Safety Critical Networks using Commodity SDNs

Ashish Kashinath*§, Monowar Hasan†§, Rakesh Kumar*, Sibin Mohan*, Rakesh B Bobba‖ and Smruti Padhy¶
*University of Illinois at Urbana-Champaign, †Wichita State University, ‖Oregon State University,
¶University of Texas at Austin. *{ashishk3,sibin}@illinois.edu

***Abstract*—Safety-critical networks often have stringent real-time requirements; they must also be resilient to failures. In this paper, we propose the RealFlow framework that uses commodity software-defined networks (SDNs) to realize networks with end-to-end timing guarantees, while also: *(a)* increasing resiliency against link/switch failures and *(b)* increasing network utilization. The use of SDNs in this space also improves the management capabilities of the system due to the global visibility into the network. RealFlow is implemented as a northbound SDN controller application compatible with standard OpenFlow protocols with *little to no runtime overheads*. We demonstrate feasibility on a real hardware testbed (Pica8 SDN switches+Raspberry Pi endhosts) and a practical avionics case study. Our evaluations show that RealFlow can accommodate 63% more network flows with safety-critical guarantees when compared to current designs and up to 18% when link resiliency (via backup paths) is also considered.**

I. Introduction

Network flows in safety-critical systems often have stringent timing and performance constraints often referred to as Real-Time (RT) requirements. If important messages are either not delivered or arrive late, the system could fail. For example, if a sensor on the bumper of a car detects impact then that information must be relayed to an electronic control unit (ECU) inside the car within 20 ms [1] in order to initiate airbag deployment. Delays anywhere in this chain of processes could result in serious injury to the driver and/or the passengers. In addition to timing constraints, flows can also have distinct priorities and isolation requirements. Current designs of safety-critical networks have two key issues:

- They use *conservative/over-engineered, domain-specific solutions* such as Avionics Full-Duplex Switched Ethernet (AFDX) [2] or Controller Area Network (CAN) [3] *i.e.,* proprietary, legacy, protocols and hardware.
- They incur *high infrastructure & management overheads* since designers use multiple, physically separate, networks for RT/non-RT traffic; this leads to expensive designs that are often shackled with legacy requirements.

While efforts are underway to standardize quality-of-service (QoS) provisioning for RT flows over Ethernet (*e.g.,* TSN: QoS IEEE 802.1Qav, IEEE 802.1Qbv [4]), commodity hardware supporting these standards is not yet widely available[1].

This paper is supported in part by U.S. Department of Energy (DoE) award DE-OE0000780 and 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. We also thank Prof. Radhika Mittal, UIUC, for her detailed feedback.

§The authors contributed equally to this work.

[1]As opposed to our techniques that can be implemented right now using multiple, commercially-available SDN switches.

Software Defined Networking (SDN) [5], which provides 'global' visibility into the network and enables centralized resource allocation, has seen quick adoption since its emergence; *e.g.,* enterprise systems, cloud computing services, military networks, power systems [6], [7], among others. While most SDN applications were aimed at enabling highly dynamic and adaptive networks, there is an argument for using SDNs in safety-critical networks *for predictability and determinism* [8]. Further, availability of open standards (OpenFlow, Open vSwitch) offer opportunities for realizing *predictable* RT-aware SDN networks using commodity hardware.

An early attempt at leveraging the global visibility of SDN for safety-critical systems [9] combined static path allocation with dynamic priority scheduling (*viz.,* Non-preemptive Earliest-Deadline First (EDF)). However, their approach required changes to scheduling disciplines in the switches and is not compatible with commodity off-the-shelf (COTS) SDN switches. Our early work [10] used static path allocation and fixed-priority scheduling to make commodity SDNs "*real-time aware*" *i.e.,* designing for applications with hard timing constraints. However, this work under-utilizes network resources by dedicating one queue per RT flow (see comparison in §IV-A). Neither work addresses link failures. Recent work [11], [12] attempts to tackle this problem. However, they require extra functionality at the switch level that is as yet unavailable in COTS switches. Further, Lee *et al.*'s approach [12] could lead to a large number of dropped packets and an inordinate growth in delays that directly correlates to a growth in the size of the network or an increase in the number of flow sets (see §IV-A). The shortcomings of these approaches make it impractical to use commodity SDNs for modern, complex safety-critical networks that typically consist of large numbers of subsystems with diverse flows requirements (see Table II).

We present *RealFlow*, a framework that enables the use of commodity SDNs in modern safety-critical applications; system utilization is significantly improved and timing guarantees are retained in the presence of link failures, all while using COTS components. Specifically, we, *(a)* present a new static *path (route) allocation algorithm* that can *multiplex RT flows* onto the same queues while still meeting the end-to-end delay and bandwidth requirements (thus improving the efficiency of the network [§III-C]) and *(b)* develop algorithms and mechanisms to *pre-compute and deploy backup paths for critical flows* to increase system resiliency in the presence of link failures [§III-D]. Multiplexing flows onto the same queue while still providing delay guarantees is non-trivial since dependencies across flows that share the same queue complicate the analysis.

We implemented *(a)* and *(b)* as a path allocation algorithm compatible with existing packet scheduling schemes in COTS switches. We developed a *latency-aware (deadline-aware) fast failover* mechanism for *(b)* — using our algorithms on top of the built-in mechanisms — ensuring that if the original path guaranteed that an RT flow met its deadlines then the backup path, activated on failure of the original path, guarantees the same. Note that SDN does not intrinsically lend itself to RT applications due to its lack of timing awareness.

We demonstrate, using correct-by-construction analysis as well as empirical evaluation on real hardware, that *(i)* RT flows that use our approach meet their end-to-end delay requirements even in the presence of a large volume of non-critical or non-RT traffic and *(ii)* our backup path mechanisms ensure that critical flows will meet their deadlines even when links fail. Evaluation [§IV] *(a)* using an avionics case study [13] on *actual hardware* that mimics an embedded RT system along with *(b)* synthetic topologies/simulations shows that our approach works over a large design space.

TABLE I
DESIGN SPACE OF SDN-BASED RT TECHNIQUES. COLUMN TITLES: **COTS**→CAN WORK ON COMMODITY COMPONENTS. **S**→CAN SCALE WITH # OF FLOWS; **FT**→ LINK/SWITCH FAILURE TOLERANCE BUILT IN.

Technique	COTS	S	FT
Kumar *et al.* [10]	✓	×	×
Qian *et al.* [9]	×	✓	×
Qian *et al.* [11]	×	✓	✓
Lee *et al.* [12]	×	×	✓
RealFlow	✓	✓	✓

Contributions: In summary, our main contributions are:

1) Static path allocation algorithms that can achieve higher network utilization by multiplexing multiple RT flows onto the same queues while still meeting the end-to-end delay and bandwidth constraints (§III-A – §III-C).
2) Deadline-aware back-up path provisioning for resiliency against link failures (§III-D).
3) Implementation and evaluation of our schemes on SDN-enabled COTS switches (§IV).

We first present the system model, formal definition of the problem, and a high-level description of our solution in §II.

II. System Model of a Safety-Critical Network

Safety-critical 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.,* Airbus has complete control over A380 topology and its associated subsystems). 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 and link failures.

A. System Architecture and Objectives

RT applications generate periodic traffic flows that arrive at the switch and are multiplexed into queues based on priority (§II-B). A flow needs to traverse a sequence of switches, *i.e.,* a hitherto unknown path $\mathcal{P}_i$, so that QoS constraints such as delay and bandwidth requirements can be met. We consider two types of paths: *(a)* a primary (or the default) path for each flow for use during normal operation and *(b)* a backup path for certain critical flows for use in the event of link failures. When link failures are detected, critical flows can be automatically rerouted through their corresponding backup path, and still meet deadlines, without needing network controller intervention, using the fast fail-over mechanism.

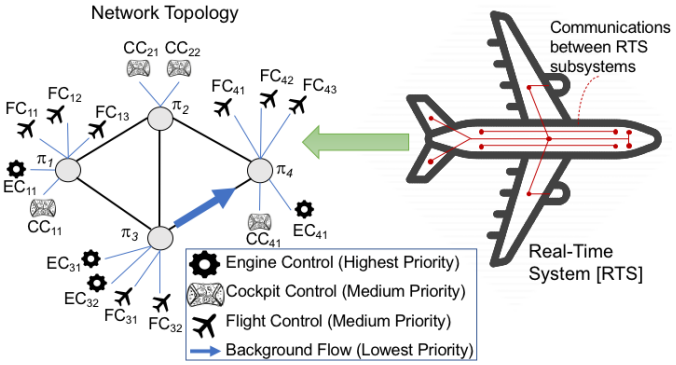


Fig. 1. Illustrative Avionics Example (also used in Evaluation in §IV-C).

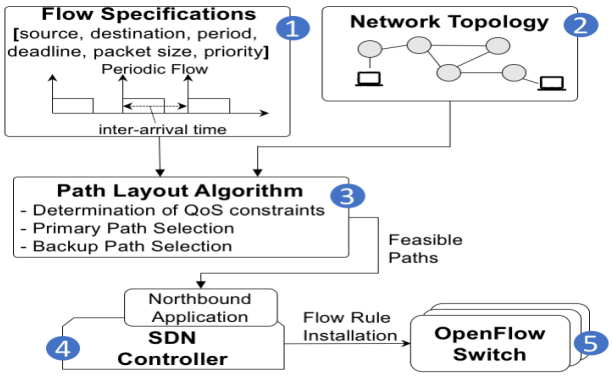


Fig. 2. **System design of RealFlow**.An overview of our proposed solution (the Avionics System is, notionally, used for evaluation). We extract the following from a Real-Time System: (1) Flow Specifications (flows of different priorities) and (2) Network Topology are input into "RealFlow"(3) along with any QoS and bandwidth constraints. A set of feasible paths (that meet the given constraints) is generated for both – primary as well as *backup* paths for the critical flows. These paths are then sent to a northbound application on a SDN Controller (4) installs flow rules to the switches (5) so that flows can meet their end-to-end guarantees and also have increased resiliency (due to the existence of backup paths in the event of failure).

RealFlow. To achieve these objectives on safety-critical networks, a high-level overview of our solution, "RealFlow", is illustrated in Fig. 2. It consists of:

1) Flow Specifications *i.e.,* the RT application's network requirements. These are *specific* to every application deployed on the network.
2) Network Topology or the switch graph *i.e.,* a representation of the network consisting of links and switches (does not include the end-hosts).
3) A Path Layout Algorithm that takes the network topology and flow specifications[2] as input and finds a 'feasible path' (primary and backup) for each RT flow, satisfying given end-to-end deadline/bandwidth requirements.

[2]Based on application requirements and given by the application designers.

TABLE II
PRACTICAL CONSIDERATIONS IN SAFETY-CRITICAL NETWORKS GATHERED FROM CERTIFICATIONS SPECIFICATIONS, ACADEMIC & INDUSTRY COLLABORATIONS.

Aspects	**Automotive Network [14]**	**Power Grid Substation Network [15]**	**Avionics Network [2], [13]**
Flow Rates	$\approx$ 100 Kbps (Control traffic), $\approx$ 10 Mbps (Data traffic)	$\approx$ 40 Kbps (IEC-61850 GOOSE), $\approx$ 5.4 Mbps (IEC-61850 SV)	0.5 - 4 Mbps
Flow Deadlines	$<$ 10 ms (Control), 10 ms (ADAS), 30 ms (Vision)	$<$ 3 ms (GOOSE Fast Trips & Blocks; SV Measurements)	$\approx$ 1 ms (between 2 end-systems on same switch)
Network Link Rates	100 Mbps - 1 Gbps	1 - 5 Gbps	100 Mbps
Scale	41 flows, 5 switches	900 flows, 70 switches	327 flows, 9 switches
Example Hardware	NXP SJA1105 Ethernet Switch Series, 10-16 Priority Queues	SEL-2740S, 4 Priority Queues, IEC-61850 standards compliant	TTTech A664 AFDX Switch 100 Mbps - 1 Gbps, 8 Priority Levels
Common characteristics of RT networks: *(i)* **Well-defined Network structure** (*viz.,* topologies, hosts and links), *(ii)* **Well-defined Flow specifications** from certification/industry standards, *(iii)* **Self-contained Networks** managed by a single authority			

4) An SDN Controller that takes the output of the path layout algorithm and translates it into forwarding rules (including those for *fast failover*) for OpenFlow switches.
5) OpenFlow switches (*i.e.,* SDN switches) that accept flow rules from the SDN controller and forward traffic from one host to the other, using preconfigured rules.

B. Formal Problem Definition

Consider an SDN topology (N) with OpenFlow switches, a controller and a specified set of RT flows $\mathcal{F}$ (generated from RT applications) with fixed maximum end-to-end delays and bandwidth guarantee requirements. We characterize each flow $F_i \in \mathcal{F}$ by the tuple $F_i = (src_i, dst_i, T_i, D_i^{max}, pkt_i, pri_i)$ where src_i and dst_i are the source and destination host, $T_i \in \mathbb{Z}^+$ is the inter-arrival time ('period'), D_i^{max} is the end-to-end delay bound ('deadline'), pkt_i is the size of the packet in bits (*e.g.,* 1 kB packet = 8 kbits) and pri_i is the priority. $B_i = \frac{1}{T_i} \times packet\_size_i$ denotes the bandwidth requirement of F_i. We model the network as an undirected graph $N(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes, each node representing a switch π_s and $\mathcal{E}$ is set of the edges, each representing a possible path from one switch (π_s) to another ($\pi_{s'}$). Additionally, $\mathcal{F}' \subseteq \mathcal{F}$ denotes a subset of critical flows for which alternative back-up routes need to be configured for resiliency against link failures.

Queues and Priorities: Each switch $\pi_i \in \mathcal{V}$ consists of a set of $L+1$ priority queues. Among $L+1$ queues we use the L highest priority queues for allocating *primary paths*[3] and use the remaining lowest priority queue for providing backup paths. We assume that flow priorities are selected from $\mathcal{L} = \{0, 1, \cdots, L-1\}$ predefined distinct priority-levels ($pri_i \in \mathcal{L}$ and level-0 is the highest priority) where flows with priority-level $l \in \mathcal{L}$ will be assigned to $l+1$-th queue in each switch $\pi_s \in \mathcal{V}$. These priorities are often derived from the actual applications and decided by the system designers [16].

We assume that applications (and hence network flows they generate) have *statically assigned priorities* – this is typical of RT systems today. Note that multiple flows can share the same priority-level (*i.e.,* $\exists F_i \neq F_j : pri_i = pri_j$) and flows with same priority-levels l will be *multiplexed* to the $l+1$-th queue. Further, a subset of flows $\mathcal{F}'$ for which designers want to provide alternate routes due to link failure will be multiplexed to lowest priority queue ($L+1$-th) to provide backup paths.

[3]We use '*path*' and '*primary path*' interchangeably throughout the paper.

C. Assumptions

Target System: We design our framework for a self-contained RT system with fixed priority, static flows that can tolerate the loss of a limited number of packets — loss of a bounded number of packets is acceptable in many safety critical networks [17], [18]. Large-scale, open distributed systems with dynamic or stochastic flows are not the focus of this work.

System Model Assumptions: We assume that queue sizes are large to accommodate the RT flow packets. In practice, switches do have finite queue sizes. For example, in Pica8 P-3297 [19], we have 8 queues per port, each queue being 250kB in size, giving a total queue size of 2MB per port. Despite this, our system model can be applied in practice due to the following two reasons: *(a)* the rate of RT flows in practice is substantially lower than link capacities and *(b)* our approach can over-provision bandwidth (see §III-B).

Backup paths are pre-computed *offline* for all potential link failures of interest. Network designers can consider all link failures or focus on specific links that are critical/prone to failures. We only discuss single link failure tolerance for illustration purposes; our approach can handle simultaneous link failures without loss of generality [see §III-D]. However, there is a trade-off for supporting multiple simultaneous link failures since resources must be provisioned for multiple potential backup paths. Input buffering is not common in COTS switches (*e.g.,* Pica8 [19]). End-hosts, in our experiments, are simply traffic sources or sinks for our test harness.

III. REALFLOW– A QOS-AWARE PATH SELECTION AND CONFIGURATION FRAMEWORK

Deadline and bandwidth-aware path selection is the core of RealFlow. We discuss the delay constraints and present our path selection backup path-based resiliency mechanisms.

A. Delay Constraints

Each flow will experience the following delays as it is routed along a network path: *(a) Transmission and propagation delay* – this depends on the data rate supported by the link as well as the speed of signal transmission (for a given link capacity, material and length, we can define a constant upper-bound for this delay) and *(b) queuing and processing delays* at each switch along the path. Queuing and processing delays comprise the following delay components: *(i) FIFO queuing delay* from flows at the same priority-level that are routed through the

same switch along the flow path; *(i) Interference delay* due to interference from higher-priority flows that share the same egress port at a switch on the path; and *(iii) Blocking delay* due to the non-preemptive nature of packet transmission.

1) Transmission and propagation delay: Every flow $F_i \in \mathcal{F}$ experiences a transmission delay, $D^t_{s',s} = \frac{pkt_i}{\text{data rate}}$. Additionally, every flow also has the signal propagation delay due to the medium, $D^p_{s',s} = \frac{\text{Length of the link}}{\text{velocity of signal in medium}}$. Let us denote by $D^{tp}_{s',s}$, the total packet transmission and propagation delay incurred on the edge $(\pi_{s'}, \pi_s) \in \mathcal{E}$ (*i.e.,* the link between $\pi_{s'}$ and π_s). This is given by $D^{tp}_{s',s} = D^t_{s',s} + D^p_{s',s}$.

2) Queuing and processing delay due to FIFO delays from flows at the same priority: Flows with the same priority-level l will be enqueued in the same $l+1$-th queue (if the bandwidth requirements can be met) at each switch π_s and processed in a first in, first out (FIFO) manner. Multiplexing the flows with same priority into a single queue can accommodate more flows than there are queues (given their constraints are satisfied).

We can calculate the queuing delay experienced by each packet of flow F_i due to the interference from other flows at the same priority level (*i.e.,* $\forall F_j \neq F_i \mid pri_j = pri_i$) routed through the switch π_s. Let us denote $\tilde{\mathcal{F}}^l_s \subset \mathcal{F}$ as the set of flows with priority-level l routed through the switch π_s (*i.e.,* each flow $F_i \in \tilde{\mathcal{F}}^l_s$ shares the same queue ψ_l in π_s). For a given flow $F_i \in \tilde{\mathcal{F}}^l_s$ we can calculate the worst-case queuing delay as [20]: $q^s_i = \sum_{F_k \in \tilde{\mathcal{F}}^l_s} \left\lceil \frac{T_i}{T_k} \right\rceil \hat{d}_k$ where $\hat{d}_k$ is the per-packet processing delay. This happens when flows $F_k \neq F_i$ are scheduled before F_i (assuming the switch π_s will arbitrarily schedule one of the flows when packets of multiple flows with same priority-level arrive at the same time). For example, consider two flows F_1 ($T_1 = 10$) and F_2 ($T_2 = 5$), *i.e.,* F_2 arrives at a faster rate, then in the worst-case F_1 will experience FIFO queuing delay from $\lceil \frac{T_1}{T_2} \rceil = 2$ packets of F_2 within one period of F_1.

3) Queuing and processing delay from interference and blocking: Switches process each packet in order, based on arrival time and priority. Packets that arrive later or belonging to flows with lower priority are processed later. Hence, for any flow F_i, we need to consider the interference from other flows F_j routed through the same switch.

Lower-priority flows will suffer interference from higher-priority flows since the switch scheduler follows a strict priority scheme where flows with higher priority are dequeued first. Let $hp(\tilde{\mathcal{F}}^l_s)$ denote the set of flows with priority higher than priority-level l and are routed through switch π_s. We can estimate the upper bound of interference from higher priority flows using traditional response-time analysis [21] as follows : $I^s_i = \sum_{F_h \in hp(\tilde{\mathcal{F}}^l_s)} \left\lceil \frac{T_i}{T_h} \right\rceil \hat{d}_h$, where I^s_i is the interference experienced by each flow $F_i \in \tilde{\mathcal{F}}^l_s$.

Since the switch processes the flows in a non-preemptive manner, a given flow may also experience at most one packet delay from lower-priority flows in the worst case. We bound this blocking delay from low-priority flows as $\beta^s_i = \max_{F_k \in \tilde{\mathcal{F}}^l_s, k \neq i} \hat{d}_k$. Hence, total queuing and processing delay (presented in §III-A2 and §III-A3) for F_i at the switch π_s, Q^s_i can be expressed as:

$$Q^s_i = \text{FIFO delay + Interference delay + Blocking delay} \tag{1}$$

Let $\mathfrak{D}_i(\mathcal{P}_i)$ denote the *total delay* for F_i considering *(a)* and *(b)* . To ensure that the delay requirement of each F_i is satisfied this total end-to-end delay should be less than its delay bound, *i.e.,* $\mathfrak{D}_i(\mathcal{P}_i) \leq D^{max}_i$.

Total Delay — Queuing Delay

$$\mathfrak{D}_i(\mathcal{P}_i) = \sum_{(\pi_{s'},\pi_s) \in \mathcal{P}_i} D^{tp}_{s',s} + \sum_{\pi_s \in \mathcal{P}_i} Q^s_i \tag{2}$$

Trans. & Prop. Delay

B. Bandwidth Constraints

Every flow consumes resources that is quantified by its *bandwidth utilization*, defined as the ratio of bandwidth requirement to the bandwidth available on the link. We denote the bandwidth utilization for a given link $(\pi_{s'}, \pi_s)$ as $\mathfrak{B}_i(\pi_{s'}, \pi_s)$. A flow can be assigned to a path only if its bandwidth utilization is less than 1 on every edge on the path. We define the *total bandwidth utilization* (denoted as $\mathfrak{B}_i(\mathcal{P}_i)$) for F_i over a path $\mathcal{P}_i$ as the bandwidth utilization for every edge in $\mathcal{P}_i$. In prior work [10], researchers show that the bound on bandwidth utilization over a path can be represented as a function of link bandwidth and network size, *i.e.,* $\max_{(\pi_{s'},\pi_s) \in \mathcal{E}} \mathfrak{B}_i(\pi_{s'}, \pi_s)|\mathcal{V}|$ where $|\mathcal{V}|$ is the cardinality of a set of switches for a given network. Therefore, to satisfy the bandwidth requirement, we define (in a conservative manner) the following constraint on bandwidth utilization for each flow F_i: $\mathfrak{B}_i(\mathcal{P}_i) \leq \max_{(\pi_{s'},\pi_s) \in \mathcal{E}} \mathfrak{B}_i(\pi_{s'}, \pi_s)|\mathcal{V}|$.

C. Path Layout

A path is *feasible* for F_i if it is able to fulfill the QoS guarantees (*i.e.,* delay and bandwidth constraints) for that flow and the network system is considered *schedulable* if it *admits all* flows, *i.e.,*, all flows in the network have a feasible path. The deadline and bandwidth-constrained path layout problem can be formalized as a multi-constrained path (MCP) problem that is NP-Complete [22], [23]. To our knowledge there is no direct method to obtain the paths (primary and backup) for our problem and existing routing heuristics (refer to surveys *e.g.,* [24]) are *not* directly applicable since they are not designed with RT requirements in mind. Therefore, we develop a low-complexity heuristic solution for calculating flow paths.

As observed in prior work [10], the end-to-end delays are lower when the flows experience less queuing interference (each RT flow has its own queue). The goal of our path selection algorithm here is therefore to find a path $\mathcal{P}_i$ for each flow F_i that leads to less interference from other flows. We achieve this by spatially separating the flows across the network (while satisfying QoS requirements) so that the interference at each switch can be reduced. For this, we define a metric (called *interference index*) to find the paths with minimal contention (*i.e.,* less interference from other flows). In particular, we define the interference index of a given path $\mathcal{P}_i$ (denoted as $\text{II}(\mathcal{P}_i)$)

Algorithm 1 Delay and Bandwidth-Aware Path Selection

For each flow $F_i \in \mathcal{F}$, generate all possible loop free paths (called candidate paths) using standard graph theoretical approaches [25].
while some flow has more than one candidate path **do**
  Find the path $\mathcal{P}_{max}$ with highest interference. Let the flow F_α uses highest interference path.
  if $\mathcal{P}_{max}$ is the only path for F_α **then**
    / This is the only available path for F_α */*
    Assign the path $\mathcal{P}_{max}$ for F_α
  else
    / Remove the path with maximum interference */*
    Remove $\mathcal{P}_{max}$ from candidate path set
    Recalculate interference for remaining paths (that intersects with $\mathcal{P}_{max}$) in the candidate path set
  end if
end while
/ Unable to find a path that respects QoS constraints */*
if $\exists F_i \in \mathcal{F}$ with path $\mathcal{P}_i$ such that $\mathfrak{D}_i(\mathcal{P}_i) > D_i^{max}$ **or** $\mathfrak{B}_i(\mathcal{P}_i) > \max_{(\pi_{s'},\pi_s)\in\mathcal{E}} \mathfrak{B}_i(\pi_{s'},\pi_s)|\mathcal{V}|$ **then**
  return Unschedulable
end if
/ Otherwise flows in the system are schedulable */*
return the paths for all flows

as a function of available slack time (*i.e.,* $\mathfrak{D}_i(\mathcal{P}_i) - D_i^{max}$) and the total bandwidth utilization $U(\mathcal{P}_i)$:

$$\Pi(\mathcal{P}_i) \stackrel{\text{def}}{=} \left(\mathfrak{D}_i(\mathcal{P}_i) - D_i^{max}\right) + [U(\mathcal{P}_i)]^+, \quad (3)$$

Interference Index; Bandwidth Utilization; Delay of F_i on P_i; Deadline of F_i

where the operator $[x]^+ = x$ if $x \geq 0$, $\bar{x}^+$ otherwise, where $\bar{x}^+$ is an arbitrary large non-negative number (*i.e.,* the contention in a path will be higher when there is less slack and more flows are trying the access the links in path than the available link bandwidth). This metric provides us a way to quantify the interference experienced by a flow (and measure QoS constraints) for a given path.

We then developed an iterative, pruning-based scheme (Algorithm 1) that uses this interference index to compute the (primary) paths for the flows. The algorithm has two steps: *(i)* generate a set of candidate paths for all flows and *(ii)* assign the paths to flows by iteratively discarding paths with higher interference (based on interference index). This algorithm is executed offline and has a polynomial complexity of $O(|\widehat{\rho}|(\mathcal{V}+\mathcal{E}))$, where $|\widehat{\rho}|$ is the number of candidate paths and $\mathcal{V}$ and $\mathcal{E}$ are as defined before.

D. Backup Path for Critical Flows

RealFlow provides resiliency by computing backup paths for critical flows; accounting for failed primary links by reserving lowest priority queues in switches along backup paths.

Example 1. Consider the topology in Fig. 3 and a critical flow F_1 with path $P_1 = \{\pi_1, \pi_2, \pi_4, \pi_7, \pi_9, \pi_{10}\}$. A backup flow can be provisioned to account for the failure of link (π_7, π_9), by defining an additional flow $F_1^{'}$ from the switch π_7 to the destination π_{10}. The backup flow has the lowest priority L and uses the last queue on switches $\{\pi_7, \pi_8, \pi_9, \pi_{10}\}$.

Being pushed down to the lowest priority queue on failure, the critical flows will now experience hitherto unseen interference from other, potentially lower priority flows. This could result in *priority inversions* and we need to account for this extra interference during the computation of backup paths. Note that being on the lowest priory queue[4], backup flows will not interfere with other primary path flows.

Given the primary paths for all flows, and K edge failures to consider, our backup path calculation algorithm first updates the network topology $N'(\mathcal{V}, \mathcal{E}')$ where $\mathcal{E}' \subset \mathcal{E}$ is the set of edges that have not failed. Note that our approach is *not* picking specific links for failure *apriori*[5]. Rather, the following procedure is repeated for all critical flows that need resiliency and considers all potential link failures on their primary paths. For each critical flow F_i and failed link considered, an additional, independent flow F_i' is defined with lowest priority *i.e.,* $pri_i' = L$ (*i.e.,* routed through the lowest-priority queue) with its source set to the switch preceding the failed link and its destination set be the same as of flow F_i.

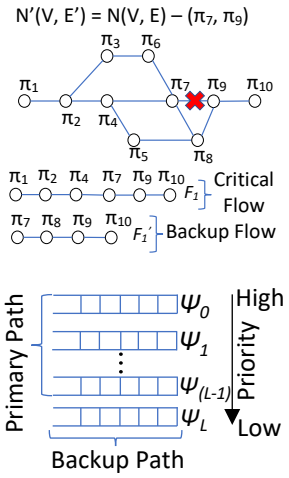


Fig. 3. **Illustration of Backup Path scheme**. Top L queues reserved for primary path. Lowest priority queue is for backup path.

Let us consider $\mathcal{P}_i = \{\pi_1, \pi_2, \cdots, \pi_{j-1}, \pi_j, \pi_{j+1} \cdots, \pi_q\}$ as the q-length primary path of flow F_i obtained by primary path selection mechanism (§III) (where $\pi_1 = src_i$ and $\pi_q = dst_i$) and the failed link $(\pi_j, \pi_{j+1}) \in \mathcal{E}$ is in the path. To calculate backup path, we first update the source node of backup path flow F_i' as π_j – since the flow will be rerouted from this switch to the original destination switch π_q (using fast failover mechanism). We then execute the path selection algorithm over the updated topology $N'(\mathcal{V}, \mathcal{E}')$ from this source node (*i.e.,* point of failure). Note that the link (π_j, π_{j+1}) does not belong to the updated set of edges $\mathcal{E}'$. Besides, if the primary path of a critical flow F_i does not traverse the failed link, we do not need to calculate backup paths for F_i – since the primary path is unaffected by the link failure. Let $\mathcal{F}_B = \{F_i'\}$ denote the set of flows which requires backup path. As we described in §III-A, the backup path traffic of each flow $F_i' \in \mathcal{F}_B$ will experience interference from: *(a)* same priority backup flow traffic (*e.g.,* packets from $\forall F_j' \in \mathcal{F}_B$ routed through same switch) and *(b)* from all other flows (*e.g.,* $\forall F_j \in \mathcal{F} \setminus \{F_i\}$) from their primary path traffic. We repeat the above steps for remaining $K-1$ link failures.

E. Open-Sourced Implementation[6]

1. *Path Layout Algorithm* is implemented as a custom SDN (*northbound*) application. It leverages the global state available

[4]We could reserve more than one lower queue for backup paths at the cost of having fewer remaining queues for primary paths for critical flows.

[5]Although that can certainly be done if the system designers expect a certain link to be more prone to failures.

[6]https://github.com/synercys/RealFlow

in the controller to implement Algorithm 1 for finding flow paths that meet the delay and bandwidth requirements.

2. *Flow Rule Installation*: we push flow rules to individual switches using standard Open vSwitch frameworks and tools, *viz.,* `ovs-ofctl` and `ovs-vsctl` [26]. These flow rules used fast failover groups and strict priority queue scheduling.

IV. Evaluation

We evaluate our framework in three stages:

1. *Simulation-based Performance Evaluation and Comparison with State of the Art* [§IV-A]: We carried out extensive, simulation-based experiments to measure the performance of our scheme for different network topologies as well as varying flow parameters, QoS requirements and random link failures. We also compare it with prior work [10], [12].

2. *Demonstration of Algorithm Behavior and Link Resiliency using COTS Hardware* [§IV-B]: We used a COTS hardware setup (Table III) to demonstrate that our algorithms: *(a)* provide end-to-end timing guarantees, prioritization and isolation for RT flows, *(b)* can multiplex flows and *(c)* provide resiliency to link failures while still guaranteeing end-to-end timing for RT flows on the failed links. We used a mixture of periodic RT and background traffic – to simulate various realistic application scenarios and also to stress-test our mechanisms.

3. *Avionics Case-study [13]* [§IV-C]: We select the flow parameters such as period, packet size, *etc.* similar to those proposed in avionics standards and used in prior research [13], [27].

TABLE III
Switch and Host Configurations

Configuration	Details
Switch Model	Pica-8 P-3297 [19]
Switch OS	PicOS v2.8
Switch Software	Open vSwitch v2.3.0 [26]
OpenFlow	1.5
Host Model	Raspberry Pi
Host OS	Linux Kernel v4.14
Switch-Switch B/W	1 Gbps
Host-Switch B/W	95 Mbps

End-to-end Delay Measurement: While our *algorithms* do not require time synchronization, we used precision time protocol (PTP) between source and destination hosts [28] to *measure* end-to-end delays as the hosts do not share a common clock. We timestamp the packet in the kernel at the source host before sending it and record the time at the destination host after receiving it and capture the wireshark traces. End-to-end delay = (receive timestamp) − (send timestamp) + PTP Synchronization Error.

Deadlines: We calculated the mean, standard deviation and 99.99^{th} percentile for the distribution of empirically observed end-to-end delays on a network with only RT flows. Deadline for each flow is set to $1.1x$ the 99.99^{th} percentile value.

Per-packet Processing Delays: We measured the per-packet processing delay within the Pica8 switch from ingress to egress by subtracting the difference between end-to-end times for a linear network with K switches and $(K-1)$ switches. We varied K from 2 to 4, as we had access to 4 hardware switches. These values are used in §IV-A.

A. Simulation-based Performance Evaluation

We analyze the performance of the algorithms for path layout, multiplexing and backup path calculations, by varying different parameters – essentially, a broad design-space exploration. For this purpose, we simulated a large number of synthetic network topologies and flow sets. We compared the performance of our approach with prior works [10], [12].

1) Simulation Setup: Our parameters were similar to that used in literature [10], [29]. We assumed that the bandwidth of each link $(\pi_{s'}, \pi_s) \in \mathcal{E}$ was 10 Mbps. We considered packet sizes to be $[256, 1024]$ bytes and set the flow periods between $[10, 1000]$ ms. We assumed that each switch contained three queues and there exist three priority-levels, *e.g.,* $L = 3$ and $\mathcal{L} = \{\text{HI}, \text{MED}, \text{LO}\}$. In our experiments we varied the number of flows $|\mathcal{F}| \in [3, 33]$ (*e.g.,* there were $f/3$ flows in each priority-level where $f \in [3, 33]$).

For network-wide evaluation, we generated random graphs with 5 switches (each switch with 2 connected hosts) for each experiment. For each flow F_i, source host src_i and destination dst_i were selected randomly as long as $src_i \neq dst_i$, $(src_i, dst_i) \in \mathcal{V}$. Note that for a fixed packet size and a specific link bandwidth the transmission delay can be considered as constant. We set the upper bound of the transmission delay to: $\frac{1024\times8 \text{ bits}}{10 \text{ Mbps}} = 819.20$ μs (*i.e.,* time to transmit the packet with maximum length). The propagation delay depends on the physical link length and propagation speed in the medium (*e.g.,* the speed varies $.59c$ to $.77c$ where c is speed of light in vacuum). We assume that the length of a link to be no more than $100m$ and the propagation delay is upper bounded by $\frac{100 \text{ m}}{0.66\times3\times10^6} = 505ns$ in fiber-link media [10]. Hence, the transmission and propagation delays $D^{tp}_{s',s}$ for each link $(\pi_{s'}, \pi_s) \in \mathcal{E}$ was set to $819.20 + 0.505 = 819.705$ μs.

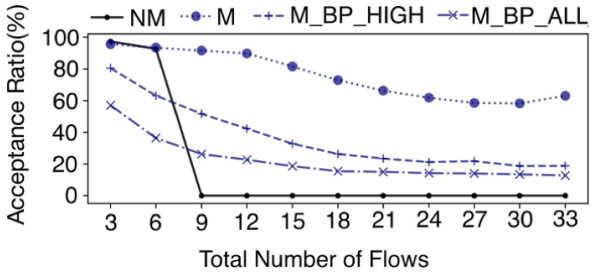


Fig. 4. **Performance comparison of Multiplexing with related work [10] for primary and backup path with varying number of flows**. **NM** is No Multiplexing [10], **M** is Multiplexing. **M_BP_HIGH** is **Realflow** with backup paths for high-priority flows; **M_BP_ALL** is **Realflow** with backup paths for all flows. For each of the experiments we set the deadline $D_{min} = 2400$ μs.

We considered a 'deadline-monotonic' priority assignment scheme [10] where higher priority flows always have more stringent deadlines than lower priority flows (*i.e.,* $\forall F_i, F_j \in \mathcal{F} \wedge pri_i \neq pri_j$, priority of F_i is higher than F_j if $D_i^{max} \leq D_j^{max}$). However, flows belonging to the same priority-level may have different deadlines. To observe the impact of end-to-end deadline constraints in different network topologies, we set the deadline D_i^{max}, $\forall F_i \in \mathcal{F}$ as a function of the topology

[10] as follows: for each randomly generated network topology G_i, we set D_i^{max} to the minimum delay requirement for the highest priority flow as $D_{min} = \beta\delta_i$ and incremented it by $\frac{D_{min}}{10}$ for each of the remaining flows. δ_i denotes the diameter (*i.e.,* 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 we varied D_{min} (β) as an experimental parameter.

2) Simulation Experiments and Results: A given network topology with set of flows is considered *schedulable* if *all* the RT flows in the network can meet the delay and bandwidth requirements[7]. We therefore use *acceptance ratio*, defined as the *number of accepted flow configurations (i.e., configurations that satisfy bandwidth and deadline constraints) over the total number of generated ones*, as a metric to evaluate the schedulability of the flows. For each (end-to-end deadline, number-of-flows) pair, we randomly generate 250 different topologies (using parameters from §IV-A1) and measure the acceptance ratios. As one would expect, stricter deadline requirements limit the schedulability, as does increasing the number of flows. From our simulations, we found that a 5-switch network as per §IV-A1 can admit up to 33 flows as long as $D_{min} \geq 3.2$ ms, as shown in Figure 5.

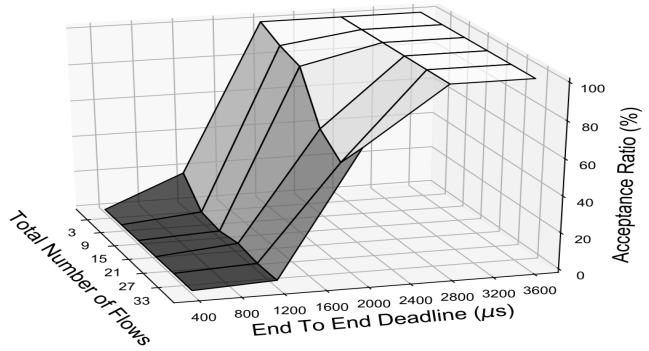
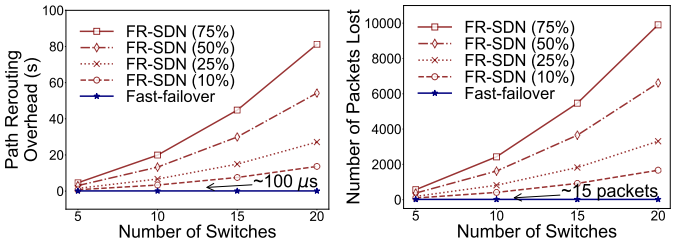


Fig. 5. Acceptance ratio with varying number of flows and end-to-end deadline requirements. Our approach can accommodate up to 33 flows if the deadline $D_{min} \geq 3.2$ ms.

Comparison with Kumar et al. [10]. We next compared (Fig. 4) the performance of our approach with prior work [10] to evaluate the improvement in acceptance ratio due to flow multiplexing. In Kumar *et al.* [10], each flow is assigned an *individual queue* and hence queuing delay is zero. However, their performance is limited by the number of queues supported by the physical switches; they considered 8 queues per port for each of the switches. For a fair comparison, we also limited our experiments to 8 queues per port. The x-axis in Fig. 4 represents total number of flows and y-axis is the acceptance ratio. For the multiplexing scenario we consider the following cases: *(a)* primary path only (no tolerance for link failure); *(b)* backup path for only the highest priority flows ($\forall F_i \in \mathcal{F} \mid pri_i = 0$) and *(c)* backup path for all the flows ($\forall F_i \in \mathcal{F}$).

[7]Note: if a flow set is not schedulable for primary paths it will also be unschedulable for backup paths.



(a) Runtime overhead (in Raspberry Pi) of both for varying number of switches (b) Number of packets lost with varying number of switches

Fig. 6. **Comparison of the runtime performance with the state-of-the-art backup path mechanism**: The percentage (for FR-SDN) denotes the fraction of flows affected by link failure. The results are averaged over 50 trials;

(a) When we only layout primary paths, both Kumar *et al.* [10] and our approach perform similarly when the number of flows is small ($|\mathcal{F}| \leq 8$). For $|\mathcal{F}| > 8$, as expected, our multiplexing scheme significantly outperforms theirs and accepts **63% more flow configurations** as their scheme is limited by the number of queues available in hardware(*i.e.,* up to 8 flows per port).

(b) When provisioning backup paths, the acceptance ratio drops as expected since backup path traffic is rerouted to a (possibly) longer path and, being the lowest priority, it also experiences interference from all other primary path traffic. This causes more interference (due to FIFO queuing of traffic – § III-A2 and § III-D)Thus, there is a *trade-off between acceptance ratio and resiliency*. This reduced acceptance ratio is the cost to improve resiliency. *(c)* Lastly, even when all flows have backup paths, our multiplexing based approach has **18% higher acceptance ratio** since we accept more than 8 flows. Thus, our approach exhibits significantly better network utilization compared with prior work and is able to smoothly trade off between network resilience and utilization.

Comparison with Lee et al. [12]: We also compared our approach with the state-of-the art fault tolerance mechanism (FR-SDN [12]). FR-SDN computes backup paths at runtime by solving an MCP problem [10] at individual switches when link failure occurs. In Fig. 6 we compare the runtime performance of our fast failover scheme with FR-SDN running on Raspberry Pi with a varying number of switches. We set the rate to 1 Mbps for each flow and vary the number of flows that are affected by a link failure as a percentage of total number of flows. As Fig. 6 shows, increasing the size of the network results in increased runtime overheads and packet drop rates for FR-SDN – this is expected as the time complexity of MCP increases with number of flows and the size of the network [10]. In contrast, our scheme (solid, horizontal blue line in Fig. 6) has constant, hardware-based fail-over time that also results in a small, but constant, number of packet drops irrespective of number of flows and network size. Similar results were observed when we change the total number of flows in the system. Note that we recreated the FR-SDN scheme in python[8] using its description [12]. Even if FR-SDN was implemented using a low-level language like C and benefited from the resulting speed-up (Python to C speed-up $\approx$

[8]We requested but were not provided access to FR-SDN implementation.

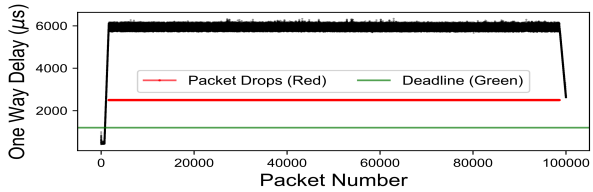


Fig. 7. **End-to-end delays without Algorithm 1**. The end-to-end delay (black line) of a flow that saturates the link capacity exceed the deadline (green line).

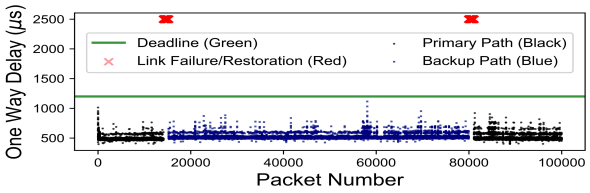


Fig. 8. **End-to-end delays after applying Algorithm 1**. On link failure, the end-to-end delay of the packets are *still* met, barring some that are *in transit* on the failed link (red cross).

10x-20x [30]), fast fail-over still results in significantly lower rerouting delays (100 μs) when compared to FR-SDN (tens of seconds). However, this constant, bounded rerouting time and low packet loss is achieved at the expense of network utilization due to overprovisioning of resources (*e.g.,* the lowest priority queue) that may otherwise be used by other flows. Thus, FR-SDN and RealFlow (pre-computed back-up paths+hardware supported fast failover), represent different design points on opposite ends of the *latency-utilization* trade-off space. Also our approach is COTS compatible while they modify both switches and the OpenFlow protocol.

B. Algorithm Behavior & Link Resiliency on COTS Hardware

We validated our scheme and evaluated its performance with synthetically generated flows with different rates and priorities *running on hardware switches* as listed in Table III. We connected the hosts in a pairwise client-server fashion (*i.e.,* flows are routed from the client host to the server host through our SDN network). The topology is illustrated in Fig. 9 that shows 4 switches: π_1, π_2, π_3 and π_4, with 11 hosts connected to switch π_3 and another 11 hosts connected to switch π_4. The switch to switch links are 1 Gbps Ethernet links. However, the host to switch links are limited by the rate of the hosts, *i.e.,* 95 Mbps — the observed limit for our model of Raspberry Pi. In addition, we have pervasive background traffic in the system, through the path $\pi_1 \rightarrow \pi_2 \rightarrow \pi_3 \rightarrow \pi_4$, designed to saturate the network and is thus generated at 1 Gbps. This is done to stress-test our approach. Note that the background traffic is at a priority lower than the lowest RT flow.

With these flow rates, up to 10 flows can share the shortest path between source and destination hosts. Additional (*i.e.,* 11^{th}) flows will saturate the link capacity (Fig. 7; X-axis: packet number; Y-axis: end-to-end delay for each packet), and would lead to missed deadlines and may result in many packet drops when added to the same path as the other 10.

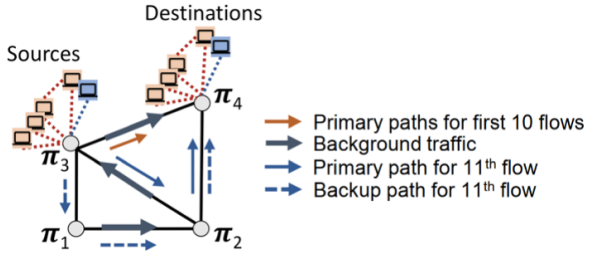


Fig. 9. **Hardware topology demonstrating the Path Selection Algorithm**.

TABLE IV
RT CHARACTERISTICS OF FLOWS USED IN OUR AVIONICS CASE STUDY

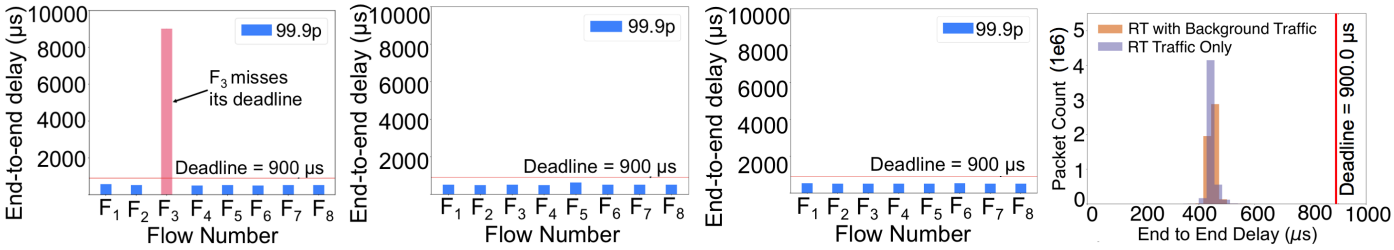
Flows	Size	Period	Deadline	Nature
Flight Control Flows (MEDIUM)				
(F_1–F_4) (*e.g.,* Fly-by-wire control system, Autopilot Flight Director, Flight Data Recorder)	1024 Byte	1 ms	900μs	Hard RT
Cockpit Control Flows (MEDIUM)				
(F_5,F_6) (*e.g.,*. Flight Warning, Flight Management systems)	1024 Byte	1 ms	900μs	Hard RT
Engine Control Flows (HIGH)				
(F_7,F_8) (*e.g.,* Power Systems, FADEC# Systems)	1024 Byte	1 ms	900μs	Hard RT
Cabin Flows (LOW)				
5 Flows (*e.g.,* Cabin Temperature Control, Lighting, Multimedia)	64 Byte	1μs	×	Best Effort

#Full Authority Digital Engine Control

Algorithm 1 routes flows along shortest paths (as long as end-to-end deadlines are met and link capacity is available). When computing paths for a new flow – one that would either saturate link capacity or result in its own deadline to be missed – the algorithm will pick a path that is no longer the shortest one based on the interference calculations from §III-A3. In Fig 9, the path for the 11^{th} flow will be: $\pi_3 \rightarrow \pi_2 \rightarrow \pi_4$, represented by thin blue solid arrows. Furthermore, we calculate backup paths for this flow and find an alternate route that will satisfy its end-to-end deadlines ($\pi_3 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \pi_4$, dashed arrows) in case the link $\pi_3 \rightarrow \pi_2$ on its primary path fails.

Fig. 8 shows the results due to our path assignment (axes similar to Fig. 7). When the packets for the 11th flow follow its assigned primary path, they all meet their deadline (the black dots indicating primary path). To demonstrate the effectiveness of our backup path calculation mechanisms, we *induced a link failure on the $\pi_3 \rightarrow \pi_2$ link*. As the Fig. 8 shows, even after the failure event (the first set of red crosses), packets still meet their end-to-end deadlines (the blue dots indicating backup path) as the fast failover mechanism routes the packets along the pre-computed backup path ($\pi_3 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \pi_4$) *without any assistance from any external entity, say, the controller*. This transition happens in a short, bounded time-frame limited only by the switch hardware. When the original link is restored the packets are rerouted along their original path.

We found that 15 packets were dropped during link failure (at RT flow rates of 1 Mbps) as the fast failover mechanism is not instantaneous. It takes a small, finite, amount of time in COTS SDN switches — as low as 100μs for modern switches operating at line speeds [32]. The "path restoration time" [12] for our system is *bounded* by this hardware feature and is



(a) Naive method (b) Algorithm 1 (c) Algorithm 1 + Link Failure (d) Histogram for F_8

Fig. 10. 99.9^{th} **percentile end-to-end delays for RT flows across 100,000 packets** when: *(a)* using the shortest-path and RT traffic share a common link, *(b)* using paths allocated by Algorithm 1, *(c)* a single link fails between π_1 and π_3 – affected F_3 & F_7 are rerouted through a backup path via switch π_2, *(d)* Histogram for flow F_8 with Kolmogorov—Smirnov (K-S) [31] statistic value of 0.135.

independent of the number of flows, switches or network topology. In contrast, existing approaches [12] compute backup paths at runtime, causing variable delays (§ IV-A2).

C. Case Study: Avionics Network

To further demonstrate the effectiveness of our algorithm, we constructed a 4-switch, 8-flow topology, as shown in Fig. 1 where the specifications are drawn from a case study of an Avionics network [13], summarized in Table IV. All RT flows have a period of 1000 μsecs and packet size of 1024 bytes. We also introduced spurious background traffic between two switches: π_3 and π_4 (at a lower priority than *all* the RT flows). As the number of flows (8) $>$ number of queues (2), multiple flows are multiplexed onto the same priority level. We performed three experiments and our observations (of 99.9^{th} $\%^{ile}$ end-to-end delays) are summarized in Fig. 10.

In the first experiment, we demonstrate the problem with using the shortest-hop path for scheduling the RT flows. We saturated the link between switches π_1 and π_3 by introducing a fixed-rate interfering traffic flow at medium priority. Flows F_3 and F_7 that use this link as their shortest-path.Always using the shortest path can cause flows (*e.g.,* medium priority flow, F_3; Figure 10a) to miss deadlines. The high priority flow F_7 still meets its deadline due to priority-based scheduling.

Next, we lay out flow paths using Algorithm 1 – for the same system. Flow F_3 now takes a 2-hop path: $\pi_1 \rightarrow \pi_2 \rightarrow \pi_3$ instead of the shortest path $\pi_1 \rightarrow \pi_3$. Note that the interference traffic between switches π_1 and π_3 is irrelevant since F_3 no longer uses that link. Figure 10b shows that *all flows (including F_3) meet their deadlines*. Finally, we set up backup paths using the fast failover mechanism. We then fail link π_1 and π_3 that causes F_3 and F_7 to be re-routed through their pre-computed backup paths. Figure 10c shows that *all flows (including F_3 and F_7) meet their deadlines despite the link failure.*

Figure 10d compares the end-to-end times for F_8 with (orange) and without (blue) background traffic on a shared link. As shown by the histogram, the two distributions are very similar. This is also highlighted by the Kolmogorov—Smirnov (K-S) statistic [31] value of 0.135 – a statistical test that provides insights into the similarities between two distributions (lower numbers $\implies$ higher similarity). Importantly, *even in the presence of background traffic saturating the link, packets for F_8 continue to meet their deadlines* ($900\mu s$, the red line).

V. Related Work

We conducted a detailed comparison of RealFlow with state of the art [10], [12] (Section IV-A & Table I). Resilient packet transmission through SDN by reporting link failures to the controller which then generates alternative paths by considering the run-time network status has been proposed [33], [34]. A two-step failure recovery mechanism is also proposed where flow rules are updated by the controller considering QoS requirements [35]. Those approaches, however, may incur long delays [36]. Similarly, some work aims to synthesize flow rules to provide routing resiliency guarantees using fast failover [7], [37], [38] and provisioning alternative paths. However, they have not considered RT constraints on the backup paths.

Recent standardization efforts such as IEEE 802.1Qav and IEEE 802.1Qbv aim to provision RT QoS using Ethernet (TSN approach). However, commodity hardware supporting these standards is not yet widely available. Time-sensitive SDN (TSSDN) [39] considers an SDN architecture but deviates from commodity Ethernet as it assumes a TDMA extension. Meyer *et al.* [40] provided a simulation-based assessment of integration of time-triggered traffic in the Ethernet AVB (audio video bridging). These approaches do not optimize link bandwidth. In contrast, the global view provided by SDNs allows us to optimize path layouts considering both deadline and bandwidth constraints. Liao *et al.* [41] proposed timestamping Link Layer Discovery Protocol (LLDP) packets for link latency monitoring in SDNs. These packets are injected by controllers for global network topology discovery and timestamping them does not require flow table entries to be set up, thus reducing overheads.

Azodolmolky *et al.* [42] and Bemten *et al.* [43] used network calculus based models to estimate delay for packets routed through a switch using extensive benchmarking. However, they neither consider switch scheduling effects (*e.g.,* multiple queues in a port) nor link failures along flow paths. We hope to explore network calculus based approaches in our future work. There also exists work for provisioning end-to-end flows with delay guarantees [44]. but they don't optimize for bandwidth allocation nor considered flow multiplexing. Other work considers QoS constraints for SDN-enabled networks [24] — however RealFlow distinguishes itself by meeting end-to-end timing constraints for flows while improving network utilization and resiliency for such systems.

VI. Conclusion

The use of COTS hardware and software for designing and provisioning RT networks can reduce costs and network sizes – allowing for significant reduction in hardware/software in systems as diverse as automobiles, avionics, power grid substations, industrial control systems and manufacturing plants. The SDN paradigm provides a global view of the network, allows for centralized resource allocation and, with widely available support for COTS switches, presents an interesting opportunity to realize resilient RT networks for critical applications — as shown in this paper.

References

[1] H. Gabler and J. Hinch, "Evaluation of advanced air bag deployment algorithm performance using event data recorders," *Annals of Advances in Automotive Medicine / Annual Scientific Conference*, 2008.

[2] A. E. E. Committee *et al.*, "Aircraft data network part 7, avionics full duplex switched ethernet (afdx) network, arinc specification 664," *Aeronautical Radio*, 2002.

[3] N. Instruments, "Controller Area Network (CAN) Overview." [Online]. Available: http://www.ni.com/white-paper/2732/en/

[4] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," pp. 1–57, March 2016.

[5] 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*, vol. 38, no. 2, pp. 69–74, 2008.

[6] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[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*. IEEE, 2016, pp. 1–6.

[8] R. Bobba, D. R. Borries, R. Hilburn, J. Sanders, M. Hadley, and R. Smith, "Software-defined networking addresses control system requirements," *Government Report.[Online]. Available: https://www. selinc. com/WorkArea/DownloadAsset. aspx*, 2014.

[9] 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*.

[10] 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 *IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[11] T. Qian and F. Mueller, "A failure recovery protocol for software-defined real-time networks," in *in Conference on Compiler, Architecture and Synthesis on Embedded Systems (CASES'18), Oct 2018*, 2018.

[12] K. Lee, M. Kim, H. Kim, H. S. Chwa, J. Lee, and I. Shin, "Fault-Resilient Real-Time Communication Using Software-Defined Networking," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019.

[13] B. Annighoefer, C. Reif, and F. Thieleck, "Network topology optimization for distributed integrated modular avionics," in *Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd*. IEEE, 2014, pp. 4A1–1.

[14] J. M. Nicolas NAVET, Josetxo VILLANUEVA and M. BOYER. (2017) Insights on the performance and configuration of avb and tsn in automotive applications. [Online]. Available: https://bit.ly/2XHMkhr

[15] H. León, C. Montez, O. Valle, and F. Vasques, "Real-time analysis of time-critical messages in iec 61850 electrical substation communication systems," *Energies*, vol. 12, no. 12, p. 2272, 2019.

[16] F. Giroudot and A. Mifdaoui, "Buffer-aware worst-case timing analysis of wormhole nocs using network calculus," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2018*.

[17] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *IEEE Trans. Comput.*, 2001.

[18] S.-N. Yeung and J. Lehoczky, "End-to-end delay analysis for real-time networks," in *Real-Time Systems Symposium*. IEEE, 2001.

[19] "Pica8 P-3297 SDN Switch Datasheet," https://bit.ly/35LJxs5.

[20] S. Altmeyer, S. M. Sundharam, and N. Navet, "The case for fifo real-time scheduling," University of Luxembourg, Tech. Rep., 2016.

[21] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[22] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, no. 1, pp. 95–116, 1984.

[23] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *International Conference on Communications*. IEEE, 1998.

[24] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Communications Surveys & Tutorials*, 2017.

[25] L.-E. Thorelli, "An algorithm for computing all paths in a graph," *BIT Numerical Mathematics*, vol. 6, no. 4, pp. 347–349, 1966.

[26] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch." in *NSDI*, vol. 15, 2015, pp. 117–130.

[27] O. Hotescu, K. Jaffrès-Runser, J.-L. Scharbarg, and C. Fraboul, "Towards quality of service provision with avionics full duplex switching," *Edited by Patrick Meumeu Yomsi*, p. 19, 2017.

[28] K. Correll, N. Barendt, and M. Branicky, "Design considerations for software only implementations of the ieee 1588 precision time protocol," in *Conference on IEEE*, vol. 1588, 2005, pp. 11–15.

[29] T. Qian, F. Mueller, and Y. Xin, "Hybrid edf packet scheduling for real-time distributed systems," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*. IEEE, 2015, pp. 37–46.

[30] "Python 3 versus C gcc fastest programs," https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html, [Online].

[31] W. W. Daniel, "Kolmogorov–smirnov one-sample test," *Applied Nonparametric Statistics*, vol. 2, 1990.

[32] M. Hadley, D. Nicol, and R. Smith, "Software-defined networking redefines performance for ethernet control systems," in *Power and Energy Automation Conference*, 2017.

[33] S. Hegde, S. G. Koolagudi, and S. Bhattacharya, "Path restoration in source routed software defined networks," in *9th IEEE International Conference on Ubiquitous and Future Networks (ICUFN) 2017*.

[34] S. Paris, G. S. Paschos, and J. Leguay, "Dynamic control for failure recovery and flow reconfiguration in SDN," in *International Conference on the Design of Reliable Communication Networks*. IEEE, 2016.

[35] L. Wang, L. Yao, Z. Xu, G. Wu, and M. S. Obaidat, "CFR: A cooperative link failure recovery scheme in software-defined networks," *International Journal of Communication Systems*, vol. 31, no. 10, p. e3560, 2018.

[36] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015, p. 25.

[37] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[38] D. Gyllstrom, N. Braga, and J. Kurose, "Recovery from link failures in a Smart Grid communication network using OpenFlow," in *International Conference on Smart Grid Communications*. IEEE, 2014.

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

[40] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending ieee 802.1 avb with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *2013 IEEE Vehicular Networking Conference*, Dec 2013, pp. 47–54.

[41] L. Liao and V. C. M. Leung, "Lldp based link latency monitoring in software defined networks," in *2016 12th International Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 330–335.

[42] 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*. IEEE, 2013, pp. 1397–1402.

[43] A. Van Bemten, N. Deric, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: Predictable latency in small networks," in *International Conference on Emerging Networking Experiments And Technologies*, 2019.

[44] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 2014.