Synthesis of Queue and Priority Assignment for Asynchronous Traffic Shaping in Switched Ethernet

Johannes Specht[1] and Soheil Samii[2,3]
johannes.specht@uni-due.de, soheil.samii@{gm.com, liu.se}
[1]University of Duisburg–Essen, Germany
[2]General Motors R&D, Michigan, USA
[3]Linköping University, Sweden

***Abstract*—Real-time switched Ethernet communication is of increasing importance in many cyber-physical and embedded systems application areas such as automotive electronics, avionics, and industrial control. The IEEE 802.1 Time-Sensitive Networking (TSN) task group develops standards for real-time Ethernet, for example a time-triggered traffic class (IEEE 802.1Qbv-2015). New application areas, such as active safety and autonomous driving using radar, lidar, and camera sensors, which do not fall into the strictly periodic, time-triggered communication model, require a flexible traffic class that can accommodate various communication models while still providing hard real-time guarantees. In our previous work, we developed such a traffic class, Urgency-Based Scheduler (UBS), and its worst-case latency analysis. UBS is currently under standardization (P802.1Qcr) in the TSN task group. In this paper, we introduce and solve the UBS synthesis problem of assigning hard real-time data flows to queues and priority levels to queues, the main parameters that determine communication latencies. The synthesis problem is particularly challenging due to the flexibility offered by UBS to aggregate flows and assign individual priority levels per network hop. We present an SMT approach, a cluster-based heuristic, and an extensive experimental evaluation.**

I. Introduction

Due to increasing bandwidth requirements, real-time communication on Ethernet is becoming of increasing importance in many embedded application domains. Examples include industrial automation and control, automotive networking for active safety and automated driving applications, and avionics. Although several time-driven approaches like 802.1Qbv [1] or TTEthernet [2] exist to enhance standard Ethernet technology with real-time properties such as time determinism and packet delivery guarantees, these rely on coordination efforts at design time to define time schedules globally harmonized among all switches, end devices and application software. Another limitation in such time-driven communication approaches is the strong dependency on availability of a global time base: For example, the loss or failure of time synchronization (e.g., Grand Master failure in IEEE 802.1AS) will lead to loss of data communication.

Urgency Based Scheduler (UBS) is a new real-time traffic class that is asynchronous in that it does not rely on time synchronization or progression of time for scheduling decisions [3]. It improves upon Ethernet AVB [4], which has up to two traffic classes with multiple flows aggregated. Traffic in these two classes are scheduled for transmission by Ethernet end stations and switches based on a credit-based traffic shaper. UBS is in fact currently in progress of standardization within the IEEE 802.1 Working Group.[1] UBS operates independently of time synchronization and provides hard real-time guarantees for leaky-bucket constrained flows (i.e., more inclusive than periodic and sporadic traffic models) at high bandwidth utilization. These properties enable use of UBS for distributed real-time applications.

Several UBS parameters need to be synthesized and configured at design time. The synthesis problem addressed in this paper is the assignment of (a) flows to queues and (b) queues to fixed priority levels. For a given network topology and a set of deadline-constrained unicast and multicast flows, a flow-to-queue and queue-to-priority level assignments at each hop in the network must be decided. This synthesis problem is impractical to solve manually, and it is, due to combinatorial problem complexity, computationally infeasible in practice to explore all possible combinations. This calls for the need of effective and efficient synthesis algorithms.

Contribution: To solve the UBS synthesis problem, we present a Satisfiability Modulo Theories (SMT) based approach, which is optimal in that it always finds a solution if it exists. The performance of the SMT solution does not scale well for large applications. We propose a topology rank, cluster based heuristic to improve the synthesis efficiency. We compare the performance of all approaches by extensive experimental evaluation.

The remainder of this paper is organized as follows. In Section II, we present the system model and preliminaries, followed by related work in Section III. In Section IV, the synthesis problem at hand is explained and formulated. Section V describes our synthesis approaches, which are evaluated in Section VI. We summarize our findings in Section VII.

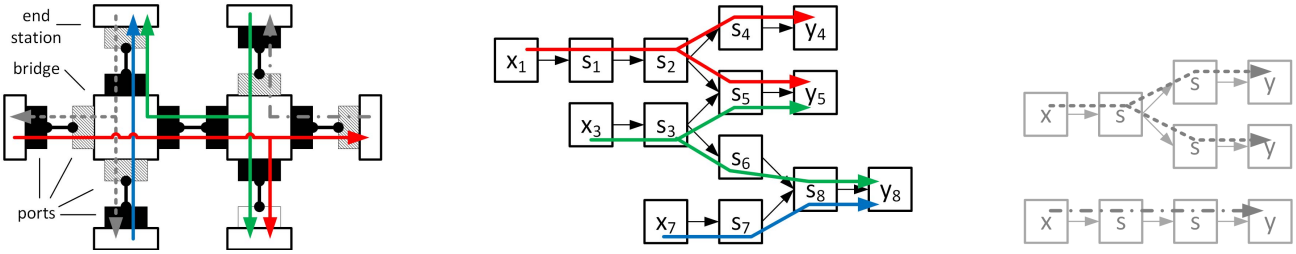
II. System Model and Preliminaries

A. Network Model

The network model used in this paper is an abstraction from a given physical switched Ethernet network topology and a set of data flows (unicast or multicast). The physical network comprises end stations and multi-port switches with

[1]The active project acronym and title within IEEE 802.1, respectively, are P802.1Qcr and "Asynchronous Traffic Shaping" [5].

1052-8725/17/31.00 ©2017 IEEE
DOI 10.1109/RTSS.2017.00024



(a) Physical switched network. (b) First feed-forward network. (c) Second and third feed-forward networks.

Figure 1. Decomposition of a physical network into three logical feed-forward networks (best viewed in color). Black ports in Fig. 1a are associated with the servers $\{s_1, ..., s_8\}$ in Fig. 1b, and gray ports are associated with the servers in Fig. 1c.

full-duplex, point-to-point connections. Moreover, we assume that flows are sent from one end station to one (unicast) or multiple (multicast) other end stations along pre-defined static paths. Our network model is a logical feed-forward network composed of a set of servers S, a set of sources X and a set of sinks Y, all connected by logical links in a feed-forward topology. This logical network represents a subset of the elements found in a physical network. For distinction, we use the terms *physical network* and *physical link* to refer to the physical representation, and just *network* and *link* to refer to the respective elements of the network model. In the physical network, we assume full-duplex physical links and single-ported end stations.

A physical network (Figure 1a) with at least one associated flow is a composition of one (Fig. 1b) and potentially more (Figure 1c) networks. Sources and sinks represent sending and receiving applications in end stations. Each server is associated with the output portion of one port. Every server implements the UBS traffic shaping algorithm according to [3] and has an associated constant transmission speed $r(s)$. The latter represents the speed of the connected physical link (e.g., 100 Mbit/s or 1 Gbit/s). Servers in the same switch are independent from each other. In conjunction with the full-duplex property of the physical links, this allows to independently consider only elements of a physical network in which flows cross at least one server. The links connecting sources, servers, and sinks represent the logical paths of the flows between the former. Each link has at least one associated flow. Thus, the flows implicitly define the feed-forward topology of a network. To give an example, server s_1 in Figure 1b belongs to the black port of the leftmost end station (x_1) in Figure 1a. As another example, server s_5 belongs to the lower black port of the rightmost bridge in Figure 1a (aggregating two flows, green and red).

We denote by F the set of all flows in a network, $F(s)$ as the subset of flows sent by a server $s \in S$, $S(f)$ as the subset of servers that send a flow f, $x(f)$ as the source of a flow f and $Y(f)$ as the set of sinks that receive flow f. If a server s is located in a switch, it receives flows from a set directly connected upstream servers, denoted as $S^-(s)$. A server s may send flows to another set of directly connected downstream servers, denoted as $S^+(s)$. For example, server s_2 from the network in Figure 1b receives flows from $S^-(s_2) = \{s_1\}$ and sends to $S^+(s_2) = \{s_4, s_5\}$. A server s may have one associated source $x \in X$ if both are located in the same end station, one associated sink $y \in Y$ if the server is located in the output port of a switch connecting to a receiving end station, or both. The presence of x or y implies $S^-(s) = \emptyset$ or $S^+(s) = \emptyset$, respectively. For example, server s_1 in Figure 1b receives flows from source x_1, and sink y_5 receives flows from server s_5.

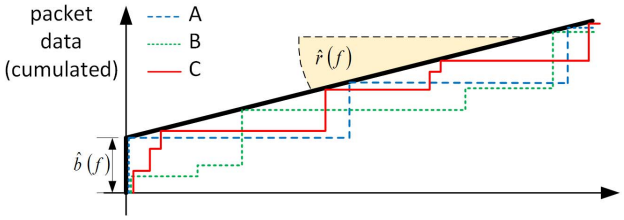


Figure 2. Three traffic patterns (A,B,C) satisfying the same leaky bucket constraint. The bold black line represents the limit by parameters $\hat{b}(f)$ and $\hat{r}(f)$ of the latter.

B. Traffic Model

Sources emit all flows asynchronously, each flow satisfying a leaky bucket constraint [6], [7]. Any server s with $S^-(s) \neq \emptyset$ re-enforces the respective constraint of every flow by interleaved traffic shaping, as shown in [3]. The leaky bucket constraint characterizes a flow f by an associated *burstiness* $\hat{b}(f)$ and a *leak rate* $\hat{r}(f)$. For any arbitrary time interval of duration d, the leaky bucket constraint limits the cumulated packet data $w(f, d)$ of flow f to

$$w(f, d) \leq \hat{b}(f) + d \cdot \hat{r}(f). \quad (1)$$

The same constraint in Figure 2 covers periodic traffic (pattern A), variable packet length AVB flows shaped with the Credit-Based Shaping Algorithm [8] (pattern B), and bursty traffic with back-to-back packet emissions (pattern C) [3].

C. Server Model

Each server s implements a set of shaped queues, denoted as $Q(s)$, which buffer all packets of flows $F(s)$. Additional implicit queues may exist for other traffic types like unshaped best effort traffic, which is typically present in switched

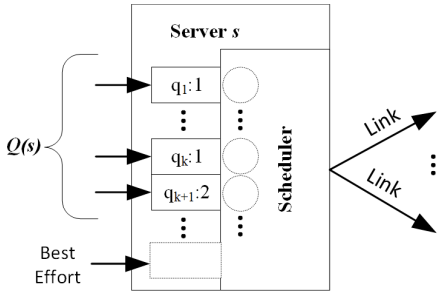


Figure 3. Model of a server in the feed-forward network model. Shaped queues $Q(s)$ are indicated by a circle in the scheduler.

Ethernet (Figure 3). Packets in each queue are selected for transmission in strict First-In-First-Out order. A scheduler arbitrates among all queues according to the priority levels and the shaping algorithm. In this paper, the interleaved *Length-Rate Quotient* (LRQ) algorithm [3] is used.

The (shaped) queues $Q(s)$ are assigned to fixed priority levels and flows $F(s)$ are assigned to queues $Q(s)$. For a queue q, set $F(q)$ denotes the set of assigned flows. For each server s, there is a queue-to-priority level assignment function

$$p(s): \quad Q(s) \quad \rightarrow \quad \{1, ..., |Q(s)|\} \ . \tag{2}$$

Numerically lower values in the target set represent higher priority levels. Note that $p(s)$ is not a one-to-one function, meaning that multiple queues may be assigned the same priority, subject to constraints described in this subsection. For a particular queue q, notation $p(q)$ is used to denote the priority level assigned to this queue.

The flow-to-queue assignment in a server s depends on the priority level assignments and the path of flows. Consider a server s receiving flows $F(s)$ from servers $S^-(s)$. For any two flows $f_i, f_j \in F(s)$, the following three constraints $QC1$, $QC2$ and $QC3$ define when f_i and f_j shall *not* be assigned to the same $q \in Q(s)$ [3]:

- QC1: Flows f_i and f_j shall not be assigned to the same queue q, if f_i and f_j are sent by different servers in $S^-(s)$.
- QC2: Flows f_i and f_j shall not be assigned to the same queue q, if f_i and f_j are sent in different priority levels by the same server $s^- \in S^-(s)$.
- QC3: Flows f_i and f_j shall not be assigned to the same queue q, if f_i and f_j are sent by server s in different priority levels.

In all other cases, flows f_i and f_j can be assigned to the same queue of server s. Note that $QC2$ and $QC3$ utilize flow-to-priority level assignments, which result from the combination of queue-to-priority level and flow-to-queue assignments. Constraints $QC1$ and $QC2$ only have to be considered for server-to-server transmissions (i.e., when $S^-(s) \neq \emptyset$). In contrast, $QC3$ applies to any server s.

The number of available queues in a server is implementation dependent. On one hand, more queues provide freedom for assignments. On the other hand, more queues typically increase the implementation effort of the scheduler. A server s in an N-port switch is always capable to serve an arbitrary number of flows, as long as $N-1 \leq |Q(s)|$ holds, albeit the flow-to-queue and queue-to-priority level assignments are restricted if a server only implements the set of *mandatory* $N-1$ queues [3]. In this paper, considering the feed-forward network model (which captures the network topology and the data flows), we can further reduce the set of mandatory queues according to $|S^-(s)| \leq |Q(s)|$ for any server s (recall that $|S^-(s)| < N$ always holds for an N-port switch).

In the remainder of this paper, the combination of network-wide flow-to-queue and queue-to-priority level assignments (details in Section IV) is called a configuration c out of a combinatorial configuration space C, where c is used as parameter for existing and subsequently introduced configuration-dependent symbols (e.g., $p(q)$ is extended to $p(c,q)$).

D. Delay Model and Analysis

In this section, we summarize the UBS end-to-end and per-hop delay analysis presented in [3]. The delay model in this paper is characterized by (a) *delay bounds* which describe the maximum delay experienced by a packet as a result of flow-to-queue and queue-to-priority level assignments, and (b) *deadlines* as defined by application requirements. Delay bounds and deadlines are end-to-end properties for every flow f. To account multicast in the general form, we describe both properties for each sink $y \in Y(f)$ as $\hat{d}(c,f,y)$, which denotes the end-to-end delay bound from the source of flow f to sink y, and $\bar{d}(f,y)$, which denotes the end-to-end deadline from the source of flow f to sink y.

The end-to-end delay bound $\hat{d}(c,f,y)$ is composed of per-hop delay bounds, each per link, along the path to a sink $y \in Y(f)$. Let $S(f,y)$ be the set of servers along this path with $S(f,y) = (s_1, ..., s_n)$ and $n = |S(f,y)|$. The end-to-end delay bound is the sum of partial delay bounds and given by

$$\hat{d}(c,f,y) = \hat{d}(c,f,x(f)) + \sum_{k=1}^{n-1} \hat{d}(c,f,s_k,s_{k+1}) + \hat{d}(c,f,s_n) \ . \tag{3}$$

The constituents of Equation 3 are described in the following.

$\hat{d}(c,f,x(f))$: A delay bound of form $\hat{d}(c,f,x(f))$ describes the delay bound from a source $x(f)$ to the first server. The respective link is located within end stations of the physical network. The delay bounds are end station implementation-specific (e.g., operating system, drivers, and application load), which is beyond the scope of this paper. Moreover, these delay bounds are unaffected by the flow-to-queue and queue-to-priority level assignments. Without loss of generality, we thus consider $\hat{d}(c,f,x(f)) = 0$.

$\hat{d}(c,f,s)$: A delay bound of form $\hat{d}(c,f,s)$ is used in (a) Equation 3 to describe the delay bound from the last server to the associated sink, and (b) in the subsequent explanation of $\hat{d}(c,f,s,s^+)$. Let $F_{HP}(c,f,s)$, $F_{SP}(c,f,s)$ and $F_{LP}(c,f,s)$ be the sets of all flows assigned to queues of server s, where these queues are assigned to a higher, the

same, or a lower priority level than $q(c,f,s)$, respectively. The delay bound $\hat{d}(c,f,s)$ is given by

$$\hat{d}(c,f,s) = \frac{\hat{l}(f)}{r(s)} + \frac{\sum_{g \in F_{HP}(c,f,s) \cup F_{SP}(c,f,s) \setminus \{f\}} \hat{l}(g) + \hat{l}_{LP}(c,f,s)}{r(s) - \sum_{g \in F_{HP}(c,f,s)} \hat{r}(g)} . \quad (4)$$

Term $r(s)$ is the transmission speed of server s (Sec. II-A), term $\hat{r}(g)$ is the leak rate of a flow g (Sec. II-B). Terms $\hat{l}(f)$ and $\hat{l}(g)$ describes the maximum packet length of flows f and g, respectively. For the LRQ shaping algorithm, the maximum packet length of a flow f equals the burstiness $\hat{b}(f)$ [3]. Term $\hat{l}_{LP}(c,f,s)$ is the maximum packet length that is sent by server s. This maximum covers all flows $F_{LP}(c,f,s)$, but also best effort packet transmissions at an additional (implicit) lowest priority level.

$\hat{d}(c,f,s,s^+)$: A delay bound of form $\hat{d}(c,f,s,s^+)$ describes the delay bound between two adjacent servers s and s^+ with $s^+ \in S^+(s)$. This delay bound uses Equation 4, and is

$$\hat{d}\left(c,f,s,s^+\right) = \max_{g \in F(c,q(f,s^+))} \left(\hat{d}(c,g,s)\right). \quad (5)$$

III. Related Work

For switched networks, a priority level assignment and dynamic admission control scheme is described for EDF scheduling [9], and in a modified variant for Static-Priority (SP) Scheduling in [10]. Configuration of a single unicast flow f is based on a round-trip message between source and sink. On the forward trip, the smallest possible end-to-end delay is computed, considering the existing priority level assignment fixed for the existing flows in the network and asserting that deadlines of existing flows remain met. This leads to the smallest configurable deadline (for EDF) and highest configurable priority level (for SP) at each server in the path. These configurations are then relaxed by distributing any remaining time slack to the deadline, resulting in a configuration of flow f. While such a round-trip based approach is applicable to plug-an-play networks (e.g., Multiple Stream Reservation Protocol, MSRP, the dynamic admission control protocol of AVB [8]), the options for priority level assignment is limited: Assignments for a new flow f and potential re-assignments of already established flows are limited to servers on the path of flow f. For example, re-distributing the slack of existing flows with overlapping, but different paths to enable smaller end-to-end delay bounds to flow f is not possible. This makes the success or failure of round trip-based priority level assignments sensitive to the order in which flows appear.

A priority level assignment heuristic (QoS-AFDX) for Avionics Full-Duplex Switched Ethernet (AFDX) with multiple priority levels is found in [11]. AFDX servers implement a work-conserving strict priority scheduling policy, identical to standard Ethernet [8]. Flows are assigned to network-wide priority levels (two in case of QoS-AFDX), which are assigned using Optimal Priority Assignment (OPA) [12], [13] in a sense that the entire network is considered as a single processor [11]. For our synthesis problem, however, this global view limits the configurations space significantly because UBS permits different flow-to-priority level assignments per server (e.g., the configuration in Figure 4c would not be found).

Heuristic Optimizing Priority Assignment (HOPA) is a priority level assignment algorithm for systems composed of processors and networks [14]. End-to-end task deadlines (unicast) are first distributed among the resources. Deadline Monotonic Priority Ordering (DMPO) [15] is used for priority level assignment per server, followed by a re-distribution of slack. HOPA is limited to unicast paths and would effectively require per-flow queues for disjoint per-flow priority levels. In contrast to, for example, CAN [16], UBS allows multiple flows to be assigned to a single priority level uniquely at a per-hop level (Section II-C). Last, dependencies among adjacent servers (QC1, QC2 and QC3 in Section II-C) are not considered.

Recent real-time Ethernet research has focused on topology synthesis and time-triggered communication. In [17], a heuristic is proposed for optimization of network topologies of industrial Ethernet networks using a simple, constant delay model. While relevant and applicable in early phases of system architecture definition, the heuristic does not consider network design tasks such as traffic deployment and synthesis considering data flows and real-time constraints. Further work has been done for time-triggered task and Ethernet schedule synthesis [18], based on a mixed integer programming model of time-triggered schedules, as well as precision and errors of the underlying time synchronization mechanism. In [19], an SMT-based approach is presented to integrate real-time applications and create their data communication schedules on TTEthernet, with the goal of minimizing re-certification costs. An SMT approach for schedule synthesis in TTEthernet networks with multiple hops was proposed in [20]. This approach has been adapted in [21] to support a recent IEEE 802.1Q amendment for scheduled traffic (IEEE 802.1Qbv-2015 [1]). Time-triggered schedule synthesis has also been studied specifically for industrial control systems utilizing Profinet [22].

Time-triggered Ethernet communication schedule synthesis has recently been extended in various directions, such as bandwidth optimization for best-effort traffic [23], mixed-criticality systems [24], and resilience to link failures [25]. TTEthernet network synthesis has also been extended towards routing synthesis [26], [27] and traffic class assignment among time-triggered, rate-constrained, and best effort traffic [28].

This paper advances the state-of-the-art of real-time communication synthesis by considering the unique properties of UBS (an asynchronous traffic scheduler [3] that is currently under standardization by the IEEE 802.1 TSN task group [5], [29]), offering per-hop flow-to-queue and queue-to-priority level assignments. This flexibility leads to an unprecedented synthesis complexity that we address with an exact SMT solution and an efficient heuristic based on topology ranks.

IV. Problem Statement and Motivation

A. Decision Variables

The decision variables of the herein considered problem are as follows (see also section II-C):

1) For each flow $f \in F$, select one queue assignment from the set $\prod_{s \in S(f)} Q(s)$
2) For each server $s \in S$, determine $p(c,q)$ for all $q \in Q(s)$, where $p(c,q)$ is the numeric priority level of queue q with $p(c,q) \in \{1, ..., |Q(s)|\}$.

This definition is generic in the sense that it allows that the priority level ordering of a set of flows can be different at different servers (a key property of UBS). This is in contrast to a class-based, network-wide, priority level ordering. Such an ordering would simplify synthesis due to the reduced combinatorial space. In the simplest case, there would be exactly one class, where all flows are associated to the same priority level in all servers. Finding the assignments for such a scheme would be trivial: All flows in a server are assigned to queues only according to $QC1$ based on the pre-defined paths, and $p(c,q) = 1$ are assigned to every queue q in the network. In contrast, finding assignments for the generic case is harder but, as illustrated with an example in Section IV-C, it is required for systems with tight deadline constraints.

B. Configuration Space and Synthesis Goal

The definitions in Section IV-A determine the full configuration space, which is reduced by the following configuration constraints for a configuration c:

CC1: In any server $s \in S$, queue assignment rules QC1, QC2 and QC3 (II-C) shall not be violated.

CC2: For any flow f, the end-to-end delay bound $\hat{d}(c,f,y)$ to any of its sinks y must be lower than or equal to the associated deadline $\overline{d}(f,y)$.

CC3: The required number of queues at any server $s \in S$ must not exceed the number of queues $|Q(s)|$.

We categorize configurations according to the following:

- A configuration is called a *valid configuration* if $CC1$ and $CC3$ are satisfied.
- A *solution* is a configuration that satisfies $CC1$, $CC2$ and $CC3$.

The synthesis problem at hand is to find a solution, where we consider all solutions to be equally well. We are thus addressing a constraint satisfaction problem and are not concerned with any optimization goal.

For explanations throughout this paper, we introduce the following additional configuration property:

CC4: In any server $s \in S$, no more queues are assigned than needed to satisfy $CC1$.

We now define two special configuration categories:

- A *minimal configuration* denotes a configuration that satisfies $CC1$, $CC3$ and $CC4$.
- A *single priority configuration* is a minimal configuration, where all queues are assigned to the same priority level.

Flow	**Sink**	$\bar{d}(f,y)$	$\hat{d}(c_i,f,y)$		
			$c_i = c_1$	$c_i = c_2$	$c_i = c_3$
f_1	y_3	6.5 µs	**7 µs**	6 µs	6.0 µs
f_2	y_3	8.0 µs	7 µs	7.7 µs	6.0 µs
f_3	y_3	9.5 µs	7 µs	7 µs	9.0 µs
	y_4	6.5 µs	6 µs	6 µs	5.7 µs
f_4	y_4	6.5 µs	6 µs	6 µs	5.7 µs

Figure 5. Deadlines and delay-bounds for the configurations in Fig. 4.

C. Motivational Example

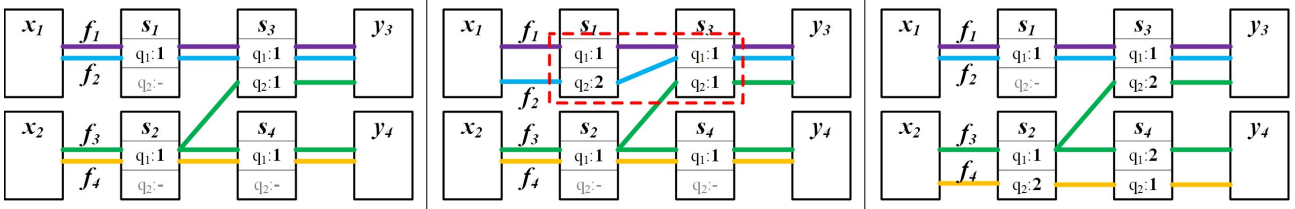
Consider the network in Figure 4a comprising four servers ($S = \{s_1, ..., s_4\}$), two associated sinks and sources ($X = \{x_1, x_2\}$ and $Y = \{y_3, y_4\}$), four flows ($F = \{f_1, ..., f_4\}$), and two queues per server ($Q(s) = \{q_1, q_2\}$ for all $s \in S$). The figure shows two queues in each server. The flow to queue assignment is depicted, and the priority level (lower value is higher priority level) of each queue is shown next to the queue separated by a colon. Note that we have three unicast flows (f_1, f_2, f_4) and one multicast flow (f_3). To simplify the discussion for this example, we consider network-wide identical transmission speeds of $r(s) = 1\,Gbit/s$ for all $s \in S$, leak rates of $\hat{r}(f) = 0.25\,Gbit/s$, and maximum packet lengths of $\hat{l}(f) = \hat{l}_{LP}(f,s) = 1000\,bit$ for all $f \in F$. The first configuration c_1 has a flow-to-queue assignment given by Figure 4a and assigns priority level 1 (highest) to all queues in all servers. Thus, c_1 is a single priority configuration. While a single priority configuration always satisfies $CC3$, the delay bound $\hat{d}(c_1, f_1, y_3)$ calculated according to Section II-D (summarized in Figure 5) violates the deadline $\overline{d}(f_1, y_3)$, and thus violates $CC2$.

The second configuration c_2 (Figure 4b) would solve this violation by assignments $F(c_2, q_2) = f_2$ and priority level $p(c_2, q_2) = 2$ with $q_2 \in Q(s_1)$, which consequently would decrease delay bound $\hat{d}(c_2, f_1, y_3)$, compared to $\hat{d}(c_1, f_1, y_3)$. However, as summarized in Figure 5, configuration c_2 violates $QC2$ and thus constraint $CC1$. This is depicted with the dashed (red) rectangle, where flows f_1 and f_2 are sent in two different priority levels in server s_1 and aggregated to the same queue in s_3 (violating $QC2$).

The third configuration c_3 (Fig. 4c) satisfies all configuration constraints and is thus a solution for the given example. Delay bound $\hat{d}(c_3, f_1, y_3)$ is decreased by the assignments $F(c_3, q_2) = f_3$ and $p(c_3, q_2) = 2$ with $q_2 \in Q(s_3)$. This modification in isolation would satisfy deadline $\overline{d}(f_1, y_3)$, but it would increase the delay bound $\hat{d}(c_3, f_4, y_4)$ to 7.3 µs (not illustrated), causing deadline violation of $\overline{d}(f_4, y_4)$. This violation is avoided by the assignments in servers s_2 and s_4, and thus all deadlines are satisfied. The example demonstrates the design-space complexity and the trade-offs in finding a solution, which satisfies UBS assignment constraints and timing constraints.

V. Synthesis Approach

We developed and implemented the following solver algorithms: (1) a pure SMT solver (SMTS) and (2) a Topology



(a) Single priority level configuration c_1 (valid) violating deadline constraint CC2.

(b) Invalid configuration c_2 (violates $QC2$)

(c) Solution c_3 (and likewise a minimal configuration)

Figure 4. Configuration example (best viewed in color)

Rank Solver (TRS). These two algorithms are described in this section.

A. Pure SMT Solver (SMTS)

The pure SMT Solver (SMTS) instruments an SMT-back-end solver to find a solution. In our case, it is the Z3 SMT solver [30]. While there are other SMT solvers (e.g., Yices2 [31] or CVC4 [32]) and our underlying code generation is widely independent of Z3 [33], the decision for Z3 was of practical nature [34].

By definition, an SMT solver always finds a solution, if one exists. Moreover, it gives an indication about the relative performance of heuristic solvers like TRS, which we propose in the next subsection. The implementation of SMTS is as follows: First, corresponding equations and assertions are generated for the network model (Section II-A), the traffic model (Section II-B), the delay model (Section II-D), and the configuration constraints $CC1$, $CC2$, $CC3$ and $CC4$ in Section IV-B. Second, the back-end SMT solver (Z3) is used to determine whether the combination of this model is satisfiable. If it is satisfiable, the found solution (i.e., flow-to-queue and queue-to-priority level assignments) is retrieved from the solver.

While a solution must not necessarily satisfy $CC4$ (i.e., use of minimal number of queues per server), initial experiments with Z3 have shown that this additional constraint is beneficial because it reduces the combinatoric space and thus increases the performance. It is also beneficial from a systems perspective as it finds the best utilization of UBS resources while providing room for adding real-time flows. By the same reason, SMTS generates additional constraints that ensure that the queue-to-priority level assignments are in a non-decreasing order of priority levels per server.

Z3 implements specialized optimizations for integer numbers. Therefore, our SMTS approach is configurable to instrument the SMT solver with (a) Integer-typed data rates and (b) Real-typed data rates. The impact of these two types are evaluated in Section VI.

B. Topology Rank Solver (TRS)

The Topology Rank Solver (TRS) is an heuristic approach that reduces the computational effort by utilizing the feed-forward property of the network model. The operation of TRS is illustrated in Figure 6 and described in the following.

In a first phase, TRS decomposes the entire topology into clusters of one or more servers. The clusters are formed by

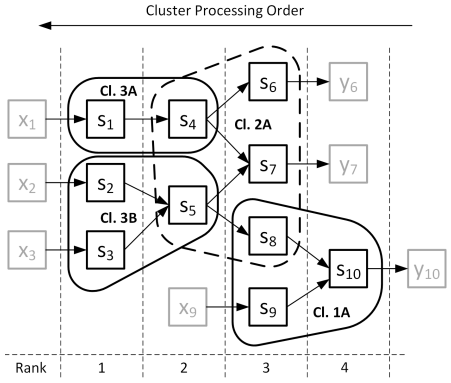


Figure 6. Illustration of TRS for cluster depth 2.

topology ranks, where a rank comprises all servers in a single "column" in Figure 6 (showing four ranks). The clusters have a depth of 1 or greater. The depth is a setup parameter of TRS and describes the number of adjacent ranks for decomposition into clusters, which typically overlap for depths greater than 1. For the topology in Figure 6, there are four clusters 1A, 2A, 3A and 3B for cluster depth 2. An individual cluster is defined to comprise a set of servers among the considered adjacent ranks connected by links between these ranks. For cluster depth 1, each single server forms an individual cluster.

After the construction of clusters, a second phase starts with a single priority configuration. This configuration is then modified by sequential processing of the individual clusters in reverse order of topology ranks. As a step in the processing of each cluster, a cost minimization is performed, which results in a flow-to-queue and queue-to-priority level assignment for this cluster. As a back-end for this cost minimization, we used the optimization capabilities of the Z3 SMT solver [35] in conjunction with different cost functions (defined later in this section). The basic utilization of the SMT solver is similar to SMTS. In addition, assignments at servers outside the cluster are fixed by respective assertions. In summary, TRS is parameterized by the cluster depth and the cost function that drives the cluster synthesis. We will now discuss these two aspects in detail.

Impact of the Cluster Depth: While the fixed flow-to-queue and queue-to-priority level assignments outside of the considered cluster reduce combinatoric space, the limitations implied by $QC1$ and, in particular, $QC2$ and $QC3$, become

stronger. For example, consider the processing of cluster 2A in Figure 6. When this cluster is processed, queues $Q(s_{10})$ may already be used to realize a wide variation of priority levels for flows $F(s_{10}) \cap F(s_8)$ due to earlier processing of cluster 1A. For example, all queues in s_{10}, to which flows $F(s_{10}) \cap F(s_8)$ are assigned, are themselves assigned to distinct priority levels. As a consequence, one particular queue $q_i \in Q(s_8)$ can only be assigned to all or a subset of flows $F(q_j)$, where $q_j \in Q(s_{10})$. If only a (non-empty) subset F_i of flows $F(q_j)$ with $F_i \subset F(q_j)$ is assigned to q_i, then at least one additional queue $q_k \in Q(s_8)$ is required for the remaining flows in $F(q_j)$, which means $F(q_k) \subseteq F(q_j) \setminus F(q_i)$. In the final consequence, an assignment of only a subset increases the required number of queues at servers of an earlier rank. A cluster depth greater than 1 can at least mitigate this effect to the price of larger clusters. Assignments of lower ranks within a cluster can be considered as a "look-ahead" assignment. For example, assignments for s_8 are analyzed during processing of cluster 1A, while they are finalized during processing of cluster 2A. Nonetheless, a larger cluster depth comes to the cost of increased computational effort due to the larger combinatoric space (TRS converges towards SMTS with increasing cluster depth).

Cost Functions: We have considered four cost functions denoted by $\overleftrightarrow{d}^{TRS}_{cost1}(c, S_{cluster})$, $\overleftrightarrow{d}^{TRS}_{cost2}(c, S_{cluster})$, $\overleftrightarrow{d}^{TRS}_{cost3}(c, S_{cluster})$ and $\overleftrightarrow{d}^{TRS}_{cost4}(c, S_{cluster})$, where $S_{cluster}$ denotes the set of servers in the cluster under consideration. Let $F_{cluster}$ be the set of flows served by at least one server in $S_{cluster}$ and $Y_{cluster}(f)$ be the set of sinks reachable by at least one server in $S_{cluster}$ in downstream direction (i.e., increasing rank direction). The cost functions are defined as

$$\overleftrightarrow{d}^{TRS}_{cost1}(c, S_{cluster}) = -\sum_{\substack{f \in F_{cluster},\\ y \in Y_{cluster}(f)}} \min\left\{0, \overline{d}(f,y) - \hat{d}(c,f,y)\right\}, \quad (6)$$

$$\overleftrightarrow{d}^{TRS}_{cost2}(c, S_{cluster}) = -\sum_{f \in F_{cluster}} \min\left\{0, \min_{y \in Y_{cluster}(f)} \left\{\overline{d}(f,y) - \hat{d}(c,f,y)\right\}\right\}, \quad (7)$$

$$\overleftrightarrow{d}^{TRS}_{cost3}(c, S_{cluster}) = \sum_{\substack{f \in F_{cluster},\\ y \in Y_{cluster}(f)}} \min\left\{0, \overline{d}(f,y) - \hat{d}(c,f,y)\right\}^2, \quad (8)$$

and

$$\overleftrightarrow{d}^{TRS}_{cost4}(c, S_{cluster}) = \sum_{f \in F_{cluster}} \min\left\{0, \min_{y \in Y_{cluster}(f)} \left\{\overline{d}(f,y) - \hat{d}(c,f,y)\right\}\right\}^2. \quad (9)$$

Cost functions $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost2}$ indicate the negative slack (hence, we minimize the negative slack, equivalent to slack maximization) of the involved source-to-sink paths in a linear manner. The difference between the two functions is how multicast flows are treated: Cost function $\overleftrightarrow{d}^{TRS}_{cost1}$ treats the negative slack of different source-to-sink paths individually (hence, the inner summation over all sinks), whereas cost function $\overleftrightarrow{d}^{TRS}_{cost2}$ only considers the worst-case negative slack over all source-to-sink paths of a particular flow (hence, the inner minimization over all sinks instead of summation). It depends on the considered hop which of both approaches is appropriate. For example, consider cluster 2A in Figure 6 and assume a multicast flow f with $x(f) = x_1$ and $Y(f) = \{y_6, y_7\}$ with the shown cluster depth 2. For the downstream hops from s_6 and s_7, the individual treatment appears more appropriate, given that two different underlying physical links are considered. At the downstream hop from s_4, however, there is only one underlying physical link. It appears more appropriate to consider only the worst-case negative slack, given that the assignments in s_4 largely affect both outgoing paths in a symmetric manner (Eq. 5), as well as for hops upstream towards the source. Cost functions $\overleftrightarrow{d}^{TRS}_{cost3}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ are similar to $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost2}$, except that the negative slack is squared to prioritize optimization of larger slack values.

VI. Experimental Evaluation

A. Methodology

The development and performance evaluation of our proposed algorithms was carried out by our tool *UbsConf*. The purpose of *UbsConf* is to (a) generate parameterized random topologies and traffic, (b) develop solver algorithms, and (c) execute series of experiments to evaluate the effectiveness and efficiency of the algorithms. A series comprises multiple test cases for a given set of parameters, where the test cases differ by the randomization seeds. In this section, we summarize the generation steps, describe the generation parameters and introduce the evaluation metrics.

1) Topology Generation: A random topology of servers, sinks, and sources is generated using the parameters $depth$ and $servers$, and an associated random seed. Parameter $depth$ defines the server ranks in the topology (Section II-A), parameter $servers$ defines the number of servers. The servers are (a) randomly distributed over the ranks, (b) connected by randomly chosen links such that no server remains disconnected, and (c) extended by associated sinks and sources, if a server has either no incoming or outgoing links, respectively. Finally, the transmission speed and the number of available queues $|Q(s)|$ is chosen randomly for each server s. The transmission speed is either $100MBit/s$ or $1GBit/s$ with equal probability. The number of queues $|Q(s)|$ is constrained by the parameters $minQueueFactor$ and $maxQueueFactor$ and is selected per server s with

$$minQueueFactor \leq \frac{|Q(s)|}{|S^-(s)|} \leq maxQueueFactor.$$

2) Traffic Generation: Random traffic generation generates random flows with random paths iteratively until all edges are used by a minimum number of flows. This minimum number of flows per edge is constrained by parameter $minFlowPerLink$, and a similar parameter $minFlowPerServer$ defines the minimum number of

flows per server. For each flow f, source $x(f)$ and the set sinks $Y(f)$ are selected randomly. The set $Y(f)$ is constrained by the reachability of the topology and the parameter $maxSinksPerFlow$ such that $1 \leq |Y(f)| \leq maxSinksPerFlow$. The leak rate $\hat{r}(f)$ of each flow f is chosen randomly with $0 < \hat{r}(f) \leq 1$. Once done for all flows, the leak rates are proportionally scaled up such that a configured link utilization of $maxLinkUtilization$ is reached for at least one edge. The packet length $\hat{l}(f)$ is chosen randomly and is constrained by a minimum and maximum value given by the parameters $minPacketLengthPerFlow$ and $maxPacketLengthPerFlow$.

3) Configuration Generation: Since we are interested to generate test cases that are guaranteed to have at least one solution, we generate the deadlines for each flow by constructing a valid random configuration for the topology and the traffic. This configuration exists only temporarily to generate deadlines (as discussed in this section) and is discarded afterwards. The generation algorithm starts with a single priority configuration, which is always a valid configuration (Section IV-B). In an iterative approach, flows sets are consecutively re-assigned to unused and already used queues, and queues are assigned to random priority levels, while always satisfying constraints $CC1$ and $CC3$ to result in a new valid configuration. The number of re-assignment iterations is controlled by the parameter $iterationLimit$. For the final configuration c the end-to-end delay bound $\hat{d}(c, f, y)$ of each flow f to each of its associated sinks $y \in Y(f)$ is computed. The end-to-end deadline $\overline{d}(f, y)$ is then generated by

$$\overline{d}(f, y) = \hat{d}(c, f, y) \cdot niceness Factor,$$

where parameter $nicenessFactor \geq 1$ is used to control the tightness of the deadline constraints of the generated test case. Configuration c is always one solution a solver can discover. The parameter $nicenessFactor$ controls the difficulty. Greater values increase the number of further solutions besides c.

4) Metrics: The two metrics considered for the experiments are (a) the amount of successfully solved test cases out of all finished test cases, and (b) the number of test cases for which a configured timeout expired (i.e., unfinished test cases). The first metric quantifies effectiveness, while the second metric shows efficiency and performance in terms of execution time.

B. Experiments

The solver algorithms with different setups were executed for sixteen series of randomly generated test cases. The underlying parameter setups of *UbsConf* are summarized in Figure 7. Two different classes of topologies were generated: Wide topologies and deep topologies (controlled by parameters $depth$ and $servers$). Wide topologies are constrained to short maximum paths, while the number of servers is large. In average, this results in topologies with larger fan-in for particular server in bridges, considering that each source requires already a dedicated connected server (both placed in a single end station). Deep topologies allow for longer paths, while the average fan-in is lower. For example, a physical chain topology composed of 3-port bridges (2 ports to connect to the chain and 1 port to connect to the end station) would be covered by the given parameter setup. The two classes of topologies were combined with two different traffic setups: unicast flows only and multicast flows with up to four sinks per flow (parameter $maxSinksPerFlow$). 100 random networks were generated for each of the resulting four combinations of topology classes and flow setups. Each of these networks was then combined with four values for $nicenessFactor$.

Parameter	Unicast		Multicast	
	Wide	Deep	Wide	Deep
$depth$	4	8	4	8
$servers$	18	14	18	14
$minQueueFactor$	1			
$maxQueueFactor$	3			
$minFlowPerLink$	3			
$minFlowPerServer$	1			
$maxSinksPerFlow$	1		4	
$minPacketLengthPerFlow$	84 [Byte]			
$maxPacketLengthPerFlow$	1542 [Byte]			
$maxLinkUtilization$	99%			
$iterationLimit$	20			
$nicenessFactor$	$\{1.0, 1.1, 1.2, 1.3\}$			

Figure 7. Setup of unicast and multicast experiments with wide and deep topologies, combined with different values for $nicenessFactor$.

All experiments were executed on a dual processor AMD Opteron 6376 server with 128 GByte RAM, where one solver instance used at most one of the 32 overall available processor cores. A timeout of 30 minutes was used to terminate solver instances in case the respective instance did not terminate either by providing a solution or failing without a solution.

C. Results

The results of the experiments are shown in Figure 8, which is organized in a two-row, eight-column matrix of charts. There is one column for each combination of (a) deep vs. wide topologies, (b) multicast vs. unicast flows, and (c) integer-typed vs. real-typed solver setup. Each chart shows the performance of the pure SMTS solver, as well as TRS with cluster depths 1 and 2 and with the four cost functions in Equations 6–9. As an exception for unicast flows, cost functions $\overleftrightarrow{d}^{TRS}_{cost2}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ are omitted from the respective charts, as the functions are identical to $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost3}$, respectively, in absence of multicast flows. Each chart in the bottom row shows the number of test cases per solver for which the timeout expired, with increasing values of $nicenessFactor$ on the horizontal axis. Each chart in the top row shows the percentage of *solved* test cases (i.e., a solution was found) out of all *finished* test cases (i.e., solver terminated without a timeout). Note that SMTS performed at 100 percent. Except due to rounding errors to integer data rates (not observable in Fig. 8), this is ensured by definition.

As can be seen in Figure 8, for both linear cost functions $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost2}$, TRS outperforms SMTS in terms of speed. The number of timeouts is significantly different,

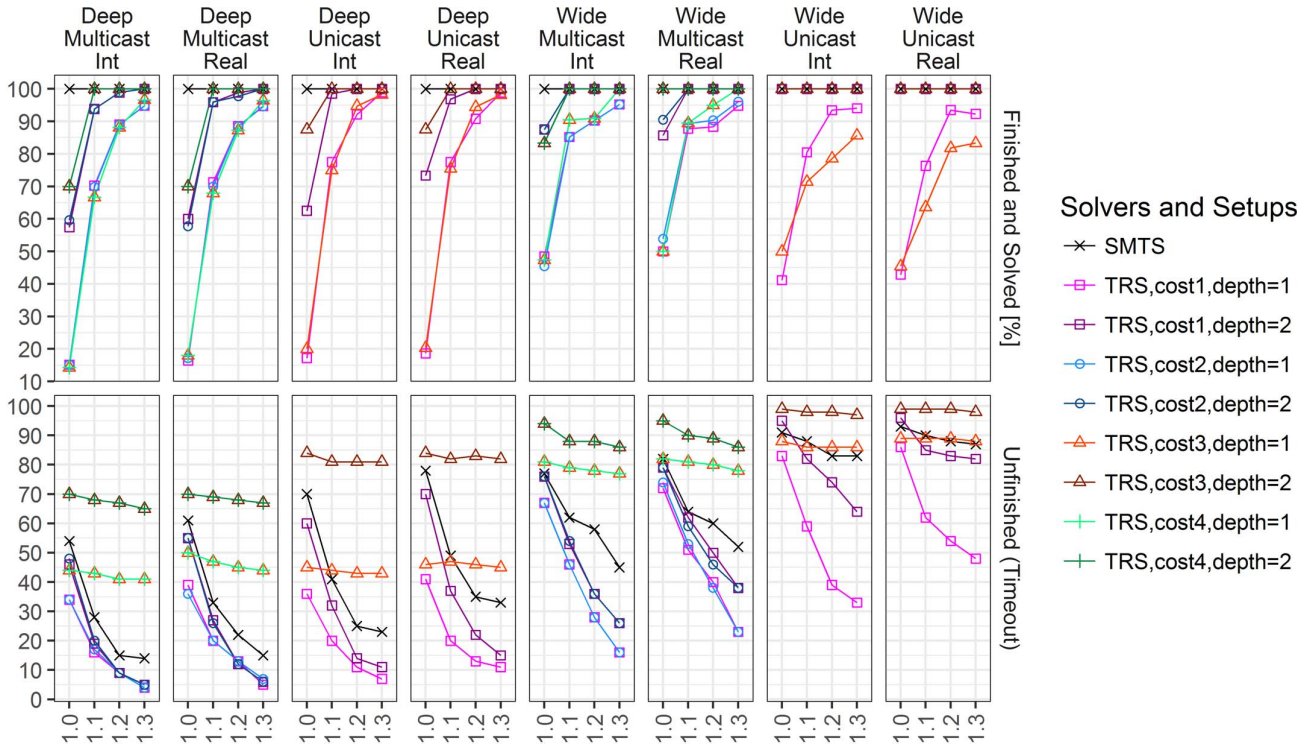


Figure 8. Results of all experiments (best viewed in color). The bottom row, consisting of 8 charts (each a combination of deep vs. wide topologies, multicast vs. unicast flows, and integer-typed vs. real-typed variables in the solver), indicate on the vertical axis the amount of test cases for which a timeout expired and on the horizontal axis increasing values of $niceness Factor$. Charts in the top row show the percentage of successfully solved test cases out of all *finished* test cases (i.e., for which the timeout did not expire) for increasing values of $niceness Factor$. The unicast results for TRS with cost functions $\overleftrightarrow{d}^{TRS}_{cost2}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ are omitted, given that these functions are in such cases identical to $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost3}$, respectively.

especially for $niceness Factor = 1.3$. Here, SMTS experienced up to approximately three times more timeouts than TRS with linear cost functions, as visible for deep topology experiments. For example, in the bottom left chart (deep topologies, multicast flows, and integer-typed data rates) for $niceness Factor = 1.0$, we can observe that SMTS times out in around 55 percent of the cases, while the TRS timeout rate for either cluster depth is around 35 percent. For the same chart, the timeout rates of SMTS and TRS, respectively, are around 15 and 5 percent for $niceness Factor = 1.3$. The speed difference between cluster depth 1 and cluster depth 2 appears insignificant for these topologies, in contrast to the wide topology experiments. For the latter, the number of timeouts with linear cost functions differs significantly, especially for the results with unicast traffic. For example, considering the bottom right chart (wide topologies, unicast flows, and real-typed data rates) and $niceness Factor = 1.3$, the timeout rate of TRS with cost function $\overleftrightarrow{d}^{TRS}_{cost1}$ is around 80 percent for cluster depth 2 and around 50 percent with cluster depth 1. Contrary, TRS becomes even slower than SMTS with square cost functions $\overleftrightarrow{d}^{TRS}_{cost3}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ in most situations. In cases where TRS with square cost functions is faster than SMTS (e.g., deep topologies with unicast traffic and $niceness Factor = 1.0$), the success rate is low and the difference in results is insignificant compared to TRS with linear cost functions, although we highlight performance improvements for the square functions later in this section.

The TRS cluster depth has a significant positive impact on the success rate. Considering only the linear cost functions $\overleftrightarrow{d}^{TRS}_{cost1}$ and $\overleftrightarrow{d}^{TRS}_{cost2}$, the number of failed test cases converges quickly to zero with cluster depth 2, while still being faster than SMTS, but likewise slower than the respective setups with cluster depth 1. For example, considering the top row and the chart for deep topologies, unicast flows, and integer-typed data rates, the success rate converges to 100% already for $niceness Factor = 1.1$ for TRS with depth 2 and $\overleftrightarrow{d}^{TRS}_{cost1}$, while, as shown in the corresponding bottom row chart, it times out less frequently than SMTS. As expected, the success rate of TRS increases with increasing cluster depths, given that the combinatorial space converges towards the one of SMTS with increasing cluster depth. Note that the success rate of TRS with linear cost functions and a small cluster depth 1 is already significant. For example, the success rate for these TRS setups is already between 70 and 90 percent for $niceness Factor = 1.1$, which represents test cases with very tight deadline constraints (i.e., only 10 percent extra on the achievable end-to-end deadlines).

While TRS setups with cluster depth 2 failed less times than those with depth 1, there are small cost function dependent differences for the depth 2. For deep topology experiments and $niceness Factor = 1.0$, the success rate of the square

cost functions $\overleftrightarrow{d}^{TRS}_{cost3}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ is higher than the success rate of the linear cost functions.

The difference between integer and real typed data rates appears symmetric throughout the experiments. A small but notable difference from this was measured for wide topologies with unicast traffic. One can observe a more significant improvement for integer data rates for the linear TRS cost function $\overleftrightarrow{d}^{TRS}_{cost1}$ for depth 1 as well as depth 2. Moreover, it appears cost function $\overleftrightarrow{d}^{TRS}_{cost1}$ returned more solutions than cost functions $\overleftrightarrow{d}^{TRS}_{cost3}$ for wide topologies with unicast traffic, contrary to all other series, where the square cost functions $\overleftrightarrow{d}^{TRS}_{cost3}$ and $\overleftrightarrow{d}^{TRS}_{cost4}$ were notably closer to the linear ones.

To summarize, TRS executed with the linear cost functions provides significant performance improvements compared to SMTS while keeping a high success rate in synthesizing network configurations that meet tight real-time constraints.

VII. Conclusions

Ethernet is an increasingly common communication technology in many embedded real-time systems due to its scalability and increasing bandwidth requirements in applications areas like automotive electronics, avionics, and industrial control. We previously presented Urgency Based Scheduler (UBS), an asynchronous real-time traffic scheduling policy that is currently under standardization (P802.1Qcr). In this paper, we introduced solving strategies for the synthesis problem of determining flow-to-queue and queue-to-priority level assignments for hard real-time Ethernet networks with UBS. We presented an SMT-based solution and a heuristic based on topology ranks. Our experimental results demonstrate the effectiveness of an SMT solution and the efficiency of our TRS heuristic.

References

[1] LAN/MAN Standards Committee of the IEEE Computer Society, *IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks Amendment 25 : Enhancements for Scheduled Traffic, IEEE Std. 802.1Qbv-2015*. 2015.

[2] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Time-Triggered Ethernet : TTEthernet," *Time-Triggered Communication*, 2011.

[3] J. Specht and S. Samii, "Urgency-based Scheduler for Time-Sensitive Networks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[4] IEEE, "IEEE Standard for Local and Metropolitan Area Networks – Audio Video Bridging (AVB) Systems, IEEE Std. 802.1BA-2011," 2011.

[5] IEEE, "Official Project Website of 802.1Qcr - Asynchronous Traffic Shaping," 2016.

[6] R. L. Cruz, "A calculus for network delay–I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, 1991.

[7] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks : The Single-Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[8] IEEE, "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridge Networks, IEEE Std. 802.1Q-2014," 2014.

[9] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, 1990.

[10] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," *INFOCOM '93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future*, vol. 1, pp. 227–236, 1993.

[11] T. Hamza, J. L. Scharbarg, and C. Fraboul, "Priority assignment on an avionics switched Ethernet Network (QoS AFDX)," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2014.

[12] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," no. YCS-164, p. 31, 1991.

[13] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, 2001.

[14] J. Garcia and M. Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," *Proceedings of Third Workshop on Parallel and Distributed Real-Time Systems*, pp. 124–132, 1995.

[15] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[16] International Organization for Standardization, "ISO 11898: Road vehicles – Controller area network (CAN)," 2003.

[17] L. Zhang, M. Lampe, and Z. Wang, "Topological design of industrial ethernet networks with a fast heuristic," *15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010*, 2010.

[18] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and Network-level Schedule Co-Synthesis of Ethernet-based Time-triggered Systems," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 119–124, 2014.

[19] S. Beji, S. Hamadou, A. Gherbi, and J. Mullins, "SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures," *IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT*, pp. 165–174, 2014.

[20] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Real-Time Systems Symposium*, pp. 375–384, 2010.

[21] S. S. Craciunas, R. S. Oliver, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802 . 1Qbv Time Sensitive Networks," in *24th International Conference on Real-Time Networks and Systems (RTNS)*, 2016.

[22] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT message scheduling with temporal constraints," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 369–380, 2010.

[23] D. Tamas-Selicean and P. Pop, "Optimization of TTEthernet networks to support best-effort traffic," *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, pp. 1–4, 2014.

[24] W. Steiner, "Synthesis of Static Communication Schedules for TTEthernet-Based Mixed-Criticality Systems," pp. 11–18, 2012.

[25] G. Avni, "Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links," 2016.

[26] D. Tamas-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Systems*, vol. 51, no. 1, pp. 1–35, 2014.

[27] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[28] V. Gavrilut and P. Pop, "Traffic class assignment for mixed-criticality frames in TTEthernet," *ACM SIGBED Review*, vol. 13, no. 4, pp. 31–36, 2016.

[29] I. Working Group 802.1, "Project Authorization Request (PAR) of P802.1Qcr (approved June 30, 2016)," 2016.

[30] N. Bjorner and L. de Moura, "Z3: An efficient SMT solver," in *TACAS (2008)*, vol. 4963 LNCS, pp. 337–340, Springer-Verlag, 2008.

[31] B. Dutertre, "Yices 2.2," in *CAV'14*, pp. 737–744, Springer-Verlag, 2014.

[32] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovi, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *CAV'11*, pp. 171–177, Springer-Verlag, 2011.

[33] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB Standard Version 2.5," p. 85, 2015.

[34] S. Conchon, D. Déharbe, M. Heizmann, and T. Weber, "The 11th International Satisfiability Modulo Theories Competition (SMT-COMP 2016)," 2016.

[35] N. Bjorner, A.-D. Phan, and L. Fleckenstein, "vZ - An Optimizing SMT Solver," in *TACAS*, pp. 194–199, Springer-Verlag, 2015.