Dependable End-to-End Delay Constraints for Real-Time Systems using SDN*

Rakesh Kumar[1], Monowar Hasan[1], Smruti Padhy[2], Konstantin Evchenko[1],
Lavanya Piramanayagam[3], Sibin Mohan[1] and Rakesh B. Bobba[4]
[1]University of Illinois at Urbana-Champaign, USA, [2]National Center for Supercomputing Applications, USA
[3]PES University, India, [4]Oregon State University, USA
Email: [1,2]{kumar19, mhasan11, spadhy, evchenk2, sibin}@illinois.edu
[3]lava281995@gmail.com, [4]rakesh.bobba@oregonstate.edu

ABSTRACT

Real-time systems (RTS) require dependable delay guarantees on delivery of network packets. Traditional approaches for providing such guarantees for mission critical applications require the use of expensive custom built hardware and software resources. We propose a novel framework that reduces the management and integration overheads of the traditional approaches by leveraging the capabilities (especially global visibility and ease of management) of Software-defined networking (SDN) architectures. Given the specifications of flows requiring real-time guarantees, our framework synthesizes paths through the network and associated switch configurations that can meet the requisite delay requirements for real-time flows. Using exhaustive simulations as well as emulations with real switch software, we show the feasibility of our approach.

1. INTRODUCTION

Software-defined networking (SDN) [23] has become increasingly popular since it allows for better management of network resources, application of security policies and testing new algorithms and mechanisms. It finds use in a wide variety of domains – from enterprise systems [21] to cloud computing services [17], from military networks [28] to power systems [25] [7]. The centralized (and global) view of the network obtained by the use of SDN architectures provides significant advantages when compared to traditional networks. It enables network designers and engineers to push down rules to various nodes in the network to manage the bandwidth and resource allocation for flows through the entire network to a fine level of precision.

Real-time systems (RTS), especially those with stringent timing constraints, need to reason about delays. *Packets must be delivered between hosts with guaranteed upper bounds on end-to-end delays.* Examples of such systems include avionics, automobiles, industrial control systems, power substations, manufacturing plants, *etc.* Another property of such systems is that they often include traffic flows with mixed criticality, *i.e.,* those with varying degrees of timing (and perhaps even bandwidth and availability) requirements: *(a) high priority/criticality traffic* that is essential for the correct and safe operation of the system; examples include sensor data and control commands for closed loop control in avionics, automotive or power grid systems; *(b) medium criticality traffic* that is critical to the correct operation of the system, but with some tolerances in delays, packet drops, *etc.*; for instance, navigation systems in aircraft, system monitoring traffic in power substations, *etc.*; *(c) low priority traffic* – essentially all other traffic in the system that does not really need guarantees on delays or bandwidth such as engineering traffic in power substations, multimedia flows in aircraft, *etc.*

The high priority flows (that we henceforth refer to as "Class I" traffic) have stringent timing requirements and can often tolerate little to no loss of packets. Often these flows also have predefined *priority levels* among themselves. Typically, in many safety-critical RTS, the properties of all Class I flows are well known, *i.e.,* designers will make these available ahead of time. Any changes (addition/removal of flows or modifications to the timing or bandwidth requirements) will often require a serious system redesign. The number (and properties) of other flows could be more dynamic – consider the on-demand video situation in airplanes where new flows could arise and old ones stop based on the viewing patterns of passengers.

Current safety-critical systems often have separate networks (hardware and software) for each of the aforementioned types of flows (for safety and sometimes security reasons). This leads to significant overheads (equipment, management, weight, *etc.*), increases potential for errors or faults, and even increases the attack surface. Existing systems, *e.g.,* avionics full-duplex switched Ethernet (AFDX) [4, 9], controller area network (CAN) [15], *etc.* that are in use in many of these domains are either proprietary, complex, expensive or even require custom hardware. Despite the fact that AFDX switches ensure timing determinism, traffic flows transmitted on such switches may be changed frequently at run-time when sharing resources (*e.g.,* bandwidth) among different networks [22]. In such situations, a dynamic configuration is required to route packets based on switch workloads and flow delays to meet all the high priority QoS (*e.g.,* end-to-end delay) requirements. In addition AFDX protocols require custom hardware [11].

In this paper we present mechanisms to *guarantee end-to-end delays for high-criticality flows (Class I) on networks constructed using SDN switches, i.e., essentially commercial-off-the-shelf (COTS) components.* The advantage of using SDN is that it provides a centralized mechanism for configuring and managing the system. The global view is useful in providing the end-to-end guarantees that are required. Another advantage is that the hardware/software resources needed to implement all of the above types of traffic can be reduced since we can use the same network infrastructure (instead of three separate ones as is the case these days). On the other hand, the current standards used in traditional SDN (OpenFlow [23,27]) generally do not support end-to-end delay guarantees or even existing real-time networking protocols such as AFDX. Retrofitting OpenFlow into AFDX is not straightforward and generally less effective [14].

A number of issues arise while developing a software-defined networking infrastructure for use in real-time sys-

*The material presented in this paper is based upon work supported in part by the Department of Energy under Award Number DE-OE0000780 and the National Science Foundation Award NSF CPS 1544901.

tems. For instance, flows belonging to Class I need to meet their *timing* (*e.g.,* end-to-end delay) requirements for the real-time system to function correctly and be dependable. Hence, we need to *find a path* through the network (and allocate the resources such as bandwidth[1]) that will meet these guarantees. In contrast to traditional SDNs, it is not necessary to find the *shortest* path through the network. Oftentimes, Class I flows can arrive *just in time* [24, 26], *i.e.,* just before their deadline – there is no real advantage in getting them to their destinations well ahead of time. Path layout for real-time SDN is a *non-trivial* problem since, *(i)* we need to understand the delay(s) caused by individual nodes (*e.g.,* switches) on a Class I flow and *(ii)* compose them along with an understanding of the delays/problems caused by the presence of other flows in that node as well as the network in general.

In this work we consider Class I (*i.e.,* high-criticality) flows and develop a scheme to meet their timing constraints[2]. We evaluate the effectiveness of the proposed approach using different custom topologies and UDP traffic[3].

2. SYSTEM MODEL

Consider an SDN topology (N) with open flow switches and controller, and a set of real-time flows (F) with specified delay and bandwidth guarantee requirements. The *problem is to find paths for the flows (through the topology) such that the flow requirements (i.e., end-to-end delays) can be guaranteed for the maximum number of critical flows.* We model the network as an undirected graph $N(V, E)$ where V is the set of nodes, each representing a switch port in a given network and E is set of the edges[4], each representing a possible path for packets to go from one switch port to another. Each port $v \in V$ has a set of queues v_q associated with it, where each queue is assigned a fraction of bandwidth on the edge connected to that port.

Consider a set F of unidirectional, real-time flows that require delay and bandwidth guarantees. The flow $f_k \in F$ is given by a four-tuple (s_k, t_k, D_k, B_k), where $s_k \in V$ and $t_k \in V$ are ports (the source and destination respectively) in the graph, D_k is the maximum delay that the flow can tolerate and B_k is the maximum required bandwidth by the flow. We assume that flow priorities are distinct and the flows are prioritized based on a *"delay-monotonic"* scheme *viz.,* the end-to-end delay budget represents higher priority (*i.e.,* $pri(f_i) > pri(f_j)$ if $D_i < D_j$, $\forall f_i, f_j \in F$ where $pri(f_k)$ represents priority of f_k).

For a flow to go from the source port s_k to a destination port t_k, it needs to traverse a sequence of edges, *i.e.,* a flow path $\mathcal{P}_k$. The problem then, is to synthesize flow rules that use queues at each edge $(u, v) \in \mathcal{P}_k$ that can handle *all* flows F in the given system while still meeting each flow's requirement. If $d_{f_k}(u, v)$ and $b_{f_k}(u, v)$ is the delay faced by the flow and bandwidth assigned to the flow at each edge $(u, v) \in E$ respectively, then $\forall f_k \in F$ and $\forall (u, v) \in \mathcal{P}_k$ the following constraints need to be satisfied:

$$\sum_{(u,v)\in\mathcal{P}_k} d_{f_k}(u, v) \leq D_k, \quad \forall f_k \in F \qquad (1)$$

$$b_{f_k}(u, v) \geq B_k, \quad \forall (u, v) \in \mathcal{P}_k, \forall f_k \in F. \qquad (2)$$

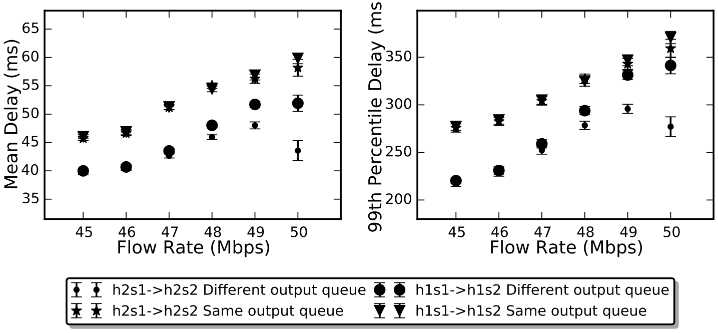

Figure 1: Delay measurement experiments: the measured mean and 99th percentile per-packet delay for the packets in the active flows in 25 iterations.

Once we obtain path layouts[5] that satisfy the above constraints, the next step is to map flows assigned to a port to the queues at the port. The possible approaches could be: *(a) allocate each flow to an individual queue* or *(b) multiplex flows in different queues* and dispatch the packets based on priority. Our intuition is that the *end-to-end delays are lower and more stable* when *separate queues* are provided to each critical flow – especially as the rates for the flows get closer to their maximum assigned rates. We carried out some simple experiments to demonstrate this (and to highlight the differences between these two strategies) – this is outlined in the following section.

2.1 Queue Assignment Strategies

We propose synthesizing configurations for Class I traffic such that it ensures *complete isolation of packets in a designated class I flow.* In order to test how using output queues can provide isolation to flows in a network so that each can meet its delay and bandwidth requirements simultaneously, we setup a experiment using Mininet [20]. The experiment uses a simple topology that contains two switches (`s1`, `s2`) connected via a single link. Each switch has two hosts connected to it. We configured flow rules and queues in the switches to enable connectivity among hosts at one switch with the hosts at other switch. We experimented with two ways to queue the packets as they cross the switch-to-switch link: *(i)* in one case, we queue packets belonging to the two flows *separately* in two queues (*i.e.,* each flow gets its own queue), each configured at a maximum rate of 50 Mbps *(ii)* in the second case, we queue packets from both flows in the *same queue* configured at a maximum rate of 100 Mbps. Finally, we used *ingress policing* such that if traffic from a host exceeds its maximum rate (50 Mbps in this case), fits traffic is throttled before it enters the switch.

After configuring the flow rules and queues, we triggered packet flows using `netperf` [6]: the first starting at the host `h1s1` destined to host `h1s2` and the second starting at host `h2s1` with a destination to host `h2s2`. Both packet flows are triggered simultaneously and they last for 10 seconds. We changed the rate at which the traffic is sent across both flows and performed each experiment 50 times to measure the average and 99^{th} percentile for per-packet delays. Figure 1 plots the average value and standard error over all iterations. The x-axis indicates the rate at which the traffic is sent via `netperf`, while the y-axis shows the mean and 99th percentile delay. The following key observations stand out:

1. The per-packet average and 99^{th} percentile delay increases in both cases as traffic send rate approaches the configured rate of 50 Mbps. This is an expected queue-theoretic outcome and motivates the need for slack allocations for all applications in general. For example, if an application requires a bandwidth

[1] Current SDN implementations reason about bandwidth instead of delays. Hence, we must find a way to extend the SDN infrastructure to reason about delays for use in real-time systems.

[2] Integrating the other types of traffic is left for future work.

[3] Since most hard real-time systems use UDP traffic [11].

[4] We use the terms *edge* and *link* interchangeably throughout the paper.

[5] We formulate this problem as a multi-constrained path (MCP) problem and describe the solution in Section 3.

guarantee of 1 Mbps, it should be allocated 1.1 Mbps for minimizing jitter.

2. The case with separate queues experiences *lower* average per-packet average and 99^{th} percentile delay when flow rates approach the maximum rates. This indicates that when more than one flow uses the same queue, there is interference caused by both flows to each other. This becomes a source of unpredictability and eventually may cause the end-to-end delay guarantees for the flow to be not met or perturbed significantly.

Thus, *isolating flows using separate queues results in lower and more stable delays* especially when traffic rate in the flow approaches the configured maximum rates. The maximum delay along a single link can be measured. Such measurements can then be used as input to a path allocation algorithm that we describe in the following section.

3. PATH LAYOUT: OVERVIEW AND SOLUTION

We now present a more detailed version of the problem (composing paths that meet the end-to-end delays for critical real-time flows) and also an overview of our solution.

3.1 Problem Overview

Let $\mathcal{P}_k$ be the path from s_k to t_k for flow f_k that needs to be determined. Let $\mathfrak{D}(u, v)$ be the delay incurred on the edge $(u, v) \in E$. The total delay for f_k over the path $\mathcal{P}_k$ is given by $\mathfrak{D}_k(\mathcal{P}_k) = \sum_{(u,v)\in\mathcal{P}_k} \mathfrak{D}(u, v)$. Therefore we define the following constraint on end-to-end delay for the flow f_k as

$$\mathfrak{D}_k(\mathcal{P}_k) \leq D_k. \tag{3}$$

Note that the end-to-end delay for a flow over a path has following delay components: *(a)* processing time of a packet at a switch, *(b)* propagation on the physical link, *(c)* transmission of packet over a physical link, and *(d)* queuing at the ingress/egress port of a switch. As discussed in the Section 2, we use separate queues for each flow with assigned required rates. We also over-provision the bandwidth for such flows so that critical real-time flows do not experience queueing delays.

The second constraint that we consider in this work is *bandwidth utilization*, that for an edge (u, v) for a flow f_k, can be defined as: $\mathfrak{B}_k(u, v) = \frac{B_k}{B_e(u,v)}$ where B_k is the bandwidth requirement of f_k and $B_e(u, v)$ is total bandwidth of an edge $(u, v) \in E$. Therefore, bandwidth utilization over a path $(\mathcal{P}_k)$, for a flow f_k is defined as: $\mathfrak{B}_k(\mathcal{P}_k)) = \sum_{(u,v)\in\mathcal{P}_k} \mathfrak{B}_k(u, v)$. Note that the bandwidth utilization over a path $\mathcal{P}_k$ for flow f_k is bounded by

$$\mathfrak{B}_k(\mathcal{P}_k) \leq \max_{(u,v)\in E} \mathfrak{B}_k(u, v)|V| \tag{4}$$

where $|V|$ is the cardinality of a set of nodes (ports) in the topology N. Therefore in order to ensure that the bandwidth requirement B_k of the flow f_k is guaranteed, it suffices to consider the following constraint on bandwidth utilization

$$\mathfrak{B}_k(\mathcal{P}_k) \leq \widehat{B}_k \tag{5}$$

where $\widehat{B}_k = \max\limits_{(u,v)\in E} \mathfrak{B}_k(u, v)|V|$.

Remark 1. *The selection of an optimal path for each flow $f_k \in F$ subject to delay and bandwidth constraints in Eq. (3) and (5), respectively can be formalized as a multi-constrained path (MCP) problem that is known to NP-complete [16].*

Therefore we consider a polynomial-time heuristic similar to that presented in literature [10]. The key idea is to *relax* one constraint (*e.g.,* delay or bandwidth) at a time and try to obtain a solution. If the original MCP problem has a solution, one of the relaxed versions of the problem will also have a solution [10]. In what follows, we briefly describe the polynomial-time solution for the path layout problem.

3.2 Polynomial-time Solution

Let us represent the delay and bandwidth constraint as follows

$$\widetilde{\mathfrak{D}}_k(u, v) = \left\lceil \frac{X_k \cdot \mathfrak{D}(u, v)}{D_k} \right\rceil, \quad \widetilde{\mathfrak{B}}_k(u, v) = \left\lceil \frac{X_k \cdot \mathfrak{B}_k(u, v)}{\widehat{B}_k} \right\rceil$$

where X_k is a given positive integer. For instance, if we relax the bandwidth constraint (*e.g.,* represent $\mathfrak{B}_k(\mathcal{P}_k)$ in terms of $\widetilde{\mathfrak{B}}_k(\mathcal{P}_k) = \sum_{(u,v)\in\mathcal{P}_k} \widetilde{\mathfrak{B}}_k(u, v)$), Eq. (5) can be rewritten as

$$\widetilde{\mathfrak{B}}_k(\mathcal{P}_k) \leq X_k. \tag{6}$$

Besides, the solution to this relaxed problem will also be a solution to the original MCP [10]. Likewise, if we relax the delay constraint, Eq. (3) can be rewritten as

$$\widetilde{\mathfrak{D}}_k(\mathcal{P}_k) = \sum_{(u,v)\in\mathcal{P}_k} \widetilde{\mathfrak{D}}_k(u, v) \leq X_k. \tag{7}$$

Let the variable $d_k[v, i]$ preserve an *estimate* of the path from s_k to t_k for $\forall v \in V$, $i \in \mathbb{Z}^+$. There exists a solution (*e.g.,* a path $\mathcal{P}_k$ from s_k to t_k) if *any* of the two conditions is satisfied when the *original MCP problem is solved by the heuristic.*

- *When the bandwidth constraint is relaxed:* The delay and (relaxed) bandwidth constraints, *e.g.,* $\mathfrak{D}_k(\mathcal{P}_k) \leq D_k$ and $\widetilde{\mathfrak{B}}_k(\mathcal{P}_k) \leq X_k$ are satisfied if and only if
$$d_k[t, i] \leq D_k, \quad \exists i \in [0, X_k] \wedge i \in \mathbb{Z}.$$
- *When the delay constraint is relaxed:* The (relaxed) delay and bandwidth constraints, *e.g.,* $\widetilde{\mathfrak{D}}_k(\mathcal{P}_k) = \sum_{(u,v)\in\mathcal{P}_k} \widetilde{\mathfrak{D}}_k(u, v) \leq X_k$ and $\mathfrak{B}_k(\mathcal{P}_k) \leq \widehat{B}_k$ are satisfied if and only if
$$d_k[t, i] \leq X_k, \quad \exists i \in [0, \widehat{B}_k] \wedge i \in \mathbb{Z}.$$

3.3 Algorithm Development

Let us consider MCP_HEURISTIC$(N, s, t, W_1, W_2, C_1, C_2)$ is an instance of polynomial-time heuristic solution to the MCP problem that finds a path $\mathcal{P}$ from s to t in any network N, such that the constraints $W_1(\mathcal{P}) \leq C_1$ and $W_2(\mathcal{P}) \leq C_2$ are satisfied[6]. Based on this MCP abstraction, we propose a path selection scheme considering delay and bandwidth constraints (refer to [19, Algorithm 2]). For each flow $f_k \in F$, starting with highest (*e.g.,* the flow with tighter delay requirement) to lowest priority, we first keep the delay constraint unmodified and relax the bandwidth constraint and solve MCP_HEURISTIC$(N, s_k, t_k, \mathfrak{D}_k, \widetilde{\mathfrak{B}}_k, D_k, X_k)$. If there exists a solution, the corresponding path $\mathcal{P}_k$ is assigned for f_k. However, if relaxing bandwidth constraint is unable to return a path, we further relax delay constraint keeping bandwidth constraint unmodified and solve MCP_HEURISTIC$(N, s_k, t_k, \widetilde{\mathfrak{D}}_k, \mathfrak{B}_k, X_k, \widehat{B}_k)$. If the path is not found after *both* of the relaxation steps, the flow-set is considered as *unschedulable* since it is not possible to assign a path for f_k such that both delay and bandwidth constraints are satisfied.

4. IMPLEMENTATION

We implement our prototype as an *application that uses the northbound API* for the Ryu controller [3]. The prototype application accepts the specification of flows in

[6] The heuristic solution of MCP problem is summarized in our extended report [19, Algorithm 1].

the SDN. The flow specification contains the classification, bandwidth requirement and delay budget of each individual flow. In this section, we describe how we realize a given flow f_k in the SDN.

4.1 Forwarding Intent Abstraction

An *intent* represents the *actions performed on a given packet at each individual switch.* Each flow f_k is decomposed into a set of intents. The number of intents that are required to express actions that the network needs to perform (for packets for a flow) is the same as the number of switches on the flow path. Each intent is a tuple given by $(\mathsf{Match}, \mathsf{InputPort}, \mathsf{OutputPort}, \mathsf{Rate})$. Here, Match defines the set of packets that the intent applies to, InputPort and OutputPort are where the packet arrives and leaves the switch and finally, the Rate is intended data rate for the packets matching the intent. In our implemented mechanism for laying down flow paths, each intent translates into a single OpenFlow [27] flow rule that is installed on the corresponding switch in the flow path.

4.2 Bandwidth Allocation for Intents

In order to guarantee bandwidth allocation for a given flow f_k, each one of its intents (at each switch) in the path need to allocate the same amount of bandwidth. As described above, each intent maps to a flow rule and the flow rule can refer to a meter, queue or both. However, meters and queues are precious resources and not all switch implementations provide both of them. As mentioned earlier (Section 2), there exist two alternative strategies for allocating bandwidth to individual intents: *(a)* individual queue/meter for each flow and *(b)* multiplexing meters and queues across multiple flows. The former strategy limits the rate of the queue/meter to B_k. It also places limits on the number of flows that a given switch can support at any point in time. On the other hand, as seen in Section 2.1, it guarantees much better isolation among flows. The second strategy (multiplexing) makes better use of the (limited) resources on each switch but cannot guarantee isolation without further changes to a switch's implementation. Hence, in our current implementation, we use the first strategy (one queue per flow) and leave the multiplexing idea for future work.

4.3 Intent Realization

Each intent is realized by installing a corresponding flow rule by using the northbound API of the Ryu controller. Besides using the intent's Match and OutputPort, these flow rules refer to corresponding queue and/or meter. If the meters are used, then they are also synthesized by using the controller API. However, OpenFlow does not support installation of queues in its controller-switch communication protocol, hence the queues are installed separately by interfacing directly with the switches by using a switch API or command line interface.

5. EVALUATION

In this section, we evaluate our proposed solutions using the following methods: *(a)* an exploration of the design space/performance of the path layout algorithm in Section 5.1 and *(b)* an empirical evaluation, using Mininet, that demonstrates the effectiveness of our end-to-end delay guaranteeing mechanisms even in the presence of other traffic in the network (Section 5.2).

5.1 Performance of the Path Layout Algorithm

Topology Setup and Parameters

In the first set of experiments we explore the design space (*e.g.,* feasible delay requirements) with randomly generated network topologies and synthetic flows. For each of the experiments we randomly generate a graph with 5 switches and create $f_k \in [1,5]$ flows. Each switch has 2 hosts connected to it. We assume that the bandwidth of each of the links $(u,v) \in E$ is 10 Mbps (*e.g.,* IEEE 802.3t standard [5]), and link delays are randomly generated within $[25, 125]$ μs. For each randomly-generated topology, we consider the bandwidth requirement as $B_k \in [1,5]$ Mbps, $\forall f_k$.

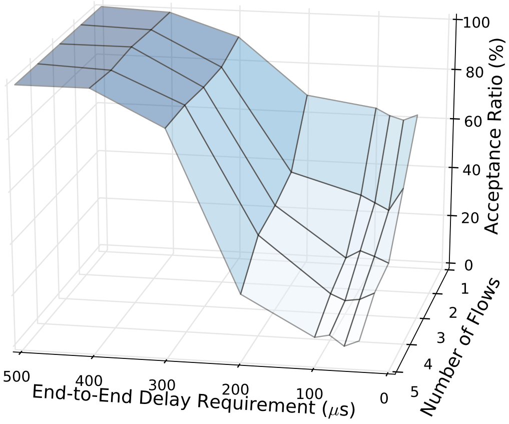

Figure 2: Schedulability of the flows in different network topology. For each of the (delay-requirement, number-of-flows) pair (*e.g.,* x and y-axis of the figure), we randomly generate 250 different topology. In other words, total $5 \times 5 \times 250 = 6250$ different topology were tested in the experiments.

Results

We say that a given network topology with set of flows is *schedulable* if all the real-time flows in the network can meet the delay and bandwidth requirements. We use the *acceptance ratio* metric (z-axis in Fig. 2) to evaluate the schedulability of the flows. *The acceptance ratio is defined as the number of accepted topology (e.g., the flows that satisfied bandwidth and delay constraints) over the total number of generated ones.* To observe the impact of delay budgets in different network topologies, we consider the end-to-end delay requirement D_k, $\forall f_k \in F$ as a function of the topology. In particular, for each randomly generated network topology G_i we set the minimum delay requirement for the highest priority flow as $D_{min} = \beta\delta_i$ μs and increment by $\frac{D_{min}}{10}$ for each of the remaining flows where δ_i is the diameter (*e.g.,* maximum eccentricity of any vertex) of the graph G_i in the i-th spatial realization of the network topology, $\beta = \frac{D_{min}}{\delta_i}$ and D_{min} represents x-axis values of Figure 2. For each (delay-requirement, number-of-flows) pair, we randomly generate 250 different topologies and measure the acceptance ratios. As Figure. 2 shows, stricter delay requirements (*e.g.,* less than 300 μs for a set of 5 flows) limit the schedulability (*e.g.,* only 60% of the topology is schedulable). Increasing the number of flows limits the available resources (*e.g.,* bandwidth) and thus the algorithm is unable to find a path that satisfies the delay requirements of all the flows.

5.2 Experiment with Mininet Topology: Demonstrating that the End-to-End Delay Mechanisms Work

Experimental Setup

The purpose of the experiment is to evaluate whether our controller rules and queue configurations can provide isolation guarantees so that the real-time flows can meet their delay requirement in a practical setup. We evaluate

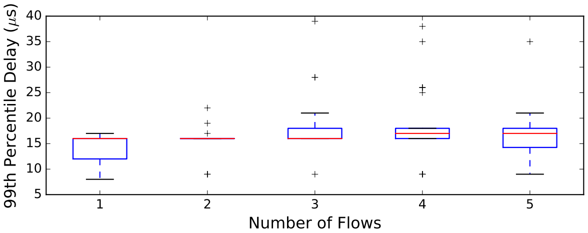

Figure 3: End-to-end round-trip 99^{th} percentile delay with varying number of flows. For each set of flow $f_k \in [1, 5]$, we examine $f_k \times 25 \times 5$ packet flows (each for 10 seconds).

the performance of the proposed scheme using Mininet [20] (version 2.2.1) where switches are configured using Open vSwitch [2] (version 2.3.0). We use Ryu [3] (version 4.7) as our SDN controller. For each of the experiments we randomly generate a Mininet topology using the parameters described in Section 5.1. We develop flow rules in the queues to enable connectivity among hosts in different switches. The packets belonging to the real-time flows are queued separately in individual queues and each of the queues are configured at a maximum rate of $B_k \in [1, 5]$ Mbps. If the host exceeds the configured maximum rate of B_k, our ingress policing throttles the traffic before it enters the switch[7]. To measure the effectiveness of our prototype with mixed (*e.g.,* real-time and non-critical) flows, we enable $[1, 3]$ non-critical flows in the network. Our flow rules isolate the non-critical flows from real-time flows. All of the low-criticality flows use a *separate, single queue* and are served in a FIFO manner – it is the "default" queue in OVS.

We use `netperf` (version 2.7.0) [6] to generate the UDP traffic between the source and destination for any flow f_k. Once the flow rules and queues are configured, we triggered packets starting at the source s_k destined to host t_k for each of the flows f_k. The packets are sent at a burst of 5 with 1 ms inter burst time. For each set of flows $f_k \in \{2, 5\}$, we randomly generate 25 different network topology. For each topology, we randomly generate the traffic with required bandwidth $B_k \in [1, 5]$ Mbps and send packets between source (s_k) and destination (t_k) hosts for 5 times and log the worst-case round-trip delay experienced by any flow. All packet flows are triggered simultaneously and last for 10 seconds.

Experience and Evaluation

We define the *expected delay bound* as the expected delay if the packets are routed through the diameter of the topology and given by $\mathfrak{D}_i(u, v) \times \delta_i$ and bounded by $[25\delta_i, 125\delta_i]$ where $\mathfrak{D}_i(u, v) \in [25, 125]$ is the delay between the link (u, v) in i-th network realization.

In Fig. 3 we illustrate the 99^{th} percentile round-trip delay (represents the y-axis in the figure) with different number of flows (x-axis). From our experiments we find that, the non-critical flows *do not* affect the delay experienced by the real-time flows and the 99^{th} percentile delay experienced by the real-time flows *always* meet their delay requirements. This is because our flow rules and queue configurations isolate the real-time flows from the non-critical traffic to ensure that the end-to-end delay requirements are satisfied.

As shown in Figure 3, increasing the number of flows decreases quality of experience (in terms of end-to-end delays). With increasing number of packet flows the switches are simultaneously processing forwarding rules received from the controller – hence, it increases the round-trip delay. Recall that the packets of a flow are sent in a bursty manner using `netperf`. Increasing number of flows in the Mininet topology increases the packet loss and thus causes higher delay.

For our experiments with Mininet and `netperf` generated traffic, we do *not* observe any instance for which a set of schedulable flow misses its deadline (*i.e.,* packets arriving *after* the passing of their end-to-end delay requirements). Thus, based on our empirical results and the constraints provided to the path layout algorithm, we can assert that the schedulable real-time flows will meet their corresponding end-to-end delay requirements.

6. DISCUSSION

Despite the fact that we provide an initial approach to leverage the benefits of the SDN architecture to guarantee end-to-end delay in safety-critical hard RTS, our proposed scheme has some limitations and can be extended in several directions. To start with, we intend to validate our implementations in actual hardware switches with more complex topology and analyze the worst-case latency experienced by the flows. Furthermore, most hardware switches limit the maximum number of individual queues[8] that can be allocated to flows. Our current intent realization mechanism reserves one queue per port for each Class I flow. This leads to depletion of available queues. Hence, we need smarter methods to *multiplex* Class I flows through limited resources and yet meet their timing requirements. Our future work will focus on developing sophisticated schemes for ingress/egress filtering at each RT-SDN-enabled switch. This will also help us better identify the properties of each flow (priority, class, delay, *etc.*) and then develop scheduling algorithms to meet their requirements.

In this work we allocate separate queues for each flow and layout paths based on the "delay-monotonic" policy. However establishing and maintaining the flow priority is *not* straightforward if the ingress policing requires to share queues and ports in the switches. Many existing mechanisms to enforce priority are available in software switches (*e.g.,* the hierarchical token buckets (HTB) in Linux networking stack). In our experience, enabling priority on hardware switches has proven difficult due to firmware bugs.

7. RELATED WORK

There have been several efforts to study the provisioning a network such that it meets bandwidth and/or delay constraints for the traffic flows. For synthesis, the NP-Complete MCP comes close and Shingang *et al.* formulated a heuristic algorithm [10] for solving MCP. We model our delay and bandwidth constraints based on their approach. Azodolmolky *et al.* proposed a Network Calculus-based model [8] for a single SDN switch that provides an upper bound on delays experienced by packets as they cross through the switch. Guck *et al.* used mixed integer program (MIP) based formulation [12] for provisioning end-to-end flows that provide delay guarantees – they do not provide a solution of what traffic arrival rate to allocate for queues on individual switches for a given end-to-end flow.

Avionics full-duplex switched Ethernet (AFDX) [4, 9] is a deterministic data network developed by Airbus for safety critical applications. Though such protocols aim to guarantee deterministic QoS through static routing, reservation and isolation, it imposes several limitations on optimizing the path layouts and on different traffic flows. With SDN architectures and a flexible QoS framework proposed in this paper, one could easily configure COTS components and meet QoS guarantees with optimized path layouts and backup paths. There have been studies towards evaluating the upper bound on the end-to-end delays in AFDX networks [9]. The evaluation seems to depend

[7] In real systems, the bandwidths allocation would be overprovisioned (as mentioned earlier), our evaluation takes a conservative approach.

[8] For example, HPE FlexFabric 12900E switch supports at most 8 queues.

on the AFDX parameters though. There are several protocols proposed in automotive communication networks such as controller area network (CAN) [15] and FlexRay [1]. These protocols are designed to provide strong real-time guarantees but have limitations in how to extend it to varied network lengths, different traffic flows and complex network topologies.

Heine *et al.* proposed a design and built a real-time middleware system, CONES (COnverged NEtworks for SCADA) [13] that enables the communication of data/information in SCADA applications over single physical integrated networks. However, the authors did not explore the synthesis of rules or path optimizations based on bandwidth-delay requirements – all of which are carried out by our system. Qian *et al.* implemented a hybrid EDF packet scheduler [26] for real-time distributed systems. Though these strategies were adopted to make the system more predictable, due to partial information of the network, it leaves lots of scope for improvements. The problem of end-to-end delay bounding in RTS is addressed in literature [18]. The proposed approach requires modification to the switches. Besides the authors do not consider the bandwidth limitations, variable number of flows and flow classifications (real-time vs non-real-time). There is a lot of work in the field of traditional real-time networking (too many to enumerate here) but the focus on SDNs is what differentiates our work.

8. CONCLUSION

With the proliferation of COTS components, designers are exploring new ways of using them, even in critical systems (such as RTS). Hence, there is a need to understand the inherent trade-offs (less customization) and advantages (lower cost, scalability, better support and more choices) of using COTS components in the design of such systems. In this paper, we presented mechanisms that provide end-to-end delays for critical traffic in real-time systems using COTS SDN switches. Hence, future RTS can be better managed, less complex (fewer network components to deal with) and more cost effective.

9. REFERENCES

[1] FlexRay Automotive Communication Bus Overview.

[2] Production quality, multilayer open virtual switch. http://openvswitch.org/.

[3] Ryu controller. http://osrg.github.io/ryu/. Accessed: 2014-11-01.

[4] ARINC Specification 664, Part 7, Aircraft Data Network, Avionics Full Duplex Switched Ethernet (AFDX) Network. 2003.

[5] IEEE Standard for Ethernet. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pages 1–3747, Dec 2012.

[6] The netperf homepage, 2016.

[7] A. Aydeger, K. Akkaya, M. H. Cintuglu, A. S. Uluagac, and O. Mohammed. Software defined networking for resilient communications in Smart Grid active distribution networks. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[8] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou. An analytical model for software defined networking: A network calculus-based approach. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 1397–1402. IEEE, 2013.

[9] H. Charara, J. L. Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10 pp.–202, 2006.

[10] S. Chen and K. Nahrstedt. On finding multi-constrained paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, volume 2, pages 874–879. IEEE, 1998.

[11] C. M. Fuchs. The evolution of avionics networks from ARINC 429 to AFDX. *Innovative Internet Technologies and Mobile Communications (IITM), and Aerospace Networks (AN)*, 65, 2012.

[12] J. W. Guck and W. Kellerer. Achieving end-to-end real-time quality of service with software defined networking. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 70–76. IEEE, 2014.

[13] E. Heine, H. Khurana, and T. Yardley. Exploring convergence for SCADA Networks. In *ISGT 2011*, pages 1–8, Jan 2011.

[14] P. Heise, F. Geyer, and R. Obermaisser. Deterministic openflow: Performance evaluation of SDN hardware for avionic networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 372–377. IEEE, 2015.

[15] N. Instruments. Controller Area Network (CAN) Overview.

[16] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14(1):95–116, 1984.

[17] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[18] D. Jin, J. Ryu, J. Park, J. Lee, H. Shin, and K. Kang. Bounding end-to-end delay for real-time environmental monitoring in avionic systems. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 132–137, March 2013.

[19] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba. Dependable end-to-end delay constraints for real-time systems using SDNs. *arXiv preprint*, 2017. https://arxiv.org/pdf/1703.01641v1.pdf [Online].

[20] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[21] D. Levin, M. Canini, S. Schmid, and A. Feldmann. Incremental sdn deployment in enterprise networks. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 473–474. ACM, 2013.

[22] Z. Li, Q. Li, L. Zhao, and H. Xiong. Openflow channel deployment algorithm for software-defined afdx. In *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, pages 4A6–1. IEEE, 2014.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[24] S. Oh, J. Lee, K. Lee, and I. Shin. RT-SDN: Adaptive routing and priority ordering for software-defined real-time networking. 2015.

[25] T. Pfeiffenberger and J. L. Du. Evaluation of software-defined networking for power systems. In *Intelligent Energy and Power Systems (IEPS), 2014 IEEE International Conference on*, pages 181–185. IEEE, 2014.

[26] T. Qian, F. Mueller, and Y. Xin. Hybrid EDF packet scheduling for real-time distributed systems. In *2015 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 37–46, July 2015.

[27] O. S. Specification-Version. 1.4. 0, 2013.

[28] J. Spencer, O. Worthington, R. Hancock, and E. Hepworth. Towards a tactical software defined network. In *Military Communications and Information Systems (ICMCIS), 2016 International Conference on*, pages 1–7. IEEE, 2016.