Incremental Path-Selection and Scheduling for Time-Sensitive Networks

Abdullah Alnajim Seyedmohammad Salehi Chien-Chung Shen
Department of Computer and Information Sciences, University of Delaware, USA
{alnajim,salehi,cshen}@udel.edu

***Abstract*—Industrial real-time applications demand their communication networks be robust and deterministic so that packets are delivered with bounded delay and jitter. The IEEE Time-Sensitive Networking (TSN) Task Group has amended the standards of IEEE 802.3 Ethernet to support time-triggered flows of real-time applications. However, the issues of routing and scheduling are left as open problems. Specifically, majority of the state-of-the-art scheduling efforts make certain assumptions, such as assuming that the routing information is known *a priori* to focus on the scheduling problem or assuming that the information of flows is known in advance. Other studies that avoided these assumptions ignore certain valuable information that could be taken advantage of to make better routing decisions, such as the residual bandwidth. In this paper, we first propose an incremental QoS-aware path selection algorithm that uses QoS measurement to route TSN flows. Then, we introduce multiple incremental scheduling algorithms that address above mentioned issues, and demonstrate their performance.**

***Index Terms*—time sensitive networks, path selection, routing**

I. Introduction

Time-Sensitive Networking (TSN) is a set of standards developed by the Time-Sensitive Networking Task Group (TSN TG) of the IEEE 802.1 Working Group for real-time communication over Ethernet networks with bounded delay and jitter [5]. TSN brings industrial-grade robustness to Ethernet to facilitate real-time, safety critical applications (e.g., Industry 4.0 and Cyber-Physical Systems) and to enable the transmissions of both time-critical and best-effort traffic over one common communication infrastructure.

Majority of the existing routing and scheduling algorithms for TSN are executed offline for static scenarios, where the routes or schedules are computed at the design time with *a priori* knowledge of either flow or routing information, which remain static while the network is operating online. Although such scenarios facilitate optimal or near optimal solutions, they become impractical with realistic deployments. For instance, in the context of Industrial Internet of Things (IIoT), it is inevitable to add and remove flows and/or devices while the system is running. Moreover, given that offline solutions are computed at the design time, their actual execution time is not a major concern.

By exploiting the logically centralized paradigm of software-defined networking (SDN), [3] is the first effort that formulated the integrated problem of routing and scheduling in Integer Linear Program (ILP) given a set of pre-defined time-triggered flows. Subsequently, the same authors proposed *incremental* algorithms [4] to dynamically schedule new flows whenever they become available. However, since they did not use the gating with Guard Band mechanism introduced in IEEE 802.1Qbv time-aware scheduler, best-effort traffic may cause real-time traffic to wait at egress ports which negatively affects latency and jitter. In addition, their use of large time-slots reduces the bandwidth available for the high priority streams. In addition, their approaches do not consider the variety of flow requirements; instead, they try to bound the end-to-end delay and jitter. In practice, some time-critical applications may have more stringent delay requirements than the others; therefore, considering the flow requirements to find the solution space is valuable.

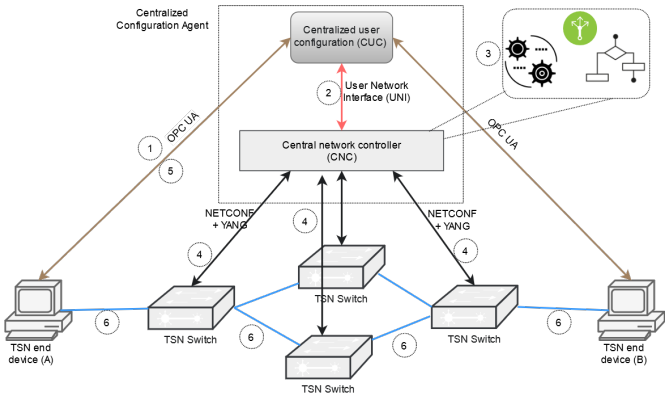


Fig. 1: TSN architecture and workflow of connection setup.

In this paper, we propose one QoS-aware path-selection algorithm and multiple scheduling algorithms to route and schedule TSN flows *incrementally*, and demonstrate their performance.

II. Architecture and Connection Workflow

Fig. 1 depicts the architecture of a TSN network. The workflow of connection setup works as follows. When end device A wants to send a TSN flow to device B, (1) it sends a TSN connection request to Centralized user configuration (CUC) using the OPC Unified Architecture (OPC UA) protocol; (2) CUC converts the TSN connection request into TSN connection requirements and forwards them to Central network controller (CNC) through user network interface (UNI); (3) CNC uses the proposed incremental path-selection and scheduling algorithms to compute both a path that satisfies the connection requirements and a feasible transmission schedule to avoid interfering with existing TSN flows; (4) If such a path and schedule exist, CNC configures all the TSN switches along the computed path with the computed schedule; (5) CNC sends the transmission schedule to CUC,

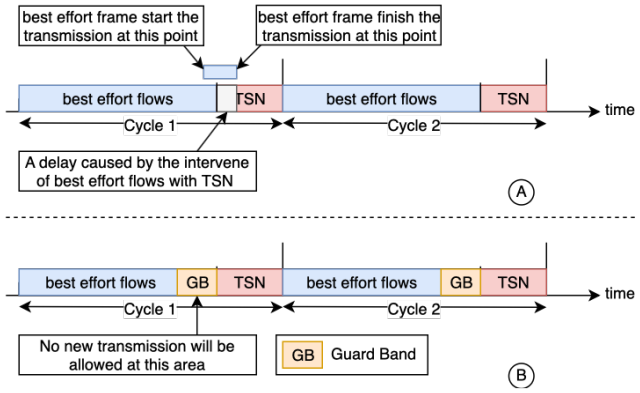


Fig. 2: Cyclic schedule of IEEE 802.1Qbv: (A) without and (B) with Guard Band (GB)

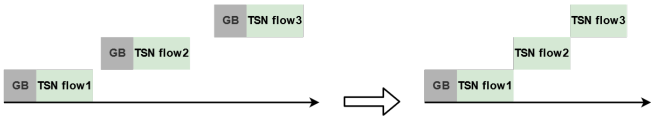


Fig. 3: The scheduling solution of Guard Bands issue

which forwards it to device A; (6) device A starts transmitting the TSN flow to device B along the selected path at the computed transmission schedule.

In Step (3) above, when CNC receives the connection requirements from CUC, it will use them to select a set of candidate paths between the source and destination that satisfy these requirements, from a set of k precomputed paths. If the candidate path set is empty, the process terminates, and the connection request is rejected. Otherwise, CNC will use the current QoS measurements to choose a path with the highest QoS compound value out of the set of candidate paths. Then, the flow's requirements and the chosen path will be sent to the scheduler. If the scheduler is able to schedule the flow, CNC will configure all the switches along the path and send the transmission start time to the sender. Otherwise, the next best path will be selected and the process repeats.

Inside each TSN switch, there exist additional mechanisms to isolate best-effort traffic from TSN flows. Based on IEEE 802.1Qbv (time-aware scheduler), in front of each egress port queue, there is a gate that controls the transmission from the queue. A queue can transmit only if its gate is open, and if there is more than one queue with an open gate, the one with the highest priority will transmit. From a queue with an open gate, choosing which flow to transmit is based on the Transmission Selection Algorithm (TSA). The open and close of gates are controlled by a Gate Control List (GCL) of multiple entries, where each entry consists of a timestamp and a bit-mask field controlling the open and close of gates.

The length of GCL is equal to the base-period of a cycle, which will be repeated to generate a cyclic schedule shown in Fig. 2(A). In this figure, however, even though gates are used, at Cycle 1, a best-effort frame may start transmission just before its gate is scheduled to be closed, so that it may continue transmitting even when the TSN flow period has started, which causes a delay for the scheduled TSN flow(s). To address this issue, Guard Bands (GBs) are used before each TSN gate opening event as shown in Fig. 2(B). Although GBs help to isolate TSN traffic from best-effort traffic, when a best-effort frame finished transmission just after a GB has started, the rest of the GB would be wasted. To mitigate this issue, there is the hardware-based frame preemption mechanism detailed in IEEE 802.1Qbu. Another solution is a scheduling-based method proposed by [2], which tries to reduce the number of GBs by reducing the number of TSN gate opening events. This method does so by sending as many TSN flows as possible whenever the gate is open. The resulted schedule is similar to the right side of Fig. 3 compared to the left side of the same figure.

III. QoS-Aware Path-Selection

A. Problem Formulation

The main objective of QoS-aware path selection is to incrementally find an appropriate route for a TSN flow based on the flow requirements and a set of QoS matrices. Therefore, the factors that need to be considered by this problem are the network topology, the set of TSN flows, and the set of QoS measurements.

The network topology of a TSN is represented as a direct graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. $V \equiv (S \cup H)$, where S is the set of TSN switches, and H is the set of end hosts. E is a set of 2-tuples that represent the links in the network, such that $E \equiv \{(i, j) \mid i, j \in V, i \neq j$ and there is a link between i and $j\}$. Each TSN flow F is represented as a 3-tuple $F \equiv (s, d, fd)$, where $s \in H$ is the flow source, $d \in H$ is the flow destination, and $fd \in \mathbb{R}^+$ is the maximum end-to-end delay in μsec.

Associated with each link $(i, j) \in E$ is a list of measurements represented by tuple (b, ld, t), where $b \in \mathbb{R}^+$ is the residual bandwidth of link (i, j) in $Mbps$, $ld \in \mathbb{R}^+$ is the link delay in μsec which is composed of the processing delay in i, the transmission delay of i, and the propagation delay of the link (i, j), $t \in \mathbb{Z}^+$ is the number of TSN flows traversing link (i, j). Note that, ld is bounded [3], and the main goal of the scheduling problem in Section IV is to avoid the queuing delay or make it predictable. Moreover, the transmission delay is the amount of time required to push an MTU-sized packet into the wire by node i. The QoS measurements are updated both periodically, and before and after routing a flow, to reflect the up-to-date status of the network. For instance, after routing a flow through a path, the t value of all links belonging to that path will be incremented by one.

A cycle-free path p between two distinct nodes is a finite sequence of nodes $p \equiv < v_0, v_1, ..., v_n >$, such that $\forall i \in [0, n-1] \Rightarrow (v_i, v_{i+1}) \in E$. A path $p(i, j)$ between i and j is valid if the length of the path is less than or equal to eight hops (as recommended by IEEE 802.1D) and all these nodes are distinct; i.e., it is a cycle-free path with a length less than eight hops. We denote the set of all valid paths between i and j as $\wp(i, j)$. Each path $p(i, j)$ has its own QoS indicators (HC, B, D, T) that could be derived from its links' QoS indicators, where $HC \in \mathbb{Z}^+$ represents the number of hops between i and j if a flow goes through this

path, $B \in \mathbb{R}^+$ denotes the residual bandwidth of the path $p(i,j)$ in $Mbps$, $D \in \mathbb{R}^+$ is the end-to-end delay of $p(i,j)$ in μsec, and $T \in \mathbb{Z}^+$ is the number of TSN flows over $p(i,j)$. We follow the formulas of [6], with certain modifications, to derive B and D from the measurements of the links that compose path p. The number of hops HC of path p is equal to the number of the intermediate nodes, which equals the nodes of path p except the source and the destination, which can be formally defined as follows.

$$p.HC = len(p) - 2 \tag{1}$$

The residual bandwidth B of path p is equal to the minimum residual bandwidth among all of its component links, and is formally defined as follows.

$$p.B = min_{i=1}^{len(p)-3}(p[i], p[i+1]).b \tag{2}$$

The end-to-end delay D of path p is equal to the summation of the delays of its links, and it can be formalized as follows.

$$p.D = \sum_{i=0}^{len(p)-2} (p[i], p[i+1]).ld \tag{3}$$

Finally, the number of TSN flows T in path p is equal to the maximum number of TSN flows t among all of its links:

$$p.T = max_{i=1}^{len(p)-3}(p[i], p[i+1]).t \tag{4}$$

For a flow f with a source s and a destination d, the value of D is used to choose a set of candidate paths $\psi(s,d)$ that satisfy flow f's requirements out of a set of all valid paths $\wp(s,d)$. Then, for each path $p_i(s,d) \in \psi(s,d)$, where $i \in (1, 2, \cdots, |\psi|)$, function $\Delta(HC, B, T)$ is used to map the other QoS indicators to a single value between zero and one, $\Delta : (\mathbb{Z}^+, \mathbb{R}^+, \mathbb{Z}^+) \rightarrow [0, 1]$. Out of all the candidate paths $\psi(s,d)$, the path with the highest Δ is chosen as a route for f. For a path $p_i(s,d) \in \psi(s,d)$, Δ is formulated as follows.

$$p_i(s,d).\Delta(HC, B, T) = \omega_1 \frac{HC_{min}}{HC} + \omega_2 \frac{B}{B_{max}} + \omega_3 F(T), \tag{5}$$

where $\sum_{j=1}^{3} \omega_j = 1$; B_{max} is the maximum B among all $p_i(s,d) \in \psi(s,d)$; HC_{min} is the minimum HC among all $p_i(s,d) \in \psi(s,d)$. Semantically, maximizing Δ means minimizing the number of hops, minimizing the number of TSN flows, and maximizing the residual bandwidth. Function $F(T)$ is defined as follows.

$$F(T) = \begin{cases} 1, & \text{if T = 0} \\ \frac{T_{min}}{T}, & \text{if T} > 0 \end{cases} \tag{6}$$

where T_{min} is the minimum T among all $p_i(s,d) \in \psi(s,d)$.

Other QoS measurements could be added into this formula, such as the path cost in the DetNet networks and the end-to-end delay D. We choose not to include D because we used it for filtering in the candidate-path selection phase. Mainly, the route with a higher B, lower HC, and lower T that satisfies the flow requirements would be chosen. The main goal of Δ is to find such a path. Choosing a path with a higher B means sparing other paths having lower residual bandwidth for future TSN and best-effort flows. Since the gate mechanism prevents best-effort traffic from transmitting when a TSN flow arrives, routing many TSN flows through an overloaded link will degrade the performance of best-effort traffic, where some of the packets of these streams may be delayed or dropped before they are re-routed through another path. Therefore, to mitigate this issue, a path with a congested link should be avoided, whenever it is possible, to avoid creating a bottleneck link in terms of bandwidth. Taking the ratio of $p_i.B$ to the highest B among all candidate paths $\psi(s,d)$ normalizes the value between zero and one. Otherwise, it would have the most significant effect out of all the other indicators and negatively affect the performance.

B. Pre-Routing and Candidate-Path Selection Phases

The primary purpose of the pre-routing phase is to find the first k paths $\Phi(s,d)$ between s and d, for all $s, d \in H$ and $s \neq d$ ordered ascendingly based on their end-to-end delays D. To find these paths, there are several existing search algorithms, such as the Yen's K-shortest paths algorithm. Then, the first k paths $\Phi(s,d)$ are passed to the next phase as input. If the k value is small, the running time of the routing phase will be small. However, the performance of the entire operation might be compromised. So, choosing the k value is a trade-off between running time and performance. The inputs of this phase are the network topology and k, and the outputs are sets of ordered k paths $\Phi(s,d)$. The worst-case running time of this phase is $O(SKV^3)$, where S is the number of TSN end devices, K is the number of chosen paths, and V is the number of nodes. Although the running time is relatively large when the size of the network is extremely large, it will not cause a problem for the system since it will take place while the network is off-line. In other words, this phase is done once, before starting to exchange flows, and remains unchanged while the network is online. That is, no change in the topology will be allowed until the network becomes off-line. Otherwise, it has to be re-executed again. When a new end device is connected to the TSN network, only the first K paths to/from this device is calculated leading to a running time equal to $O(KV^3)$. Moreover, parallelism and multithreading could be used to reduce the running time of this phase by making each thread to find the K paths of a subset $\overline{S} \subset S$. In addition, using the Fibonacci heap data structure along with Dijkstra's shortest path algorithm to implement Yen's K-shortest paths algorithm will reduce the running time of this phase to $O(\overline{S}KV(E + VlogV))$, where $\overline{S}$ is the number of elements in the subset handled by each thread, and E is the number of links in the network. Finally, even without these optimization steps, the running time of this phase is in order of seconds for TSN networks with practical sizes, as we will see in the evaluation.

C. Candidate-Path and Path-Selection Phase

Each time CNC receives a TSN connection requirements from CUC, it uses s and d to retrieve the set $\Phi(s,d)$ of k paths that connect s and d. Then, the Candidate-Path Selection Phase is used to extract a set of candidate paths $\psi(s,d)$ out of $\Phi(s,d)$ that satisfy the flow delay requirement fd.

Since the problem of finding a path that satisfies constraints involving two or more additive or multiplicative metrices had been proven to be *NP*-complete [6], for an algorithm to find such a path in polynomial time, we need to convert all these metrices into a single value (Δ). After receiving a TSN flow (f) as an input, this phase calculates the best route, based on Δ, and returns it as an output. The worst case running time for this phase is $O(K)$, where K is the number of paths computed in Section III-B. The resulted route, in addition to the TSN flow (f), will then be sent to the scheduler to determine the appropriate transmission time.

IV. Incremental Scheduling

The primary goal of the scheduling algorithm is to ensure that a new flow f will not intervene, i.e., share the same egress port at the same time, with any existing flows. Moreover, while scheduling f, the scheduler should not make any modification to the existing, scheduled flows. Since TSN flows send their data periodically, the computed schedule will be cyclical. Following [2] and [3], we assume that the transmission periods for all flows are integral multiples of the cycle length. In static scheduling where the information of the flows is known *a priori*, this assumption could be relaxed by making the cycle length of the schedule to be equal to the least common multiple (LCM) of all the flows' transmission periods. However, in incremental scheduling, this solution cannot be applied because the new flows' information is not known *a priori*.

The computed schedule will be a list of flows and their start times, so that the gate opening times will be computed based on such information. For instance, the gate opening time for the TSN queue in egress port p_i of switch sw_j, in which TSN flow f will go through, is the additive value of the propagation delays of all the links in the selected path that proceed sw_j, the transmission delays of the network interface card (NIC) of the source and all the switches in the chosen path that proceed sw_j, and the processing delays of all the switches in the path up to sw_j.

A. Scheduling Without Time-Slots - As Early As Possible (SWOTS-AEAP)

In this approach, we adapt the time-tabling problem [2] to schedule flows incrementally. The inputs of this approach are the new flow (f) and the schedule (s) which is a list of scheduled flows and their transmission start times. The output is a boolean value that determines whether the flow is scheduled successfully or not. Specifically, in this approach, we set the transmission start time of the new flow to be zero, and then increase it to make its arrival time at the egress ports just after the existing overlapped scheduled flows finish their transmissions.

The worst case running time to schedule one flow is $O(KNM)$, where K is the number of transmission operations of the new flow, N is the number of scheduled flows, and M is the average number of transmission operations for each scheduled flow. Since the number of operations for each flow is ≤ 16, i.e. bounded, the resulting running time is $O(N)$.

B. Scheduling Without Time-Slots - As Soon As Possible (SWOTS-ASAP)

In comparison to *SWOTS-AEAP*, which schedules the new flow with respect to the beginning of each cycle, *SWOTS-ASAP* schedules the new flow based on its arrival time. With this flexibility, *SWOTS-ASAP* could be adapted easily to schedule flows in a specific time required by their applications by changing the *startTime* variable to indicate that specific instance. In addition, since *SWOTS-ASAP* allows a new flow to be scheduled before some of the scheduled flows that have overlapping transmission operations, it outperforms *SWOTS-AEAP* in terms of the number of scheduled flows, but with longer running time.

C. Scheduling With Time-Slots (SWTS)

This approach divides the cycle into multiple time-slots similar to the one introduced in [4]. Each time-slot is long enough to safely transmit an MTU-sized frame through the longest path, in terms of end-to-end delay, in the network. Each time-slot has an unique ID and a list of $|E|$ boolean values indexed by i. The value of index i indicates whether any of the assigned flows will traverse link i. The new flow will be assigned to the soonest time-slot that does not have any scheduled flow which shares links with this new flow. Clearly, *SWTS* is less complicated than the *SWOTS* approaches, and incurs less computation time with the worst case running time of $O(S)$, where S is the number of time-slots. However, the performance of *SWOTS* approaches, in terms of the number of TSN flows that could be scheduled, is always better than *SWTS*. So, there is a trade-off between speed and performance.

D. SWOTS With Stop (SWOTS-WS)

The *SWOTS* approaches are designed to eliminate queuing delay completely. Allowing queuing delays, as long as the flows' maximum delay requirements are satisfied, helps to improve the schedulability in terms of the number of TSN flows that could be scheduled. Therefore, we designed modified versions of *SWOTS*, termed *SWOTS-AEAP-WS* and *SWOTS-ASAP-WS*.

In these new approaches, instead of shifting the start time whenever an overlap occurs, they keep track of these shifts by adding them to a list of queuing delays incurred at switches for each flow. When the sum of these queuing delays, in addition to the cumulative delay of the last operation of the new flow exceeds the maximum end-to-end delay, the *startTime* will be shifted to reduce the queuing delays. When the new flow has been scheduled, the new schedule and the associated per hop queuing delays of that flow are returned. With this relaxation, the *SWOTS-WS* schemes will be able to schedule more TSN flows without violating their end-to-end delay requirements.

V. Evaluation

The proposed algorithms are evaluated in terms of performance and running time, and in terms of scalability with different network sizes and different numbers of TSN flows. The TSN flows and their requirements are generated

randomly. Among the system configurations discussed in [1], our systems assume to be $\{v_{(e+s)}, \langle 8, 1, 7\rangle\}$, depicting a fully scheduled system (end devices and switches) with eight queues in each egress port (one for TSN flows, and the others for best-effort traffic).

A. Performance

Fig. 4a compares the scheduling algorithms in terms of the percentage of scheduled flows out of routed flows. The parameters used are as follows: the number of selected paths by the pre-routing phase (k) 30, the number of switches (n) 20, the probability of having a link between any two nodes (p) 0.3, the number of time-slots 5. Furthermore, we used equal weights for the QoS parameters in the routing algorithm. Then, by varying the number of TSN flows from 100 to 1000 with an increment of 100, the average results of five runs are shown. In the case of 1000 TSN flows, *SWOTS-ASAP* and *SWOTS-ASAP-WS* yielded the best results, capable of scheduling five times more TSN flows than *SWTS* and almost two times more than *SWOTS-AEAP*. Moreover, allowing a predictable queuing delay improves *SWOTS-AEAP* in some cases by 41.3%

Furthermore, we measured the percentage of reduction in the number of GBs required by the computed schedules of all the *SWOTS* algorithms against the total number of GBs required by schedules that do not allow sending more than one TSN flow consecutively in an egress port, such as the *SWTS* approach. As shown in [2], reducing the number of TSN gate opening events reduces the number of required GBs by the scheduler, which mitigates the wasted bandwidth issue resulting from GBs. We fixed the parameters to values similar to the ones mentioned above and varied the number of TSN flows between 800 and 1500. The percentages of the reduced amount for all the *SWOTS* approaches are shown in Figure 4b. With the increase of the number of TSN flows, there is an increase in the percentage of reduction for both *SWOTS-ASAP* and *SWOTS-ASAP-WS*. The best performance in terms of the reduced GBs is achieved by *SWOTS-AEAP-WS*, in which it needed 63% less GBs.

To show the effect of weights in Eq. (5), we tested different combinations of weights defined as a 3-tuple (x, y, z), where x represents the weight of the number of hops, y represents the weight of TSN flows, and z represents the weight of the residual bandwidth. Fig. 4c shows the percentage of the scheduled flows out of the total routed flows for all the scheduling algorithms using different combinations of weights. Based on the results, (0.5, 0.5, 0) leads to the highest percentage for all schedules, out of the tested combinations, with an improvement of at least 5% over the combination of (1, 0, 0) which represents the shortest path. Finding an optimal combination that maximizes the number of the scheduled TSN flows and minimizes the impact on best-effort traffics is beyond the scope of this paper. Finally, we measured the time a TSN sender has to wait before transmission, starting from the time it receives the schedule. On average, a TSN end device has to wait 13 and 2.4 μsecs, respectively, when *SWOTS-ASAP* and *SWOTS-ASAP-WS* schedulers are used, compared to 1.5, 7.5 and 8.4 milliseconds when *SWTS*, *SWOTS-AEAP* and *SWOTS-AEAP-WS* are used, respectively. The main reason is that *SWOTS-AEAP* and *SWOTS-AEAP-WS* schedule the arrived flows at the beginning of the cycles whenever it is possible, regardless of their arrival time. On the other hand, the flows that arrived in the middle of a time-slot in *SWTS* have to be scheduled in the next time-slot, whenever it is possible, and the senders have to wait until then.

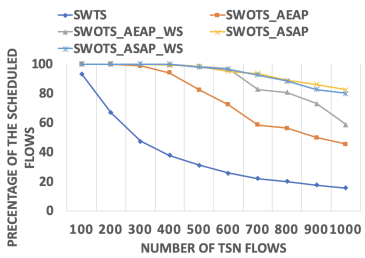
B. Running time

For each algorithm, we measured the average running time of different scenarios by varying the system parameters (e.g., number of time-slots and k values).

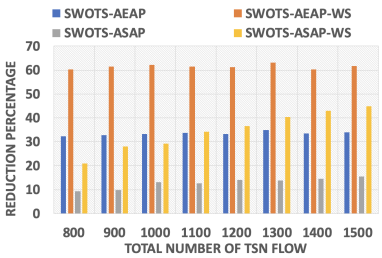
First, we tried different numbers of TSN flows between 100 and 1000, and measured the average running time to route and schedule a flow using different scheduling algorithms. The computed running times for all the algorithms are shown in Fig. 4d. Overall, as expected, only the *SWOTS* scheduling algorithms are affected by the increase of the number of TSN flows, since the increase in the number of flows means an increase in the number of the scheduled flows. The running time of *ASAP-based* algorithms increased rapidly with a highest running time of 165 *milliseconds*, while the running time of *AEAP-based* algorithms increased linearly with a highest running time of 23.5 *milliseconds*. This increase, in both cases, will continue until the schedule saturated for a long time, then, it will become constant. The routing and *SWTS* are not affected in any way by the increase in the number of TSN flows, and their running times are in orders of hundreds and tens of μsecs, respectively. The average running times of the pre-routing are measured in multiple runs, and they were in the orders of multiple seconds without any effect from the increase in the number of TSN flows.

SWTS and the routing algorithms are affected by the number of time-slots and the k values, respectively. To measure the effect of an increase in the number of time-slots on the average running time of *SWTS* , we varied the number of time-slots between 5 and 70 with an increment of 5 time-slots. We noticed that with an increment of 5 time-slots, there was an increase between 5 to 10 μsecs on average in the running time of *SWTS*, and that was due to traversing all the time-slots looking for any available one when the schedule is saturated. The routing algorithm, on the other hand, is affected by the k value of the pre-routing phase as discussed in Section III-C. To measure this effect, we varied the k value between 5 and 70 with an increment of 5. The results show an increase in the running time of the routing algorithm by 80 μsecs on average, for each increment by 5 on the k value.

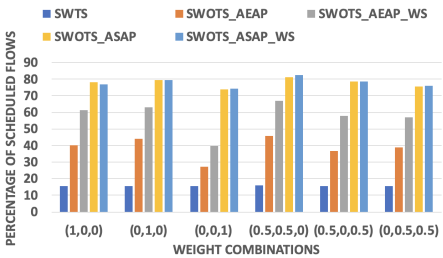
Based on the theoretical analysis of the worst case running time for the pre-routing phase in Section III-B, the main parameters that affect the running time of the pre-routing phase are: the k value, the number of switches, the number of hosts, and the number of links. We evaluated the effect of each one by varying one value while fixing the values of the other parameters. Fig. 4e shows the relationship between the increase in the average running time of the pre-routing phase, and the increase of these parameters' values. Each run in the graph corresponds to an increment of 5 in k value, number of switches, and number of hosts, and an increment of 0.1



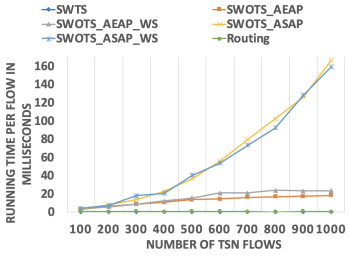
(a) Relationship between the number of TSN flows and the percentage of scheduled flows



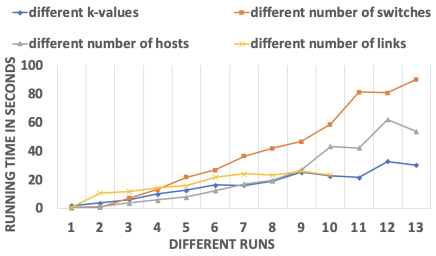
(b) Relationship between the number of TSN flows and percentage of GB reduction



(c) Effect of different combinations of weights on scheduling performance



(d) The relationship between the number of TSN flows and the average running time



(e) Effect of different parameters on running time of pre-routing phase

Fig. 4: Evaluation results

in the probability of having a link between any two nodes in the network (p) from the previous run. Overall, the number of TSN switches has the most effect on the running time of the pre-routing phase, and the k value has the least effect.

C. Scalability

To evaluate the scalability of the proposed algorithms, we used TSN networks of three different sizes (small, medium, and large) each with different numbers of TSN flows, and measured the average running time of the entire process, i.e., the time required by the system to route, schedule, and re-route and re-schedule a flow, if necessary, which is specified in Section II and the running time of the pre-routing phase described in Section III-B. Note that the measured running time does not include the time taken by the sender to send the connection request to the controller and the time taken by the controller to configure the switches and send the schedules back to the sender.

The small, medium and large TSN networks, respectively, consist of 10, 25, and 50 TSN switches, 30, 75, and 150 hosts, 500, 1000, and 1500 TSN flows, and have 10, 20, and 30 as k values, and 0.3, 0.4, and 0.5 as p values. In general, a sender may send to multiple end devices (receivers), and a receiver may receive from multiple end devices (senders). To evaluate scalability, we considered the extreme case, where all end devices are both senders and receivers, and they could send/receive TSN packets from all the other end devices. Thus, the pre-routing phase has to find the paths from each end device to all the other end devices. The average running time of the pre-routing phase from multiple runs are 0.93 *seconds*, 136 *seconds*, and 45.3 *minutes* for the small, medium, and large TSN networks, respectively. To measure the runtime of the entire operation (except the pre-routing phase), we use the *SWOTS-ASAP-WS* scheduler since it has the worst running time. The averages running times for the entire operation in the small, medium, and large TSN networks are 46, 61, and 88 milliseconds, respectively.

VI. Conclusion

To increase the number of scheduled TSN flows for higher link utilization, this paper proposed (1) an incremental QoS-aware path-selection algorithm that selects the paths with higher residual bandwidth and lower number of hops and TSN flows, to spare the bottleneck links for any future flows, and (2) multiple incremental scheduling algorithms to schedule the flows over the chosen routes. Based on the evaluation, *SWOTS-ASAP* and *SWOTS-ASAP-WS* yielded the best performance in terms of the number of scheduled flows, while *SWOTS-AEAP-WS* yielded the best performance in terms of wasted bandwidth introduced by the gate opening events and Guard Bands.

References

[1] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *Proceedings of the 24th International Conference on Real-TimeNetworks and Systems*, pages 183–192, 2016.

[2] Frank Dürr and Naresh Ganesh Nayak. No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN). In *Proceedings of the 24th International Conference on Real-Time Networks and Systems - RTNS '16*, pages 203–212, New York, New York, USA, 2016. ACM Press.

[3] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Time-sensitive Software-defined Network (TSSDN) for Real-time Applications. *Proceedings of the 24th International Conference on Real-Time Networks and Systems - RTNS '16*, pages 193–202, 2016.

[4] Naresh Ganesh Nayak, Frank Durr, and Kurt Rothermel. Incremental flow scheduling and routing in time-sensitive software-defined networks. *IEEE Transactions on Industrial Informatics*, 14(5):2066–2075, 2018.

[5] Time-Sensitive Networking Task Group. IEEE 802.1 Time-Sensitive Networking Task Group, 2017.

[6] Zheng Wang and Jon Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, 1996.