Evaluating Dynamic Path Reconfiguration for Time Sensitive Networks

Nurefşan Sertbaş Bülbül , Doğanalp Ergenç, Mathias Fischer
Department of Computer Science, Universität Hamburg, Germany
Email:{sertbas, ergenc, mfischer}@informatik.uni-hamburg.de

***Abstract*—In time-sensitive networks (TSN) based on 802.1Qbv, i.e., the time-aware Shaper (TAS) protocol, precise transmission schedules and, paths are used to ensure end-to-end deterministic communication. Such resource reservations for data flows are usually established at the startup time of an application and remain untouched until the flow ends. There is no way to migrate existing flows easily to alternative paths without inducing additional delay or wasting resources. Therefore, some of the new flows cannot be embedded due to capacity limitations on certain links which leads to sub-optimal flow assignment. As future networks will need to support a large number of low-latency flows, accommodating new flows at runtime and adapting existing flows accordingly becomes a challenging problem. In this extended abstract we summarize a previously published paper of us [1]. We combine software-defined networking (SDN), which provides better control of network flows, with TSN to be able to seamlessly migrate time-sensitive flows. For that, we formulate an optimization problem and propose different dynamic path configuration strategies under deterministic communication requirements. Our simulation results indicate that regularly reconfiguring the flow assignments can improve the latency of time-sensitive flows and can increase the number of flows embedded in the network around $\approx$ 4% in worst-case scenarios while still satisfying individual flow deadlines.**

***Index Terms*—SDN, dynamic flow migration, reconfiguration, TSN, path computation**

I. Introduction

The real-time Internet of things (IoT) driven by 5G networks and autonomous vehicular networks relies on low-latency and deterministic networking. Many safety-critical applications served by such networks, e.g., robots in automation environments, require a bounded latency and reliable data delivery. A violation of latency constraints can, in the worst case, result in physical damage. To address the real-time and deterministic communication requirements of time-sensitive and safety-critical systems, several TSN standards have been proposed by the IEEE 802.1 working group.

In TSN and on the basis of the IEEE 802.1Qcc Stream Reservation Protocol, end-hosts declare their traffic requirements before the actual communication [2]. Then, the time-critical transmissions are scheduled to prevent undesired queuing delays, and all networking elements on the routing path enforce these schedules. As the number of flows increases, it might be necessary to migrate flows to alternative paths to obtain better resource utilization. However, in TSN flow paths are configured initially and then remain fixed until the end of the flow. Reconfiguring flow paths thus would require taking down a flow and making a new reservation for it, which introduces an additional delay and signaling effort.

There is rich related work in other fields that can be adapted to this flow migration problem. For example, there is related work that formalizes the virtualized network function (VNF) mapping and scheduling problem as a mixed-integer optimization problem (MIP) by considering VNF requirements like delay and priority [3], [4] and the maximum resource consumption [5], e.g., memory and CPU. Since these optimization problems are NP-hard, several models have been proposed to decide between either migration of VNFs or their re-instantiation. Alternatively, some authors solved this problem via an approximation, which achieves close-to-optimal performance in terms of acceptance ratio and maximum link load ratio [6]. In multi-tenant cloud networks, authors solve the dynamic flow embedding problem via a simple greedy algorithm to reallocate resources at the edge while still providing predictable latencies [7].

However, there is limited literature focusing on the incremental addition of flows in TSN. Most related papers either only consider routing path changes; omit schedule changes or do not evaluate the migration cost [8], [9]. Since TSN is designed to isolate flows either spatially through different routes or temporally through different schedules, separating routes from schedules may limit the QoS.

In this extended abstract, we address the problem of dynamic path (re)configuration for TSN networks based on SDN to enable the migration of network flows under strict latency constraints. For that, we formulate a time-sensitive optimal routing problem (TSOR) with mixed-integer linear programming and present different path (re)configuration strategies by adding varying degrees of routing constraints to TSOR in [1]. This is an extended abstract based of this former work. Unlike the time-division multiple access-based schedules in which multiple frames are transmitted one after the other [10], we embed the gate opening frequency into our path computation formalization as a TSN-specific aspect. To evaluate our approach we have built a realistic simulation model of a TSN network in OMNeT++ and compared our simulation results with the optimal solution obtained by solving TSOR. With that, we can quantify the number of flows embedded into the network, and the reconfiguration cost for a TSN network. The simulation results indicate that our alternative path configuration strategies can embed more flows up to 4% without any additional delay to the time-sensitive traffic with

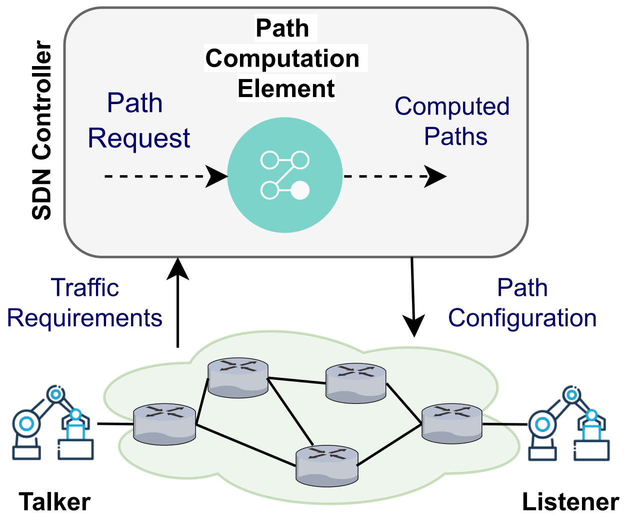

Fig. 1. Overall system block diagram

a moderate (re)configuration overhead.

The remainder of this abstract is structured as follows: Section II summarizes our overall system and introduces the TSOR. Section III describes our evaluation results. Section IV concludes the paper and summarizes future work.

II. System Design

This section introduces our dynamic path (re)configuration framework for TSNs by making use of SDN as shown in Fig. 1. The global view of the centralized SDN controller enables the deployment of centralized routing algorithms and eases the configuration. Thus, routing paths can be reconfigured dynamically at runtime considering the requirements of time-sensitive environments. To communicate in such a network, the end-host, a talker in TSN, needs to inform the network to allocate required resources before the actual communication takes place by sending a talker-advertise message that includes the traffic requirements of the flow. This message will be forwarded to the SDN controller that can compute an assignment, which efficiently uses network resources, and that can respond to varying network conditions. The received request is recorded and then, *path computation element (PCE)* computes a new path considering flow traffic requirements, current resource utilization, and the topology. The resulting forwarding rules are then distributed via the data plane.

In some cases, the computed path might be blocked already, so that existing flows need to be migrated. Here, flows are migrated sequentially, ensuring consistency, and then the new flow is accommodated. After all forwarding rules are successfully updated at the respective switches, the controller sends a listener-ready message to the talker. Then, since the required resources for the transmission have already been provided, the talker starts to send data via the allocated path.

A. Time Sensitive Optimal Routing

In this section, we formulate the time-sensitive optimal routing (TSOR) problem to migrate high-priority flows to suitable paths. Using the model, we find (i) E2E paths for given demands under different quality of service (QoS) requirements and (ii) a simple gate configuration per port minimizing the overall E2E communication latency.

The gate configuration is the primary mechanism of the core TSN protocol, 802.1Qbv Time-aware Shaper (TAS) that ensures end-to-end deterministic communication for the streams of different QoS classes via strict time-division scheduling [11], [12]. In TAS, on each (egress) port of a switch, there are eight priority queues that store frames of streams with different priorities, including best-effort, before they are forwarded to the destination. Each queue is controlled by a gate to forward a frame. When a gate is open, the next frame in the respective queue is sent at a given time. Eight gates corresponding to the eight priority classes are configured by a gate driver via a gate control list (GCL) that decides which gate(s) should be open at which time. This mechanism overall constitutes a frame-forwarding schedule with respect to the priority classes to satisfy strict timing requirements.

Accordingly, we utilize two optimization variables for our problem. x_{dp} is a binary variable to decide if demand d is assigned to path p. g_{es}, is a continuous variable representing the frequency of an open gate on the egress port of link e for the service class s among eight possible classes. While $g_{es} = 1$ infers that the gate for s should be open all the time and the entire resource of link e is assigned for that type of demand, $g_{es} \approx 0$ means that any demand of service type s is not active at all on the respective port and thus, the gate is closed.

TSOR formulation is given in Equations 1-7. Constraint (2) ensures that each demand $d \in D$ is assigned to exactly one path $p \in P$. Constraint (3) is the link capacity constraint and guarantees that each link e has sufficient capacity c_e to handle the total load h_d of all demands $d \in D$ assigned to any path p including e, s.t. $\alpha_{ep} = 1$. Constraint (4) represents the configuration of the GCL of e for each class of service s. Here, a gate for class s is decided to be open on link e proportional to value of g_{es}. Constraint (5) is the latency constraint to ensure that the E2E latency on path p is always below the latency requirement of demand d, which is l_d. Since the smaller values of g_{es} cause an increased latency due to queueing delay in the respective gate, a queueing delay factor l_e^q is added proportional to the $1-g_{es}$, when the gate is closed. Besides, a base delay l_e^o including delays such as processing and propagation delay, is considered for each link. We use $l_e^q = 0.5$ and $l_e^o = 1.0$ in our simulations. Constraint (6) forces g_{es} to be proportional to the total traffic load of service type s forwarded through the link e. Otherwise, it would lead to congestion and unexpected packet drops. Lastly, constraint (7) fixes the demands that are already assigned to a certain path p, i.e., $a_{dp} = 1$ from an existing configuration. Our objective function (1) minimizes the overall latency of the selected paths which is calculated similar to the latency constraint (5).

$$\min \sum_{d\in D}\sum_{p\in P}\sum_{e\in E} x_{dp}\alpha_{ep}\left[l_e^o + l_e^q(1-g_{es})\right] \tag{1}$$

$$\sum_{p\in P} x_{dp} = 1 \qquad \forall d \in D \tag{2}$$

$$\sum_{d\in D}\sum_{p\in P} x_{dp}\alpha_{ep}h_d \le c_e \qquad \forall e \in E \tag{3}$$

$$\sum_{s\in S} g_{es} = 1 \qquad \forall e \in E \tag{4}$$

$$\sum_{p\in P}\sum_{e\in E} x_{dp}\alpha_{ep}\left[l_e^o + l_e^q(1-g_{es})\right] \le l_d \qquad \forall d \in D \tag{5}$$

$$g_{es} - \sum_{d\in D}\sum_{p\in P} x_{dp}\alpha_{ep}\frac{h_d}{c_e} \ge 0 \qquad \forall e \in E, \forall s \in S \tag{6}$$

$$x_{dp} \ge a_{dp} \qquad \forall d \in D, \forall p \in P \tag{7}$$

Lastly, we take TSOR as a linear problem that makes it more convenient to be solved by state-of-the-art linear optimization tools. For further information on complexity and linearization, readers can follow our previous study [13].

B. Path Configuration Strategies

The strict time constraints of such environments lead to the accommodation of flows on certain paths and leave these paths untouched as long as the path meets the delay requirements of *TSOR*. However, incremental flow scheduling in TSN will change the link and switch utilization over time and affect the end-to-end latency of chosen paths. Thus, we present different path configuration strategies with varying degrees of routing constraints. Removing such constraints from the model improves the solution quality, i.e., more flows can be embedded, but adds computational complexity. If all constraints can be satisfied, PCE returns a solution that may require changes in the previous flow assignments. Otherwise, it rejects the flow.

a) Reconfiguration at every path request: To maximize the number of embedded flows, replanning all path configurations from scratch, e.g., by removing the pre-assignment constraint from TSOR, is an *unrestricted* strategy that we call **TSOR-U**. TSOR-U allows all flows to be reconfigured, e.g., migrated to different paths, to find the optimal allocation, including newly arriving flows. Even though its flexibility, it introduces additional configuration overhead, e.g., control packets exchanged between the controller and switches, that might violate the deterministic communication requirements.

b) Reconfiguration at every k-th path request: Since *TSOR* will lead to an inefficient use of resources, especially for a larger number of flows, a periodic reconfiguration is an alternative approach. For that, we propose **TSOR-P** that reconfigures the network after having received k flow requests. Therefore, it can adapt the reconfiguration period dynamically in dependence on the arrival rate of flows. This strategy can also be improved by monitoring the system and extracting a pattern for the latency violations to schedule or derive optimal reconfiguration times.

c) Threshold-triggered reconfiguration: The most straightforward strategy **TSOR-T** a network operator can apply is to use *TSOR* to embed a new flow and to compute *TSOR-U* to measure how close the resulting solution is to *TSOR-U*. Then, we only reconfigure if the computed objective exceeds a pre-defined threshold.

III. Evaluation

In this section, we evaluate the performance of time-sensitive packet delivery of the different path (re)configuration strategies from Section II that we implemented in CPLEX 12.7.0. Furthermore, we simulated a TSN network in OMNeT++ v5.5.1 using the inet framework and SDN4CoRE framework [14]. For our experiments, we used the *Integra* topology from the Topology Zoo dataset [15]. Since there is no publicly available data set for TSN traffic, we generate both time-triggered (TT) and best-effort (BE) traffic with the parameters described in [12], [16].

Table I shows the simulation results for *TSOR*. Since *TSOR-U* can fully utilize all data plane resources, it has a higher acceptance ratio. However, *TSOR* leaves less room to accommodate new flows, as the previously installed flows are not touched. So this limits the acceptance ratio of *TSOR*. Since *TSOR-P* and *TSOR-T* allow partial reconfiguration, their acceptance ratio differs from others.

We also measured the delayed frame ratio (DFR) which is the ratio of delayed frames whose delay exceeds the delay requirement of the received frames. Since *TSOR* cannot flexibly change the assigned paths based on the current network status, some low-bandwidth links may be overloaded. Thus, it has the highest DFR. Here the first expectation is that *TSOR-U* has the lowest DFR since it can flexibly use the resources. However, the network utilization directly influences the delay, but the impact gets more significant the more utilized a network is. Since *TSOR-U* can embed more flows, the network load and therefore the average TT latency increases, which also increases the DFR.

TABLE I
OMNeT simulation results

	TSOR	TSOR-P	TSOR-T	TSOR-U
Acceptance Ratio [%]	49.2	51.18	51.93	52.81
Delayed Frame Ratio [%]	5.83	4.94	4.47	5.41
Configuration Time [us]	17	63.7	137.4	189

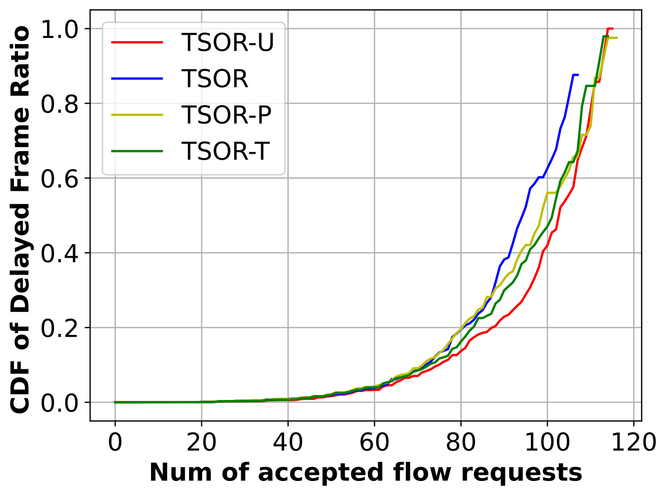

Fig. 2. Delayed frame ratio respect to number of accepted flows

To highlight this, we also plotted the cumulative density function of the DFR in dependence on the number of accepted flows in Fig. 2. Here, for the same number of accepted flows, *TSOR-U* has the lowest DFR which supports our claim. Since *TSOR-P* is triggered for every kth received request, it can adapt resources depending on the received traffic rate. In *TSOR-T*, reconfiguration is only triggered if the solution quality in terms of latency exceeds a certain threshold, e.g., 1%. Thus, both *TSOR-P* and *TSOR-T* perform better than *TSOR* and are close to *TSOR-U*. However, after a certain point, which is $\approx$ 110 flows in this experiment, the number of delays increases significantly in *TSOR-U*. Thus, even though it has the lowest DFR until there, it will not performs better than the *TSOR-P* and *TSOR-T* after that point.

Lastly, we measure the configuration time which includes the potential migration latency and time for data plane configuration and is thus directly affected by the number of reconfigurations in the data plane. Frequent reconfigurations in *TSOR-U* increase configuration time, while limited reconfigurations in *TSOR* result in lower configuration time. The results of *TSOR-P* and *TSOR-T* are located in between the results of *TSOR* and *TSOR-U*.

For our simulations we excluded the time to solve the MILP optimization models, i.e., they were solved in zero time, to fairly compare the different embedding methods. However, the actual time required to solve the MILP is still in a reasonable range, e.g., around 7.3s for 100 flows. This is a significant overhead for TSN, but only in *TSOR-U* the MIP needs to be solved for every new flow. *TSOR* builds upon the previous solution. Thus, sorting in a new flow for *TSOR* requires only around 4 ms in our settings. The other strategies (TSOR-T/P) require solving the optimization model for all flows from time to time. They will then migrate embedded flows from their potentially sub-optimal paths to their optimal ones. Even though their performance is highly related to the configuration parameters such as k in *TSOR-T* and the triggering threshold in *TSOR-P*, this migration can be done seamlessly without packet loss and increased latencies.

It should be noted that the performance of the presented heuristics is highly dependent on the selected parameters. For instance, in a uniformly distributed flow scenario, e.g., a new flow appears every second, reconfiguring at every kth request approach reconfigures the network periodically while in other distributions reconfiguration may be delayed due to varying flow arrival times. Also, increasing the k value will result in lower overhead (e.g., configuration time) but probably also a low acceptance ratio. Similarly, for the threshold triggered configuration, decreasing the threshold would result in more frequent reconfigurations and increased overhead. For further evaluation results, readers can follow our previous study [1].

IV. Conclusion

This extended abstract that summarizes previous work [1] presents and evaluates dynamic path configuration strategies for SDN-enabled time-sensitive networks. We defined a restricted optimal flow placement model that adapts path assignments based on the current resource utilization and presents three heuristics to maximize the number of accepted flows while meeting the QoS requirements of TT applications. In a highly dynamic small/medium scale environment where flows are added and removed over time, *reconfiguring at every path request* would be more appropriate for utilizing all resources more efficiently. However, it may not be desired for large networks as frequent flow migrations and thus additional delay might be the result. In such scenarios, other heuristics that we present, e.g., *reconfiguration at every k-th request* or *threshold triggered reconfiguration*, appear as promising reconfiguration solutions.

Our experiments provide insights on (i) when reconfiguration is needed and (ii) which flows need to be reconfigured, i.e., migrated. In the future, we would like to include different aspects, e.g., minimizing the required flow migrations and splitting flows into multiple paths, e.g., enabling bifurcated flows. Such aspects result in more balanced flow placement with better resource utilization. Moreover, we plan to consider the trade-off between reconfiguration and performance gains. To do so, we will convert our optimization problem as a multi-objective reconfiguration scheme by adding a dual objective to minimize the reconfiguration cost by computing the trade-off. With that, we will be able to estimate the reconfiguration cost and related benefits (reduced E2E latency) before taking a decision.

References

[1] N. Sertbaş Bülbül, D. Ergenç, and M. Fischer, "Towards SDN-based dynamic path reconfiguration for time sensitive networking," in *NOMS 2022 - 2022 IEEE/IFIP Network Operations and Management Symposium*, 2022.

[2] "IEEE standard for local and metropolitan area networks–bridges and bridged networks - amendment 31: Stream reservation protocol enhancements and performance improvements," 2018.

[3] J. Li, W. Shi, P. Yang, and X. Shen, "On dynamic mapping and scheduling of service function chains in sdn/nfv-enabled networks," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[4] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, “Dynamic, latency-optimal vnf placement at the network edge,” in *IEEE INFOCOM*, 2018.

[5] H. Hawilo, M. Jammal, and A. Shami, “Orchestrating network function virtualization platform: Migration or re-instantiation?” in *IEEE Cloud-Net*, 2017.

[6] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, “Delay-aware virtual network function placement and routing in edge clouds,” *IEEE Transactions on Mobile Computing*, vol. 20, pp. 445–459, 2021.

[7] A. Van Bemten, N. Dherić, A. Varasteh, S. Schmid, C. Mas-Machuca, A. Blenk, and W. Kellerer, *Chameleon: Predictable Latency and High Utilization with Queue-Aware and Adaptive Source Routing*. New York, NY, USA: Association for Computing Machinery, 2020, p. 451–465.

[8] P. Danielis, G. Dán, J. Gross, and A. Berger, “Dynamic flow migration for delay constrained traffic in software-defined networks,” in *IEEE GLOBECOM*. IEEE, 2017, pp. 1–7.

[9] A. Alnajim, S. Salehi, and C.-C. Shen, “Incremental path-selection and scheduling for time-sensitive networks,” in *IEEE GLOBECOM*, 2019.

[10] N. G. Nayak, F. Dürr, and K. Rothermel, “Incremental flow scheduling and routing in time-sensitive software-defined networks,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2017.

[11] “IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic,” pp. 1–57, 2016.

[12] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, “Performance comparison of ieee 802.1 TSN time aware shaper (TAS) and asynchronous traffic shaper (ATS),” *IEEE Access*, vol. 7, pp. 44 165–44 181, 2019.

[13] N. Sertbaş Bülbül, D. Ergenç, and M. Fischer, “SDN-based self-configuration for Time-Sensitive IoT Networks,” in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 73–80.

[14] T. Häckel, P. Meyer, F. Korf, and T. Schmidt, “SDN4CoRE: A simulation model for software-defined networking for communication over real-time ethernet,” in *International OMNeT++ Community Summit*, 2019.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The internet topology zoo,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[16] A. Ademaj, D. Puffer, D. Bruckner, G. Ditzel, L. Leurs, M.-P. Stanica, P. Didier, R. Hummen, R. Blair, and T. Enzinger, “Industrial automation traffic types and their mapping to QoS/TSN mechanisms,” 2019.