Improving Dependability via Deadline Guarantees in Commodity Real-time Networks

Ashish Kashinath, Monowar Hasan, Sibin Mohan, Rakesh B Bobba, Radhika Mittal

Improving Dependability via Deadline Guarantees in Commodity Real-time Networks

Ashish Kashinath*, Monowar Hasan*, Sibin Mohan*, Rakesh B Bobba† and Radhika Mittal*
*University of Illinois at Urbana-Champaign, †Oregon State University,
*{ashishk3,mhasan11,sibin,radhikam}@illinois.edu †rakesh.bobba@oregonstate.edu

***Abstract*—Software-defined Networking (SDN) can facilitate the deployment of deterministic algorithms with stringent Quality of Service (QoS) requirements that pave the way for commodity Real-time (RT) networks. However, current QoS approaches are conservative and under-provision the network, that becomes a bottleneck when scaling or upgrading infrastructure. In this paper, we argue that the use of fixed priorities for flows – a common practice in RT networks, is the root-cause of this issue. We develop algorithms that use the global view provided by SDN to develop on-demand, variable priority schemes. Evaluation shows that this "correct by construction" approach increases network utilization while still meeting the necessary QoS requirements, thereby improving network robustness under high utilization.**

I. Introduction

Deterministic networks are typically found in infrastructure applications such as industrial automation, avionics, IoT and automotive systems where safety and predictability is critical. This is because in infrastructure applications, failures can lead to potentially catastrophic consequences such as loss of human life and property. Finally, these systems have certification requirements (hardware, software, design processes) that are designated by standards such as ISO-26262 for automotive systems and ARP-4761 for avionics systems. *Delay guarantees* for network flows is one such aspect that is central to safe operations of such systems. Such networks, also called hard real-time (RT) networks are currently provisioned using expensive hardware that are built to deploy proprietary, non-interoperable protocols, each requiring its own standardization process - *e.g.,* IEEE 802.1 TSN, TTEthernet, SAE AS6802 are used in industrial ethernet, space communication and automotive systems respectively.

With the push to use commodity hardware and commodity software protocols, Software-defined Networking (SDN) [1] has found its way to guaranteeing Quality of Service (QoS) in RT systems and networks [2], [3], [4], [5], [6], [7], [8], [9]. RT networks typically have a well-defined network structure (topologies, hosts and links), clearly defined flow specifications, and are under the control of a centralized authority (*e.g.,* San Diego Gas & Electric (SDG&E) owns the substation network in San Diego and nearby counties in California). The closed nature of such networks make it amenable for enforcement of system-wide policies that take into account safety requirements such as worst-case latency.

The material in this paper is based upon work supported in part by the U.S. Department of Energy (DoE) award DE-OE0000780 and the National Science Foundation award NSF CPS 1544901. Any findings, opinions, recommendations or conclusions expressed in the paper are those of the authors and do not necessarily reflect the views of sponsors.

While SDN gives a great deal of flexibility, the push to guarantee safety has resulted in very conservative designs. Some of the conservative tactics used in RT networks include: *(i)* rate-limiting of flows, resulting in less throughput for bandwidth-intensive applications [2], [10], *(ii)* spatial isolation of flows using per-flow queues that result in lower number of flows admitted into the network [8] or *(iii)* isolating flows in time epochs using time-division multiplexing or its variants [10]. Although these techniques provide deadline guarantees by design, they lead to low network utilization and high management overheads due to the use of custom hardware and software. Due to the long lifetimes of infrastructure systems, we desire high network utilization with deadline guarantees to allow the network to support system upgrades as more flows are added to the network without having to redeploy extra networking infrastructure.

We investigate the factors involved in the trade-off between determinism and network utilization. We find that the use of *fixed (static)* priorities across the entire RT network leads to lower network utilization. Our experiments show the potential for increased utilization by devising a novel method of varying priorities of a flow at contained local points in the networks. We accomplish this by leveraging network and flow models to obtain tighter guarantees on delays and using it to improve network utilization.

Contributions: Our contributions include:
(i) Demonstrating that using spatially variable priorities for flows improves the ability to accept more flows compared to fixed priorities, *(ii)* Demonstrating that spatially variable priorities allows the network to meet tighter deadlines and *(iii)* a heuristic-based algorithm that can be used to instantiate spatially variable priorities in the network using the global view of an SDN controller giving a a more robust RT network.

II. Background

A. System Model

We consider an SDN network $G(\Pi, \mathbb{L})$ with a set of M switches $\Pi := \{\pi_1, \pi_2, \ldots, \pi_M\}$ and set of links $\mathbb{L}$, each of capacity C. The network has N flows, $\mathbb{F} := \{F_1, F_2, \ldots, F_N\}$ that are assigned paths P_i, $1 \leq i \leq N$. The timing requirement of a flow F_i is represented by the deadline, D_i which is the maximum permissible end-to-end delay the flow F_i can incur on its path P_i for safe and reliable operation of the network.

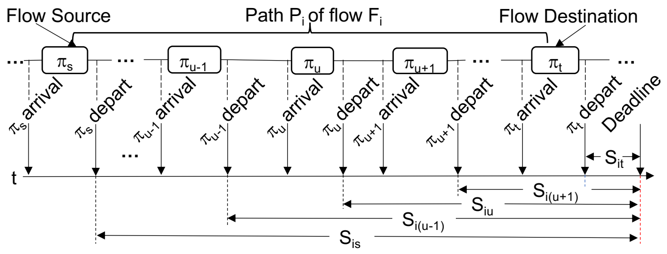

Fig. 1: **Slack of a flow** The slack of a flow F_i at a switch π_u on path P_i indicates proximity to the deadline while accounting for the estimated delay at the switch π_u

Each of the switches $\pi_j \in \Pi$ have L queues, $\{q_1, q_2, \ldots, q_L\}$. The number of flows intersecting a switch π_j is given by N_j and for simplicity $N_j \leq L$.

Modeling Flows: We use arrival curve $\alpha(t)$, a network-calculus mathematical abstraction [11], to model a flow. Specifically, $\alpha(k)$ denotes an *upper bound* on the number of bytes of a flow in a time-interval k. The representation of an arrival curve depends on the nature of the flow. Flows in the network can be classified into 3 types based on their timing requirements and what the designer knows about the flows at design time: *(a)* Periodic flows F_p, *(b)* Sporadic flows F_s, and *(c)* Aperiodic flows F_a.

A periodic flow of x bytes every T seconds is modeled by using a leaky bucket arrival curve, $\alpha_p(t) = rt$, where $r = x/T$. We have complete knowledge about the arrival times, the deadline requirements and priorities of periodic flows. Sporadic flows are modeled using a token bucket arrival curve, $\alpha_s(t) = b + rt$, where r is the maximum rate (obtained from minimum traffic arrival spacing) and b is the maximum allowable burst tolerance. Aperiodic flows, that are usually short-lived, are modeled using a ramp arrival curve, $\alpha_a(t) = min(k(t-T)/e, k), t \geq T$, where k is the number of packets due to the aperiodic event at time T and lasting for time e, the duration for which the flow's bandwidth is k. For $t < T$, $\alpha_p(t) = 0$. In contrast to periodic flows, the arrival times are not known apriori for sporadic and aperiodic flows. In addition, aperiodic flows do not require hard deadline guarantees and the switch tries to *complete each aperiodic flow as soon as possible*. A summary of the parameters of the different types of flow is presented in Table I.

Urgency of a flow: The slack of a flow F_i at a switch π_s, denoted by S_{is} is defined as the proximity to the deadline, D_i. S_{is} determines the degree of flexibility available in scheduling *i.e.,* a flow having lower slack requires more immediate (*i.e.,* is more urgent) scheduling than a flow having higher slack. Typically, at the beginning of the path, the slack of a flow is equal to its deadline and it decreases as the flow traverses its path. The slack of a flow is pictorially represented in Figure 1.

Modeling Switches: Switches are represented by their service curves, $\beta(t)$, that estimate the lower bound of bytes switched from input to the output. The most common service curve is the rate-latency service curve, $\beta(t) = R(t - T)$, that implies flows have to wait for at most T time units before being served at a rate R.

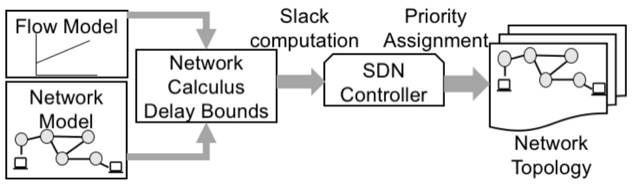

Fig. 2: **Overall Solution Workflow** We leverage the global view of SDN to develop algorithms that schedule flows to maximize utilization without sacrificing deadline guarantees.

TABLE I: **System Designer knowledge** on the different flow types in RT networks. Sporadic and aperiodic flows with unpredictability make scheduling in RT networks challenging and interesting.

FLOW TYPE	ARRIVAL	DELAY	PERIOD[1]	DEADLINE
Periodic	✓	✓	✓	✓
Sporadic	×	✓	✓	✓
Aperiodic	×	×	×	×

[1] called inter-arrival time for sporadic and aperiodic flows

B. Current Challenges & Motivation

For safe operation of RT networks, periodic and sporadic flows must meet their deadlines and aperiodic flows get the best-effort service. Current techniques to schedule these flows fall into one of 3 categories: *(i)* No support for aperiodic and sporadic flows [8], [5], *(ii)* Partial support for aperiodic and sporadic flows with no timing guarantees, [2], [3] and *(iii)* Support all the three types of flows either by providing each with separate network or by deploying on non-commodity switches and protocols [6]. While *(iii)* provides timing isolation for deadline guarantees, this is at the expense of heavily under-provisioning the network and engineering each network according to the traffic type supported.

C. Scheduling objectives

Our goal is to efficiently schedule all the flows in the network while satisfying the following scheduling objectives: *(a) increasing utilization of the network, i.e., accommodating the highest number of flows possible*, *(b) meet deadline guarantees* of existing flows as well as potential sporadic or aperiodic flows and *(c) analyze the trade-offs* between *(a)* and *(b)* .

III. Approach

Our approach to achieving the twin goal of deadline guarantees and improved utilization involves the following steps: *(a)* identifying gaps in static priority-based scheduling schemes used in RT networks, *(b)* using analytical methods (from Network Calculus and RT scheduling theory) to analyze and reason about the gaps and *(c)* proposing 'variable priority' schemes with heuristics and algorithms to enable them.

A. Shortcomings of static priority-based scheduling schemes

Consider the network shown in Figure 3 with the flow specifications and deadlines as shown in Table II. We first consider a static priority assignment scheme where the flows having nearest deadline has the highest priority and priority decreases as deadlines get farther. Thus, in this case, priority of $F_5 > F_1 > F_2 > F_3 > F_4$. We vary the utilization of the network from lightly loaded (50%) to heavily loaded ($>$ 90%) by varying the link capacities. We capture the slack of all flows at every switch along its path and plot it as a function of the switch in Figure 4.

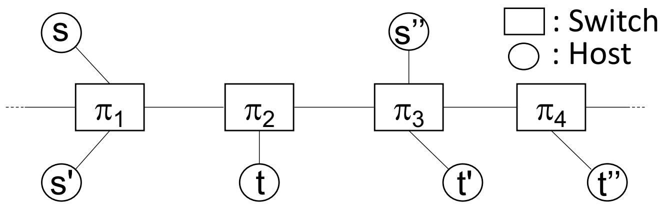

Fig. 3: **Motivational Example** 4-switch 6-host topology demonstrating the limitation of Static Priority Assignment using Deadline Monotonic Algorithm. The links have 100 Mbps capacity.

TABLE II: Flow specifications for the topology in Figure 3

Flow ID	Size[1]	Rate	Deadline	Path
F_1	1 kB	8 Mbps	9 ms	$\{\pi_1, \pi_2\}$
F_2	1 kB	12 Mbps	11 ms	$\{\pi_1, \pi_2\}$
F_3	1 kB	24 Mbps	13 ms	$\{\pi_1, \pi_2\}$
F_4	1 kB	3 Mbps	16 ms	$\{\pi_1, \pi_2, \pi_3\}$
F_5	1 kB	3 Mbps	4 ms	$\{\pi_3, \pi_4\}$

[1] size of a packet

Summary (Figures 4a - 4d) Figures 4a - 4d depict the slack of flows across different switches along its path. We observe that the slack is a monotonically decreasing function. Any flow with a non-negative slack at the end of its path is said to have met its deadline.

Impact of utilization on a given flow (Figs 4a & 4b) For a given flow at any switch, the slack decreases as the network utilization increases. For example, the slack of Flow 4 at Switch 2 decreases from 14 ms to 11 ms as utilization is increased from 50% to 70%.

Impact of utilization on a given switch path (Fig. 4c) At a given switch path (*e.g.,* Switch 1 to Switch 2), we see that the rate of decrease the slack is inversely proportional to the priority assignment. A flow with a higher priority has a lower drop in slack than a flow with lower priority. The consequence of this is that, at higher utilization, a lower priority flow such as Flow 4 misses deadlines.

Impact of utilization on deadlines at high loads (Fig. 4c-4d) We see that the slack of Flow 4 drops to zero at Switch 2 (Fig. 4d) and it reaches its destination in Switch 3 with a negative slack, thereby missing its deadline. Flow 4 meets its deadline at 90% load but misses it at 92.6% load.

Variable priority-based scheduling schemes at high loads: We now allow local priority changes for a subset of flows and observe the effects on slack. We tweak the priorities of Flow 4 (that missed its deadline in Fig. 4d) and capture the slack in Fig. 5.

Summary Figures 5a - 5b depict that the process of changing the priority of a flow at a local level helps meet deadlines as long as it is done at the appropriate switch along the path. Priority changes can be useless if performed later or earlier.

Impact of priority on deadlines at high loads (Fig. 5a-Fig. 5b) We see that by raising the priority of Flow 4 by two priority levels at a point of contention (*e.g.,* Switch 2), it is able to meet its deadline. Also, raising the priority after the points of contention (*e.g.,* Switch 3), the flow would not see any benefits as it is too late to make up for the lost slack.

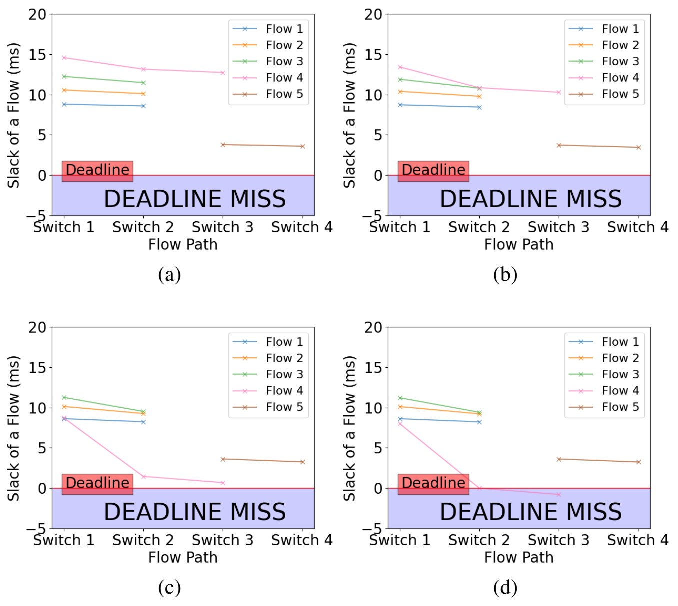

Fig. 4: **Spatial variation of slack of flows along its paths demonstrating slack going negative at high loads - 92.6% for this specific topology** : (4a) 50% load, (4b) 70% load, (4c) 90% load and (4d) 92.6% load. The slack is a decreasing function along the path of a flow, starting at a value equal to the deadline of the flow – when slack becomes negative, the flow has missed its deadline.

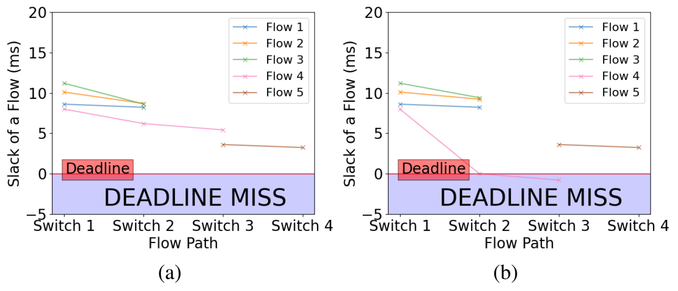

Fig. 5: **Raising priority of flow F_4 at intermediate switches helps accommodate the 92.6% load** : (5a) Raising priority of flow F_4 at Switch 2 saves enough slack for the flow to consume in future hops, (5b) Raising priority of flow F_4 at Switch 3 is too late as the slack has become 0 at Switch 2.

B. Mechanism to Enable Variable Priority Schemes: Delay Measurements via Network Calculus Analysis

To enable variable priorities, we use analytical tools from Network Calculus ([2]) to derive the upper bounds on end-to-end delays of a flow along its path considering the 3 classes of flows at a switch as described in Section II.

We reserve the highest priority queue for sporadic flows – while they have deadline requirements yet the designer has only partial knowledge of the flow specifications. This is accomplished by reframing a sporadic flow as a *synthetic periodic flow* called a 'bandwidth-preserving deferrable server' of period $p_s = \frac{b_s}{r_s}$ and buffer size b_s, where b_s is the maximum burst in bytes and r_s is the rate corresponding to the minimum inter-arrival time. Note that due to the non-preemptive nature of network links, we allocate a deferrable maximum burst and minimum inter-arrival time. This parameter can be tuned according to the desired range of sporadic flows accepted by the server. Additionally, the lowest priority queue is reserved for aperiodic flows that require only best-effort service. As a result, the remaining queues are for periodic flows. Therefore, the queue allocation can be summarized as :

- q_1 : Sporadic Flow,
- $\{q_2, q_3 \ldots q_{(L-1)}\}$: Periodic Flow and
- q_L : Aperiodic Flow

We derive analytical expressions beginning with the fixed priority model from Guck *et al.* [2] and customizing it for our variable priority model. Towards this, we extend expressions of slack that are derived for the fixed priority model.

Fixed Priorities For the case of static priorities, we can express the service curve, $\beta_{ij}(t)$, offered to the periodic flow F_i in the queue j in switch with capacity C as:

$$\overset{\text{Service Avail.}}{\beta_{ij}(t)} = \Big(\underset{\text{Total Service Cap.}}{Ct} - \big(\overset{\text{Spor. Flows}}{tr_s + b_s}\big) - \overset{\text{Higher Prio}}{t\sum_{k=1}^{j-1} r_k} - \underset{\text{Non-Pre. Blocking}}{\max_{(j+1)\le k\le N_j} l_k^{max}} - \underset{\text{Str \& Fwd}}{l_i^{max}}\Big), \tag{1}$$

where terms $tr_s + b_s$ denote the service consumed by the sporadic traffic, $t\sum_{k=1}^{j-1} r_k$ denote the service curve due to higher priority flows, $\max_{(j+1)\le k\le N_j} l_k^{max}$ the blocking delay from lower priority flows and l_i^{max} is the store-and-forward correction factor.

The service curve in Equation (1) can be rewritten as a rate-latency curve $\beta_{R_{ij},T_{ij}} = R_{ij}(t - T_{ij})$ with $R_{ij} = C - r_s - \sum_{k=1}^{(j-1)} r_k$ and $T_{ij} = \frac{b_s + \sum_{k=(j+1)}^{k=N_j} l_k^{max} + l_i^{max}}{R_{ij}}$. Note that R_{ij} is the service rate and T_{ij} is the service latency provided to the flow F_i in queue j. The *Total Switch Delay* experienced by a flow F_i in a queue j at a single switch π_s, denoted by d_{ijs} is given by Equation 2, where b_i is the burst of the flow F_i.

$$\overset{\text{Per Switch Delay}}{d_{ijs}} = \underset{\text{Service Latency}}{T_{ij}} + \frac{\overset{\text{Burst}}{b_i}}{\underset{\text{Service Rate}}{R_{ij}}}, \tag{2}$$

Slack-driven Priorities To derive priorities based on slack, we need to consider the scenario when *all* the other flows can potentially impact the flow F_i in queue j under consideration. Thus the rate latency curve now becomes $\beta_{R_{ij},T_{ij}} = R_{ij}(t - T_{ij})$ with $R_{ij} = C - r_s - \sum_{k=1,k\neq i}^{N_j} r_k$ and $T_{ij} = \frac{b_s + \sum_{k=1}^{k=N_j} l_k^{max}}{R_{ij}}$. Again, we can use Equation 2 to calculate the Total Switch Delay.

End-to-end Delay The end-to-end delay of a flow F_i over a path P_i is calculated by adding the individual switch delays incurred in the queues q_j of all the switches π_s in the path P_i. Therefore, the end-to-end delay d_i is given by :

$$\overset{\text{End-to-end Delay}}{d_i} = \overset{\text{Add Per-switch Delays}}{\sum_{F_i\subset q_j,\pi_s\in P_i} d_{ijs}}, \tag{3}$$

We use the delay computations at every switch in the path to compute the slack. The slack of a flow F_i along a path P_i at switch π_s is represented in Figure 1 and given by:

$$\overset{\text{Slack}}{S_{is}} = \overset{\text{Deadline}}{D_i} - \overset{\text{End-to-end Delay}}{\sum_{F_i\subset q_j,\pi_s\in P_i} d_{ijs}}, \tag{4}$$

The slack of a flow F_i at the end of its path P_i is denoted by S_i. If $S_i \geq 0$, we say that F_i has met its deadline. On the other hand, if $S_i < 0$, F_i has missed its deadline.

Algorithm 1 GALE-LSTF

```
Stage 1 : Initialization with LSTF We schedule the flows $F_i \in \mathbb{F}$
  on paths $P_i \in \mathbb{P}$ using Least Slack Time First (LSTF) scheduling and
  compute the slack values $S_i$ at the output of $P_i$ using Eqn 4.
if $S_i \geq 0\ \forall F_i \in \mathbb{F}$ then
    return Schedulable /* LSTF is sufficient for the flow set $\mathbb{F}$ */
end if
/* LSTF failed to meet deadlines for $\mathbb{F}$. Retry using priority exchange */
Stage 2: Linear Search for Priority Exchange
$\mathbb{F}_{miss} = \{F_i \in \mathbb{F} \mid S_i < 0\}$ /* Flows not meeting deadlines via LSTF */
/* Goal: Schedule flows in $\mathbb{F}_{miss}$ while not missing deadlines of flows in
  $\mathbb{F} - \mathbb{F}_{miss}$ */
/* Loop through all flows $F_i$ missing deadlines via LSTF */
$schedulable_i$ = false , $\forall F_i \in \mathbb{F}_{miss}$ /* A Boolean flag */
while $\mathbb{F}_{miss}$ not empty and elements in $\mathbb{F}_{miss}$ change in consecutive
  iterations do
    for $F_i \in \mathbb{F}_{miss}$ do
        /* Try all switches in the path $P_i$ in reverse order */
        for $\pi_k \in P_i$.reverse() do
            $\widehat{p}_i^k \leftarrow$ Priority of $F_i$ at switch $\pi_k$
            $\mathbb{G} \leftarrow$ Set of all flows at switch $\pi_k$ that has priority $> \widehat{p}_i^k$
            /* Try all priority exchanges at switch $\pi_k$ */
            for $F_j \in \mathbb{G}$ do
                Swap priority of flow $F_i$ and $F_j$
                Recompute $S_i$ of all flows intersecting $\pi_k$ and flows in $\mathbb{F} -
                  \mathbb{F}_{miss}$ using Eqn. 4
                if Slack $\forall$ flows in $\pi_k \geq 0$ and Slack $\forall$ flows in $\mathbb{F} -
                  \mathbb{F}_{miss} \geq 0$ then
                    /* Assign these priorities at switch $\pi_k$ */
                    $schedulable_i$ = true
                    /* Remove $F_i$ from $\mathbb{F}_{miss}$ */
                    $\mathbb{F}_{miss} \leftarrow \mathbb{F}_{miss} - F_i$
                    break
                end if
            end for
        end for
    end for
end while
/* Finished trying all flows $F_i$ */
if $schedulable_i$ = true $\forall F_i \in \mathbb{F}_{miss}$ then
    return Schedulable /* GALE-LSTF scheduled flows in $\mathbb{F}_{miss}$ */
end if
return Unschedulable /* GALE-LSTF could not schedule $\mathbb{F}$ */
```

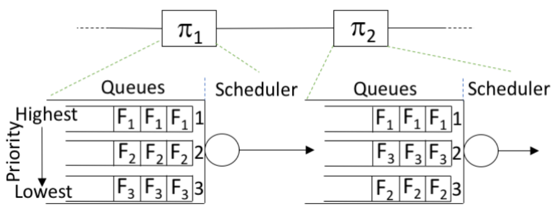

Fig. 6: **Dynamic Slack computation demonstrating spatially varying priorities across different switches**. Flow F_3 has variable priorities at π_1 and π_2, occupying different queues.

C. Policy/Heuristics to Enable Variable Priority Schemes: GALE-LSTF

The slack of a flow F_i at switch π_s, given by Equation 4, provides a measure of the degree of closeness to the deadline. Therefore, at a switch using Least Slack Time First (LSTF) scheduling, a flow with a lower value of slack is scheduled before a flow with a higher value of slack. Mittal *et al.* [12] argues that LSTF comes closest to mimicking a universal packet scheduler when there are more than two switches. However, there are possible schedules which LSTF does not capture due to two factors – *(a)* Lack of global information of flow slacks and *(b)* Lack of information about flow routes. Due to the well-defined structure of RT networks and clearly defined flow specifications, we have access to additional information with respect to *(a)* and *(b)* . Therefore, we package LSTF with heuristics and policies (called *GALE-LSTF*) with the goal of accommodating flows that were hitherto unschedulable by LSTF thereby increasing utilization of the network.

Intuition behind Policy: In practice, flows in a network typically take diverse paths and have different deadlines. It is likely that flows that take a longer path have a higher deadline and as a result if we were to assign priorities in a deadline-monotonic manner (lower deadline $\rightarrow$ higher priority), the longer path flows could be designated to lower levels of priority at every switch and as a result could potentially miss their deadlines. Instead, slack allows us to track the progress of a flow towards its deadline and if needed, change priorities of a flow from one switch to another.

According to LSTF, at every switch π_s, the flows are reorganized into priorities $F_1 \geq F_2 \geq F_3 \geq \ldots \geq F_{N_s}$, where $S_{1s} \leq S_{2s} \leq S_{3s} \leq \ldots \leq S_{N_s s}$. As a result, the priority of a flow can vary from one switch to another in the path.

GALE-LSTF: To LSTF, we introduce an optimization where we alter the priorities assigned by LSTF at certain, constrained switches in the network. We induce *slack-based priority inversions* when the slack maintained by LSTF is unable to schedule flows while meeting flow deadlines. This gives us a better acceptance ratio because LSTF uses only local slack values at the switches while GALE-LSTF uses global knowledge of the network structure and traffic characteristics to determine when a flow would miss its deadline and accordingly induce necessary priority inversions. GALE-LSTF, a 2-stage algorithm is described next and summarized in Algorithm 1. A high-level overview of the overall workflow is illustrated in Fig 2.

Stage 1: Initialization with LSTF As noted, we are provided with the assignment of flows to routes *i.e.,* for flows $F_i \in \mathbb{F}$, the routes P_i, $1 \leq i \leq N$are known. Using LSTF scheduling on routes provided, we obtain slack measurements S_i for flow F_i at the output of path P_i. If here all the flows have slack $S_i \geq 0$, then LSTF is sufficient for flow set $\mathbb{F}$ and we can return. On the contrary, if some flows missed their deadlines using LSTF, we need to go to the next stage. For this, we partition the flow set $\mathbb{F}$ into two disjoint subsets - *(i)* $\mathbb{F}_{\text{miss}}$ - flows that missed deadlines using LSTF and *(ii)* Rest *viz.,* $\mathbb{F} - \mathbb{F}_{\text{miss}}$ *i.e.,* flows that met their deadlines using LSTF.

Stage 2: Linear Search for Priority Exchange For every flow F_i in $\mathbb{F}_{\text{miss}}$, we search its path (*i.e.,* P_i) starting from the destination to find suitable switches π_k where priority exchanges can be performed. Searching in the direction opposite to the flow minimally perturbs flows already accepted by LSTF. At every candidate switch, we consider flows that have priority higher than F_i, and perform priority exchange and recompute slack using Eqn. 4 to check if the flows – both at the switch and those in $\mathbb{F} - \mathbb{F}_{\text{miss}}$ meet their deadlines. If so, we mark F_i as schedulable and use this priority assignment at switch π_k. Note that this priority reassignment at switch π_k – by performing *slack-based priority inversions* and raising the priority of flows with negative slack and lowering priority of other flows with lower slack is counter to LSTF. This process is repeated for other flows in $\mathbb{F}_{\text{miss}}$. After stage 2, we check if all flows in $\mathbb{F}_{\text{miss}}$ met its deadlines using GALE-LSTF and return accordingly.

IV. Evaluation

A. Implementation

Our solution is developed as an application implementing Algorithm 1 that takes into account flow specifications, the network topology, the routes and performs priority reassignment for every switch of the path. The application is amenable to be implemented in an SDN controller.

B. Simulation-based Evaluation

We evaluate our solution in 2 stages:

- **Fixed Priority vs. LSTF:** Simulations comparing the utilization of both LSTF and Fixed Priority (FP) algorithm such as Deadline Monotonic Scheduling.
- **Insufficiency of LSTF at high loads** *i.e.,* **Benefits of using GALE-LSTF** Simulations comparing the benefits of varying priorities of flows across switches to accommodate cases where LSTF gives lower acceptance.

1) Fixed Priority vs. LSTF: We study a simple network consisting of 2 hosts and 2 switches arranged as: *Host-Switch 1-Switch 2-Host*. We stress test this network by incrementally adding flows (4 $\rightarrow$ 8 $\rightarrow$ 12 $\rightarrow$ 16) and capturing the end-to-end delays of all flows. The deadlines of all flows are fixed at 40 ms and the network capacity is 100Mbps. The experiment is performed twice- *(a)* when flows are scheduled by Fixed Priority and *(b)* when flows are scheduled by LSTF.
Summary Fig. 7 shows that the FP scheduling scheme violates deadlines for a lower number of flows compared to LSTF. This shows that given a deadline, LSTF can schedule more flows.

Delay comparison for a fixed number of flows (Fig. 7) For a given number of flows, LSTF shows a lower variation and a lower worst-case end-to-end delay ($99.9\%^{ile}$) as compared to fixed priority assignments. This is because the slack-based priorities is able to reassign the priorities at Switch 2 to make up for lost slack in Switch 1. Static priority is not able to take advantage of this and thus flow delays are more dispersed.
Acceptance/Utilization Comparison (Fig. 7) We find that LSTF is able to accept more flows before the deadlines are violated whereas FP algorithm starts missing deadlines at a lower number of flows. In this case, we see that FP is able to accept 7 flows whereas LSTF is able to accept upto 12 flows.

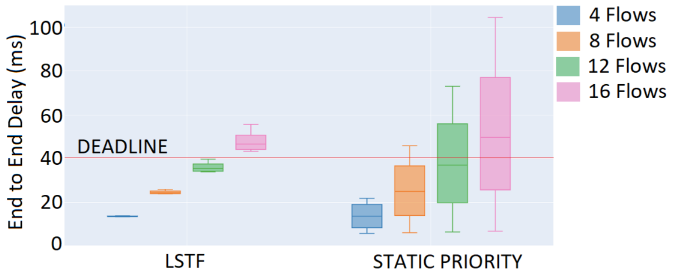

Fig. 7: End-to-End Delay values for comparison between Static Priority (FP) Scheduling and Least Slack Time First (LSTF) Scheduling Algorithm showing that Static Priority scheme violates deadline at a lower number of flows (8).

2) LSTF vs. GALE-LSTF: For networks with high loads, we show that LSTF has the tendency to miss deadlines, especially for flows across long hops – there is a possibility that flows with lower priorities (at earlier hops) can have slack dropping below 0 at an intermediate switch. We consider a linear topology of 6 switches, with 3 flows, loading the network at 90% utilization. Flows 1 and 2 have path $\{\text{Switch } 1, \text{Switch } 2\}$ whereas Flow 3 has the path $\{\text{Switch } 1, \text{Switch } 2, \ldots, \text{Switch } 6\}$.The flow parameters are identical to Table II.

Summary (Fig. 8) Both Figure 8a and Figure 8b plots the slack of flows across the network. In case of Figure 8a, Flow 3 misses its deadline as its slack becomes zero at its penultimate switch, whereas using GALE-LSTF, we promote Flow 3 in Switch 2 despite its higher slack than Flow 1 and 2, thereby allowing Flow 3 to meet its deadlines. This also shows that LSTF does not meet deadlines for all cases, and demonstrates the need to develop additional heuristics and algorithms.

V. Discussion

We introduced Global Aware Local Effects LSTF (GALE-LSTF) as a means of spatially varying the priority of a flow from one switch to another. In our current scheme, the number of flows is limited by the number of queues, as our system model does not include FIFO multiplexing delays. We used a slack-based heuristic to perform priority exchanges.

VI. Conclusion

Unlike wide-area networks, RT networks have different metrics of performance *viz.,* certification, predictability and

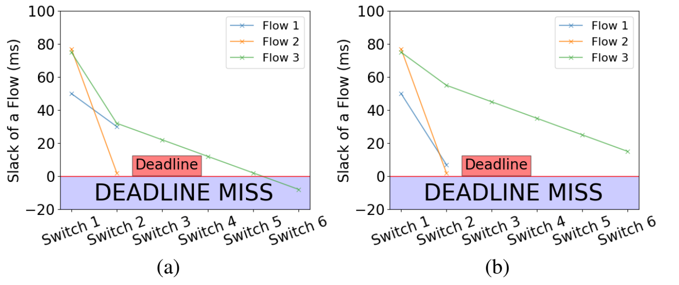

Fig. 8: **LSTF vs. GALE-LSTF for a network at 90% load** : (8a) LSTF violates deadlines of Flow 3, (8b) GALE-LSTF meets all the deadlines.

deadline guarantees. In addition, the specialized and controlled nature of RT networks with SDN allow us to tune facets such as scheduling to meet these requirements while improving utilization. Today, the most common scheduling schemes in RT systems are fixed priority schemes – we show that using variable priority schemes in network with relevant optimizations can pave the way for improved utilization, thus lowering the cost and engineering effort to deploy RT networks.

References

[1] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[2] J. W. Guck, A. Van Bemten, and W. Kellerer, "Detserv: Network models for real-time qos provisioning in sdn-based industrial environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1003–1017, 2017.

[3] A. Van Bemten, Nemanja, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: Predictable latency in small networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 355–369.

[4] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (tssdn) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 193–202.

[5] K. Lee, M. Kim, H. Kim, H. S. Chwa, J. Lee, and I. Shin, "Fault-resilient real-time communication using software-defined networking," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 204–215.

[6] K. Lee, T. Park, M. Kim, H. S. Chwa, J. Lee, S. Shin, and I. Shin, "Mc-sdn: Supporting mixed-criticality scheduling on switched-ethernet using software-defined networking," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 288–299.

[7] T. Qian, F. Mueller, and Y. Xin, "A linux real-time packet scheduler for reliable static sdn routing," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[8] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba, "End-to-end network delay guarantees for real-time systems using sdn," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 231–242.

[9] G. S. Aujla, A. Singh, and N. Kumar, "Adaptflow: Adaptive flow forwarding scheme for software-defined industrial networks," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5843–5851, 2020.

[10] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can {JUMP} them!" in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 1–14.

[11] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[12] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 501–521.