart, J. Baliosian, J. Serrat, J.-L. Gorricho, and R. Agüero, "Slicing in WiFi Networks Through Airtime-Based Resource Allocation," *Journal of Network and Systems Management*, vol. 27, no. 3, pp. 784–814, 07 2019. [Online]. Available: https://doi.org/10.1007/s10922-018-9484-x

[17] P. H. Isolani, N. Cardona, C. Donato, J. Marquez-Barja, L. Z. Granville, and S. Latre, "SDN-based Slice Orchestration and MAC Management for QoS delivery in IEEE 802.11 Networks," pp. 260–265, 2019. [Online]. Available: http://doi.org/10.1109/sds.2019.8768642

[18] Cisco, "Time-Sensitive Networking: A Technical Introduction White Paper," *Cisco Public White Paper*, 2017. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf

[19] ETSI, "Zero-touch network and service management (zsm); intent-driven autonomous networks; generic aspects," ETSI Doc. Number: GR ZSM 011, February 2023. [Online]. Available: https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=67446

[20] 3GPP, "Management and orchestration; Intent driven management services for mobile networks," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.312, 09 2023, version 18.1.1. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3554

[21] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016. [Online]. Available: doi.org/10.1109/IEEESTD.2016.8613095