Fault-Resilient Real-Time Communication Using Software-Defined Networking

Kilho Lee*, Minsu Kim*, Hayeon Kim*, Hoon Sung Chwa[†§], Jinkyu Lee[‡], Insik Shin*
School of Computing, KAIST, Republic of Korea*
Information and Communication Engineering, DGIST, Republic of Korea[†]
Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea[‡]
Email: chwahs@dgist.ac.kr

***Abstract*—The development of complex cyber-physical systems necessitates real-time networking with timing guarantees even in the presence of a link fault. Targeting firm real-time flows with the maximum allowable number of continuous deadline misses, this paper introduces FR-SDN, a fault-resilient SDN (Software-Defined Networking) framework that satisfies the timing requirements of firm real-time flows. To this end, we first investigate individual steps for path restoration: fault recognition, path recalculation, and path reassignment. We then design novel system architecture that reduces the delay of the fault recognition and path reassignment steps to potentially assign more time budget to the path recalculation step. Based on the calculation of tight upper-bounds on the delays in individual steps under the proposed system design, we derive a necessary feasibility condition that guarantees the timing requirements of firm real-time flows, and we calculate a time budget for the path recalculation step. Finally, we develop a multi-constrained path finding algorithm that can dynamically adjust the scope of flows to reroute according to the time budget. To the best of our knowledge, FR-SDN is the first study on adaptive path restoration for real-time flows, taking into account path restoration delay and fault tolerance constraints in case of link fault. We have implemented and evaluated FR-SDN on top of Open vSwitch to demonstrate its effectiveness, achieving an order of magnitude reduction in path restoration delay. In addition, we have deployed FR-SDN into a 1/10 scale autonomous vehicle and have shown, via an in-depth case study of adaptive cruise control, that FR-SDN is able to meet all fault tolerance requirements so that it can behave similarly as if there were no link failure.**

I. Introduction

Recent advances in embedded processors and software technologies have fostered the development of complex cyber-physical systems (CPS) such as autonomous driving vehicles. CPS typically integrate sensors, actuators, and computing units to continuously interact with the physical world in real-time. Thus, it is important to develop network techniques that meet CPS requirements. In this paper, we focus on two key requirements on CPS networking, which are deterministic and fault-tolerant communications.

Automotive and avionics industries seek to develop Ethernet-based networking standards [1]–[3] in order to meet higher bandwidth and lower latency requirements of complex CPS applications such as vision/LIDAR-based SLAM (Simultaneous Localization And Mapping). For deterministic communication, the IEEE TSN (Time-Sensitive Networking) working group is developing Ethernet-based new standards, such as time synchronization [4], bandwidth reservation [5], and traffic scheduling [6]. For fault tolerance, one approach is to support *fault-free* communication such that it continues to deliver messages without any loss even in the presence of network failure. To this end, the IEEE TSN [7] and AFDX [1] standards employ redundant packet transmission. This approach suits the strong requirements of hard real-time flows that allow no single packet loss, yet it incurs low network utilization ($\leq 50\%$) due to duplicate packet transmissions on multiple paths in parallel. Another approach is *fault-resilient* communication where some flows can experience packet loss or delay in case of network failures but no more packet loss or delay after recovering from the failures. Such fault-resilient communication is suitable for a class of real-time flows with so-called *firm* deadlines [8]–[13]. Firm real-time flows can tolerate the occasional loss of some deadlines. As examples, some advanced control systems are robust enough to react properly without serious consequences while not receiving input data, and users may not notice the effect of audio packet loss in video conferencing systems. It is then important to recover from failures quickly and in a deterministic way.

In the literature, several studies [14]–[16] propose to leverage SDN (Software-Defined Networking) to perform path restoration for fault-resilient communication on switched Ethernet. Though many SDN-based path restoration techniques enable to recover from link failure by finding alternative routes, they are not appropriate for firm real-time flows that have loss requirements such as at most a couple of consecutive deadline misses. This is because path restoration inherently comprises three steps of S1) *Fault Recognition*, S2) *Path Recalculation*, and S3) *Path Reassignment* on SDN networks, and each step incurs a long delay in an unpredictable manner. Our motivational benchmark shows that such delays can be hundreds of milliseconds. These long delays may have serious consequences for system safety. For example, a camera sensor operating at 60 Hz may experience continuous loss of messages dozens of times.

In this paper, we aim to develop a fault-resilient real-time networking system using SDN, called FR-SDN, which supports fast and predictable path restoration to meet the *loss* (or fault tolerance) constraints of firm real-time flows (See Figure 1). Assuming each firm real-time flow comes with its own loss parameter that indicates the maximum allowable number of continuous deadline misses upon a single link failure, this paper seeks to answer the following questions:

- Q1. What is a feasibility condition that enables FR-SDN to meet the loss requirements of firm real-time flows?
- Q2. What is a good design to perform path recalculation on SDN, aiming at maximizing the possibility of fault recovery without violating any loss requirement.

For Q1, we present feasibility conditions focusing on the temporal aspect of path restoration, which implies the important design directions of FR-SDN towards deterministic and efficient path restoration. For Q2, FR-SDN first computes the delay budget (deadline) allowed for path restoration. In order

[§]A corresponding author.

to maximize the possibility of performing path restoration successfully within a given delay budget, it is important to reduce the total time spent in the fault recognition (S1) and the path reassignment (S3) steps. This is because the performance of the path recalculation (S2) step directly depends on how much time can be spent in the S2 step. To this end, FR-SDN completely changes the way of performing path restoration, from SDN controller-driven to switch-driven. This changes not only significantly reduces the communication delays of S1 and S3 steps, but also enables to derive tight upper-bounds on the delays. Though we increased the time-budget of S2 via switch-driven path restoration, its time-budget may be still too short to carry out an exhaustive search for schedulable alternative paths. Thus, we employ an adaptive *Multi-Constrained Path* (MCP) technique that selectively reroutes only some flows in the network so as to perform path recalculation effectively within a limited time-budget. Furthermore, we propose heuristics that select flows for performance improvement.

To evaluate the performance of FR-SDN, we conducted extensive simulations with randomly generated real-time flows. The simulation shows our adaptive MCP technique outperforms static MCP techniques. In addition, we implemented FR-SDN on top of Open vSwitch [17] and undertook various experiments on an actual network testbed which consists of 21 single-board computers. Our experimental results show that FR-SDN significantly reduces the path restoration delay by an order of magnitude and improves the reliability of real-time flows, compared to the standard SDN system. Beyond the extensive simulations and experiments, we deployed FR-SDN into a 1/10 scale autonomous vehicle and conducted a case study (i.e., Adaptive Cruise Control). The case study shows that FR-SDN results in very reliable control performance despite a link fault; it can follow the preceding vehicle while keeping a reference distance. In contrast, the standard SDN cannot keep the reference distance due to the data losses caused by the long delay of path restoration.

The contributions of this paper are summarized as follows:

- We introduce FR-SDN, a fault-resilient SDN framework, which supports firm real-time flows with their own fault tolerance constraints. To the best of our knowledge, this paper presents the first study on adaptive path restoration for real-time flows, taking into account path restoration delay and fault tolerance constraints in case of link fault.
- We design novel system architecture for SDN/OpenFlow to offload the path restoration into switches (Section V).
- We derive a feasibility test to guarantee satisfaction of loss requirements of firm real-time flows, based on the upper-bounds of individual delay factors in the three path restoration steps (Section VI).
- We develop a multi-constrained path finding algorithm that can dynamically adjust the scope of the path finding according to the path recalculation budget (Section VII).
- We perform extensive simulations for path finding, implement FR-SDN on top of Open vSwitch, and experiment FR-SDN on an actual network testbed (Section VIII).
- We deploy FR-SDN into a 1/10 scale autonomous car[1], and perform a case study (Section IX).

[1]See http://cps.kaist.ac.kr/frsdn for the demo video illustrating how well FR-SDN preserves the autonomous driving performance against a link fault.

II. System Model and Background

Flow model. We consider a SDN-enabled network modeled as a graph, $N = (V, E)$, where V is the set of switches and E is the set of links that are subject to failure. A link between two switches u and v is represented as (u, v). We also consider a flow set F composed of periodic real-time *flows*. Each flow $f_i \in F$ is characterized by (s_i, t_i, T_i, A_i, D_i, B_i, K_i), where s_i and $t_i \in V$ are source and destination switches respectively, T_i is a period, A_i is a message size, D_i is an end-to-end transmission deadline, B_i is the maximum required bandwidth, and K_i is the maximum tolerable consecutive message losses. We follow the delay/bandwidth model (i.e., required bandwidth, B_i, and required propagation delay, $P_i = D_i - A_i/B_i$) from [18] and the firm-deadline model (K_i) from many existing studies, e.g., [11], [13]. Each f_i is assumed to generate a potentially unbounded series of *messages* every T_i time-units with each message needing to arrive at t_i from s_i within its end-to-end deadline of D_i time-units. Each message can be divided into multiple packets, depending on the maximum transmission unit (MTU) of the link (e.g., 1500 bytes on Ethernet).

Fluid-based flow scheduling. We consider fluid-based flow scheduling. Each switch $v \in V$ has a set of queues, and each queue is assigned a fraction of bandwidth. Any feasible path assignment guarantees each flow uses the queues on its path according to its maximum required bandwidth [18]. Every message of each flow f_i can thus be scheduled constantly at a rate equal to B_i with no queueing delay.

Path finding algorithm. For a given network and flow set, we consider the Multi-Constrained Path (MCP) problem that finds a path Φ_i for each flow f_i subject to the bandwidth and delay constraints of all flows. Since the MCP problem is known to be NP-complete [19], we take a polynomial-time heuristic algorithm from [18] that is an extension of [20]. It solves the relaxed problem, transforming one of the delay and bandwidth constraints into integers, and proves the correctness of the solution [18].

Fault detection. We consider fault recovery from any *single* link failure.[2] Link failure detection is possible by using the BFD (Bidirectional Forwarding Detection) protocol [21], which is supported by most hardware and software switches [22]–[24]. We assume that the time to detect a link failure after its occurrence can be bounded via the BFD protocol.

SDN (Software-Defined Networking). Software-Defined Networking is an emerging network architecture, which decouples the control plane from the data plane to enable flexible network control with the global view of the network. The decoupled control plane becomes a software running on the centralized controller, and controls the data plane through the standard control interface. OpenFlow [25] is the de facto control protocol between the control and data planes; it enables to dynamically manage the data plane including packet forwarding rule management, and also enables to maintain the global network information by collecting the network status from the data plane. Beyond the limitations of Ethernet such as static rule management and a lack of network information, SDN

[2]When multiple links fail simultaneously, we consider this case to be a sequential occurrence of a single link failure.

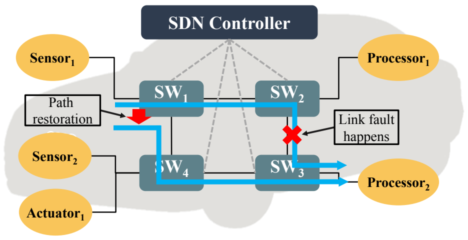


Fig. 1: Path restoration on SDN-based CPS networking system

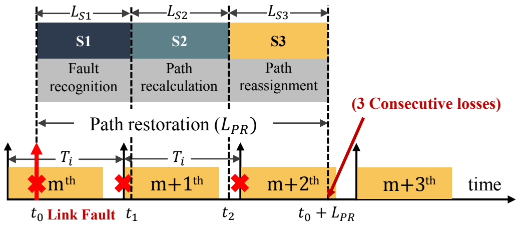


Fig. 2: An example of the path restoration

provides a great opportunity to perform the path restoration based on its flexible network control capability and the global view of the network; it enables to calculate alternative paths on-demand according to the global network view and to update forwarding rules at runtime.

III. FR-SDN: Overview

In this paper, we aim to develop a fault-resilient real-time Ethernet framework that achieves the following goals for every flow $f_i \in F$.

G1. When there is no link failure, all messages of f_i should satisfy their end-to-end delay requirement D_i; and
G2. When a link fault occurs, up to K_i consecutive messages of f_i can violate their end-to-end delay requirement D_i.

To achieve G1, FR-SDN uses fluid-based flow scheduling for each individual link, which guarantees the delivery of each message without violating its end-to-end deadline as long as its bandwidth and propagation delay constraints are met [18]. We use the MCP algorithm for finding the path of each individual flow f_i subject to satisfying its bandwidth B_i and propagation delay P_i constraints, where $P_i = D_i - A_i/B_i$. Then, any solution that the MCP algorithm finds will satisfy the end-to-end deadline of each message.

To achieve G2, FR-SDN aims to support a fault-resilient communication on SDN-enabled switched Ethernet through on-demand path restoration (see Figure 1). The path restoration process typically involves the following three steps on standard SDN environments.

S1. *Fault recognition*: as soon as a SDN switch detects a link fault, it sends a link fault notification to the SDN controller.
S2. *Path recalculation*: the controller then finds alternative paths for flows so as to bypass the faulty link.
S3. *Path reassignment*: the controller deploys new forwarding rules for the alternative paths to all switches.

Then, the whole path restoration delay is the sum of delays incurred in each step.

Figure 2 illustrates a situation where a link fault occurs in the middle of transmitting the m^{th} message of flow f_i. In the figure, the path restoration process begins immediately when the link fault is detected, and it finishes when f_i still suffers from the link failure in transmitting the $(m+2)^{th}$ message. That is, the path restoration process overlaps three periods of f_i, causing f_i to experience the loss of three consecutive messages with the link fault. Here, if flow f_i can tolerate the loss of at most 2 consecutive messages (i.e., $K_i = 2$), it violates the loss requirement of f_i.

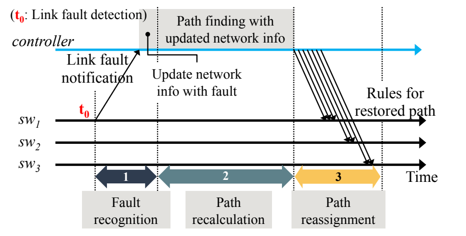
This motivating example reveals interesting design issues to consider. First, the time to recover from failure is important. In other words, path restoration has a delay budget (or a deadline) for completion in order to meet the loss requirements of real-time flows. Second, it is important to reduce the path restoration delay as much as possible in order to maximize the possibility of satisfying the loss requirements. However, we cannot simply reduce the computation time of path recalculation in S2, since its performance is directly affected by its computation time (or its search space). Thus, FR-SDN takes the following strategy: it aims to substantially reduce the time taken in the S1 and S3 steps (see Section V) and enables to find schedulable alternative paths adaptively within a limited time budget in the S2 step (see Section VII). Third, one may wonder the following question: would it be feasible to predict in advance whether each flow f_i can still meet its own loss requirement in the case of link failure? To answer this question, it is important to derive tight upper-bounds on the delays in each step and feasibility analysis for firm real-time flows (see Section VI).

IV. Path Restoration Delays on SDN: Experimental Results and Analysis

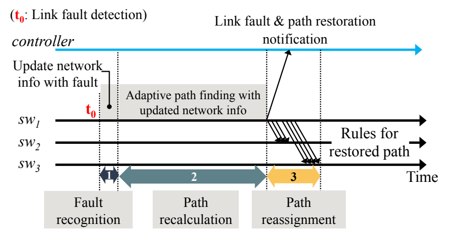
SDN makes it possible to support fault recovery through on-demand path restoration on switched Ethernet. However, the centralized network control architecture of SDN introduces very long and unpredictable delays in the path restoration process. Such delays make it difficult to minimize packet loss and satisfy the loss requirements of real-time flows in a predictable manner. This section presents our motivating experimental results in order to see how long such delays can be and examines major delay factors, which is a key to design FR-SDN in Section V.

A. Path Restoration Delays

In order to estimate the delay of path restoration, we undertook a motivating experiment on a real-world network testbed that consists of tens of single-board computers (refer to Section VIII-B for more details). For the experiment, we used a path restoration approach that follows the principle of the standard request-response SDN protocol, as shown in Figure 3(a), which is commonly deployed by most SDN-based path restoration methods [14]–[16]. This standard approach works as follows: i) a switch sends a link fault notification to the SDN controller upon detecting a link fault; ii) the controller then finds alternative paths for flows suffering from the fault; and iii) the controller deploys new forwarding rules for the alternative (restored) paths to all switches. Figure 4 shows the path restoration delay with varying the number of flows from 20 to 100. In the experiment, we emulated a link fault by taking the interface down (i.e., `ifdown` command), and we measured *fault restoration delay* as an elapsed time between



(a) Std-SDN: a controller-driven path restoration process which follows the standard request-response SDN control protocol


(t0: Link fault detection)
Link fault & path restoration notification
controller
Update network info with fault
Adaptive path finding with updated network info
t0
sw1
sw2
sw3
Rules for restored path
Time
1
2
3
Fault recognition
Path recalculation
Path reassignment

(b) FR-SDN: a switch-driven path restoration process

Fig. 3: Path restoration process and its delay analysis

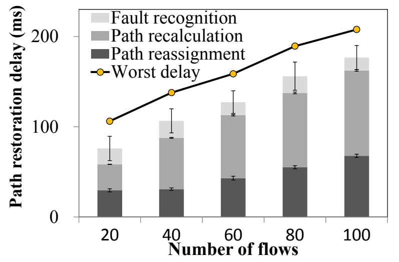


Fig. 4: Path restoration delay breakdown of the motivational experiment

the fault detection time and the time at which all new rules are deployed in every switch. The figure shows that the path restoration delay can be hundreds of milliseconds (e.g., up to $211ms$ with 100 flows) with high variances. Such a long and fluctuating delay may damage the timing guarantee of real-time flows; for instance, it may result in consecutive message losses of a camera sensor running at 60 Hz, which can bring serious consequences for system safety.

B. Major Delay Factors

To investigate the cause of the path restoration delay, we decompose the path restoration delay into three steps, S1, S2, and S3, as shown in Figure 3(a). We then analyze each step in terms of delay duration and the main causes of the delay.

S1: Fault Recognition. In this step, upon detecting a link fault, a switch sends a fault notification to the SDN controller through OpenFlow channel. Upon receiving the notification, the controller updates its network information with the fault status. Note that the computation cost for updating the network information on the controller is negligible (i.e., few microseconds), since it only requires simple memory update operations. Then, the major delay factor of this step is the OpenFlow communication between the switch and the controller.

Figure 4 shows the time taken to handle OpenFlow packets is long and fluctuated (up to 49 ms) even though the notification message is very small (i.e., 74 bytes including headers).[3] This is because the SDN controller is typically equipped with complicated software stack, including an OS network stack, a SDN controller framework, and a SDN controller application, and an OpenFlow message needs to pass through those layers on the controller. Furthermore, the OpenFlow message can be delayed by a scheduling policy or an optimization technique (e.g., batching) in each layer on the controller. Thus, it is not trivial to reduce and bound the delay on the controller.

S2: Path Recalculation. This step performs a heavy path finding computation, which incurs another long delay. In this step, we used the MCP solver proposed in [18] to find alternative paths of all flows against the link fault. The MCP solver calculates each flow's path in a sequential manner, by considering the bandwidth and end-to-end latency constraints of each flow subject to the path assignment of previous flows. Thus, its computation complexity is a linear function of the number of flows. In our experiment, the path calculation delay takes up to 94 ms for 100 flows (See Figure 4). Since this delay is inevitable for finding new paths, it is impossible to finish the path calculation step without compromising each flow's requirements. For example, the experiment with 100 flows, a flow experiencing a link fault cannot complete its path restoration within its timing requirement of 100 ms.

[3]A measurement study [26] has also reported that OpenFlow communication throughput and latency widely fluctuate depending on the controller's setup and load; for instance, the latency varies from 100 μs to 1268 ms.

S3: Path Reassignment. This step also involves OpenFlow communication with numerous OpenFlow messages to update the forwarding rules of each switch. It imposes another long and unpredictable delay, which takes a significant portion of the entire path restoration delay. Figure 4 shows that this delay takes up to 72 ms to reassign forwarding rules for the 100 flow case; note that the case changes the paths of 20 out of 100 flows. Since this delay is caused by the controller and switch software stacks for OpenFlow communication, it is not easy to reduce and bound this delay due to the same reason of the fault recognition delay.

V. FR-SDN: System Design

We propose FR-SDN, a novel SDN-based fault-resilient networking system, which provides a path restoration functionality with reduced and bounded delays. Note that providing a bounded path restoration delay is essential to satisfy the loss requirements of firm real-time flows. As shown in Figure 3(b), FR-SDN totally changes the way of performing path restoration from controller-driven to switch-driven, which is a key principle to eliminate the major delay factors in the fault recognition (S1) and path reassignment (S3) steps. Furthermore, FR-SDN employs an adaptive path finding method that selectively reroutes only some flows in order to complete the path recalculation (S2) step within a limited time budget.

FR-SDN redesigns the three steps of path restoration as follows. In S1, when a switch detects a link fault, it reflects the fault into its own global network information without notifying it to the SDN controller, while each switch holds its own copy of the global network information in synchronization with the controller. In S2, the switch then immediately runs an adaptive path finding method to discover alternative paths that bypass the faulty link. In S3, upon completing the path recalculation step, the switch broadcasts the alternative paths to all other switches in the network, and the other switches received the path information then update their forwarding rules accordingly.

FR-SDN extends the data plane in order to add components for *path recalculation manager* and *path reassignment manager* as shown in Figure 5 (colored as dark navy). This section will explain how each new component in the data plane effectively reduces and upper-bounds delays incurred in each path restoration step.

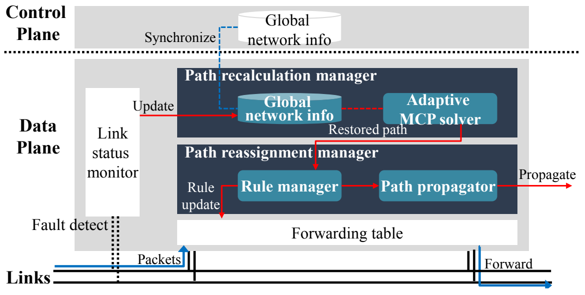


Fig. 5: FR-SDN system architecture

Switch-driven fault recognition and path recalculation. Each switch needs to know the global network information, such as network topology and bandwidth allocation, in order to perform path recalculation upon recognizing a link fault. However, the standard SDN allows only the controller to maintain such network information alone; and each switch maintains forwarding rules only for packet handling.

To overcome such limitation, FR-SDN adds *Path Recalculation Manager* that enables each switch to gather network information necessary for finding alternative paths, including network topology, link information (capacity and delay on each link), flow information, and bandwidth allocation. Whenever there is a change to the network status (e.g., addition of a new flow), the controller updates the copy of global network information on each switch, enabling the switch to respond correctly in case of a link fault. Since such a switch-driven approach eliminates OpenFlow communication, it is possible to reduce and upper-bound the fault recognition delay; it allows to replace the longer delays incurred by complex OpenFlow communication stacks on the controller and the switch with a much shorter one for simply memory instructions on the switch.

Moreover, the path recalculation manager employs an adaptive MCP (Multi-Constrained Path finding) solver, which enables each switch to start the path recalculation step immediately based on its own global network information. One of the key features of FR-SDN is that it can dynamically adjust the number of flows to reroute so as to complete the path recalculation step within a given limited time budget. This way, FR-SDN is able to provide timing guarantees to firm real-time flows subject to successful path recalculation (we will explain it more in Sections VI and VII).

Switch-driven path reassignment. The controller-driven path reassignment step causes a significant delay for path restoration, because the centralized controller should send multiple OpenFlow messages to every switch. On the other hand, FR-SDN achieves, via switch-driven path reassignment, a significant reduction in the path restoration delay.

In particular, FR-SDN adds *path reassignment manager* that consists of *rule manager* and *path propagator* sub-modules, providing a new inter-switch communication channel for path reassignment. Once the path recalculation step is done, the path propagator sub-module generates a path propagation message that encapsulates information about the new paths, describing each new path as a pair of flow ID and a sequence of switch IDs. It then broadcasts the message to other switches in the network through the bandwidth-reserved dedicated queue of each link. Upon receiving the path propagation message, the rule manager sub-module decapsulates the message, translates path information into forwarding rules, and directly updates them to the forwarding table. This way, it excludes OpenFlow communication and enables to reduce the path reassignment delay effectively.

VI. Feasibility Test for Firm Real-Time Flows

In this section, we seek to answer whether FR-SDN can recover from link failure (by performing path restoration) successfully subject to the loss requirements of any given firm real-time flows. Note that such success depends on two aspects: i) logical correctness in a sense that it finds schedulable alternative paths that satisfy G1 (without compromising their bandwidth and delay requirements) after the path restoration process completes, and ii) temporal correctness in a sense that it finds the schedulable alternative paths early enough to satisfy G2 during the path restoration process. While the former will be addressed in Section VII, this section focuses on the latter.

Different from standard SDN, FR-SDN is capable of reducing and upper-bounding the path restoration delay, and we leverage this capability to derive a feasibility test from the temporal aspect. To this end, we first explain how to calculate and upper-bound the path restoration delay under FR-SDN. We then derive a necessary offline feasibility test for every firm real-time flow, based on a runtime condition for each flow's loss requirement. Finally, we define a notion of time budget derived from the necessary condition to be utilized for adaptive path calculation in Section VII.

A. Calculation and Upper-Bound of Path Restoration Delay

Now, we derive an upper-bound on the path restoration delay of FR-SDN in case of a single link failure. As mentioned in Section III, the path restoration delay is the sum of the delays incurred in each of the three steps, and therefore we can express the worst-case path restoration delay of FR-SDN (denoted by L_{PR}) as follows:

$$L_{PR} = L_{S1} + L_{S2} + L_{S3}, \tag{1}$$

where L_{S1}, L_{S2}, and L_{S3} denote the worst-case running times in the fault recognition, path recalculation, and path reassignment steps, respectively. We now investigate individual delay components.

The fault recognition delay L_{S1} is the time to update the network information, used for the path recalculation, with the fault information. Since FR-SDN introduces the switch-driven fault recognition, this delay is made by a few memory instructions to access the network information in the switch; it can be upper-bounded by the worst-case execution time of the memory instructions.

The path calculation delay L_{S2} can be upper-bounded by the worst-case execution time of our adaptive path calculation algorithm. As detailed in Section VII, our proposed algorithm takes the number of flows to reroute (denoted by N_{route}) as input and requires $O(N_{route})$-time. Then, an upper-bound on L_{S2} can be derived as

$$L_{S2}(N_{route}) = \ell_{S2} \cdot N_{route} + \ell_{c-misc}, \tag{2}$$

where ℓ_{S2} is the maximum time to find a feasible path for one flow in a given network, and ℓ_{c-mise} is an additional overhead regardless of N_{route}, such as an initialization cost of the path finding solver. Note that ℓ_{S2} can be expressed by $O((X + B_{max}) \cdot |V| \cdot |E|)$ [18]. Here, X is a given constant parameter and B_{max} is the maximum bandwidth utilization of a path, which can be inferred by the network. Therefore, we can upper-bound ℓ_{S2} for a given network setup.

The path reassignment delay L_{S3} is the time to install new forwarding rules associated with the alternative paths into all switches. Once a switch that detects a fault finishes the fault recognition and path recalculation steps, it then propagates the alternative paths via an inter-switch communication channel to all other switches; the other switches receiving the paths update their forwarding table. Thus, L_{S3} can be expressed with the function of the number of forwarding rules to update (denoted by N_{rule}) as follows:

$$L_{S3} = L_{deliver} + L_{update}(N_{rule}), \tag{3}$$

where $L_{deliver}$ is the maximum delivery time of a rule update message between switches, and $L_{update}(N_{rule})$ is the maximum update time of new forwarding rules on each switch. $L_{deliver}$ can be derived by multiplying the worst-case delay on each hop by the maximum hop distance between switches (denoted by L_{link}):

$$L_{deliver} = (\ell_{trans} + \ell_{prop} + \ell_{proc} + \ell_{flood}) \cdot N_{link}, \tag{4}$$

where ℓ_{trans}, ℓ_{prop}, ℓ_{proc}, and ℓ_{flood} are the delays of transmission, propagation, processing, and packet-flooding, respectively. Note that ℓ_{trans} and ℓ_{prop} are determined by physical properties of the network system such as link bandwidth, physical link length, and link propagation speed, and ℓ_{proc} and ℓ_{flood} are dependent on switch architecture. Such delay factors can be bounded by constant values according to empirical measurement. Note that there is no queueing delay because we assume that each flow is assigned to an individual queue. The upper-bound of $L_{update}(N_{rule})$ can be calculated by

$$L_{update}(N_{rule}) = \ell_{update} \cdot N_{rule}, \tag{5}$$

where ℓ_{update} is the maximum required time to update a single forwarding rule in the forwarding table.

In summary, for a given link fault occurs at t, we can calculate and upper-bound L_{S1} and L_{S3}. Also, once the number of flows to reroute (N_{route}) is determined (by the adaptive path calculation algorithm in Section VII), we can calculate and upper-bound $L_{S2}(N_{route})$. In the next subsection, we derive an offline necessary feasibility test for firm real-time flows, by utilizing those upper-bounds.

B. Feasibility Conditions

For ease of presentation, let us define some notations. When a fault occurs to a link, let $F_{LF} \subseteq F$ denote a set of flows that pass the faulty link.

We present the following lemma as a necessary and sufficient condition for a flow experiencing a link fault to meet its loss requirement in the temporal aspect.

Lemma 1. *Suppose a link fault occurs at time t_0 (see Figure 2) and FR-SDN completes the path restoration process at $t_0 + L_{PR}(t_0)$ finding schedulable alternative paths. Then, FR-SDN satisfies the K_i loss requirement of flow $f_i \in F_{LF}$ if $L_{PR}(t_0) \leq R_i(t_0) + T_i \cdot (K_i - 1)$, where $R_i(t_0)$ is the time difference between t_0 and the first invocation of f_i after t_0.*

Proof. For given $K_i \geq 1$, we need to finish the overall path restoration so as to successfully reroute the $(m + K_i)^{th}$ message of f_i, where the m^{th} message of f_k is the first message to be lost. Then, $R_i(t_0)$ implies the time between t_0 and the release of the $(m + 1)^{th}$ message, and the time duration between the release of the $(m+1)^{th}$ and $(m+K_i)^{th}$ messages is calculated by $T_i \cdot (K_i - 1)$. Therefore, if the overall path restoration delay is no larger than $R_i(t_0) + T_i \cdot (K_i - 1)$, a new path for f_i is deployed in every switch in the network until the release time of the $(m + K_i)^{th}$ message of f_i, which proves the lemma. □

Lemma 1 implies the lower-bound of K_i for $f_i \in F_{LF}$ that FR-SDN can support in practice is 2. In the worst-case scenario, $R_i(t_0)$ can be a very short duration ϵ, where $0 < \epsilon < L_{PR}$. That is, the path restoration process can overlap two periods of f_i, and in this case, the two messages of f_i can be lost during the path restoration process.

On the other hand, the following lemma presents a corresponding lemma for a flow that does not experience a link fault directly.

Lemma 2. *Suppose a link fault occurs at time t_0 (see Figure 2), and FR-SDN completes the path recalculation (S2) step at t_2 finding schedulable alternative paths. Then, by definition, the path reassignment (S3) step starts at t_2 and finishes at $t_2 + L_{S3}(t_2)$. Then, FR-SDN satisfies the K_j loss requirement of flow $f_j \in F \setminus F_{LF}$ if $L_{S3}(t_2) \leq R_j(t_2) + T_j \cdot (K_j - 1)$, where $R_j(t_2)$ is the time difference between t_2 and the first invocation of f_j after t_2.*

Proof. The proof is similar to Lemma 1. When the path reassignment step overlaps the two periods of f_j, the two messages f_j within the overlapped periods can be lost due to inconsistent forwarding rule updates across switches. □

Building upon the above two lemmas that use online information (i.e., $R_i(t)$), we can derive an offline necessary feasibility condition in the following theorem.

Theorem 1. *Suppose upon a link failure, FR-SDN completes the path restoration process successfully finding schedulable alternative paths for all flows. Then, FR-SDN can satisfy the loss requirements of flows $f_i \in F_{LF}$ and $f_j \in F \setminus F_{LF}$, respectively, if*

$$L_{S1} + L_{S2}(|F_{LF}|) + L_{S3} \leq T_i \cdot (K_i - 1), \tag{6}$$

$$L_{S3} \leq T_j \cdot (K_j - 1). \tag{7}$$

Proof. The LHS (Left-Hand Side) of Eq. (6) implies the minimum path restoration time. As we explain in the previous subsection, while L_{S1} and L_{S3} are static values, L_{S2} is a function of the number of flows to reroute. Then, F_{LF} is the minimum set of flows to reroute; if any flow in F_{LF} is not given a new path, it inevitably loses its following messages. Therefore, $L_{S2}(|F_{LF}|)$ is the minimum time for the path recalculation step. On the other hand, the RHS (Right-Hand Side) of Eq. (7) implies the maximum delay for the overall path restoration obtained offline; this is because, $R_i(t)$ can be arbitrarily small and cannot be known offline. Hence, Eq. (6) is a necessary feasibility condition for the loss requirement of $f_i \in F_{LF}$.

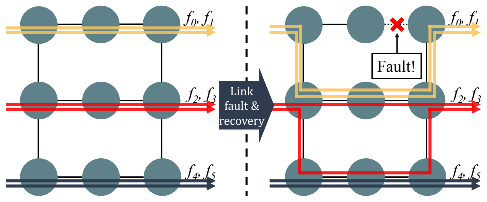


Fig. 6: Motivational MCP example: each link provides bandwidth of 100 Mbps and delay of 1 ms, and each flow has requirements of (33 Mbps, 6 ms)

Similar to Eq. (6), we can derive Eq. (7) as a necessary feasibility condition for the loss requirement of $f_j \in F \setminus F_{LF}$, by considering f_j does not pass a faulty link. $\square$

Although Theorem 1 provides a necessary condition to meeting the loss requirement of every flow, we need an effective way to find new paths at run-time that satisfy G2. To this end, we need to define a notion of time budget as the time duration we can spend time for the path recalculation step including a selection of flows to reroute, as follows.

Definition 1. *Suppose upon a link failure,* FR-SDN *completes the path restoration process successfully finding schedulable alternative paths for all flows. Also, suppose Theorem 1 holds. Then, we define the time budget L^*_{S2} for $f_i \in F_{LF}$ as follows (refer $R_i(t_0)$ in Lemma 1):*

$$L^*_{S2} = \min_{f_i \in F_{LF}} \big(R_i(t_0) + T_i \cdot (K_i - 1)\big) - \big(L_{S1} + L_{S3}\big). \tag{8}$$

Note that if we apply $R_i(t) = 0$ for every $f_k \in F_{LF}$, $L^*_{S2} \geq L_{S2}(|F_{LF}|)$ is equivalent to satisfying Eq. (6) for every $f_k \in F_{LF}$. Also, if the network does not support to calculate $R_i(t_0)$, we can use $R_i(t_0) = 0$.

As defined, the meaning of the time budget is the largest timing duration for the path recalculation step, which does not compromise the loss requirement of every flow that passes a faulty link. In the next section, we present how to efficiently utilize the time budget at run-time.

VII. Adaptive Path Calculation

In the previous section, we derived L^*_{S2}, the maximum available time to spend in the S2 step so as to guarantee the loss requirements upon successful path recalculation. In this section, we design our path recalculation method to increase the possibility of successful path finding within L^*_{S2}. To this end, our path recalculation method should be *deterministic*, in the sense that it terminates in a predetermined time (i.e., L^*_{S2}). Moreover, it should be *efficient* to improve its performance for a given limited running time.

Figure 6 gives an intuition on the path recalculation method. The figure shows a 3×3 grid network with 6 flows. In the figure, for simplicity, 6 flows have the same bandwidth requirement, and each link can accommodate up to 3 flows due to its capacity constraint. The figure illustrates a situation—when a link fault occurs, three flows (f_0, f_1, and f_3 are rerouted; it reroutes f_0 and f_1 to bypass the faulty link and one additional flow of f_3 in order to reroute f_0 and f_1 with an smaller additional delay to meet their delay constraints. This example implies that we can consider re-routing only some flows, instead of all flows, for successful path recalculation in a shorter running time.

Based on this observation, we present *adaptive path calculation*, which utilizes the number of flows to reroute (N_{route}) as a control knob in order to limit the running time of MCP computation. The minimum N_{route} is the number of flows that pass the faulty link. As shown in Figure 6, however, there are cases where it fails to find the new path. The more flows we take into account, the more successful the path recalculation would be. Hence, we take the best effort approach in choosing N_{route} to maximize N_{route} as long as our algorithm can complete within a given time budget L^*_{S2}. Then, the maximum N_{route} from L^*_{S2} can be derived as $N^*_{route} = \left\lfloor \frac{L^*_{S2} - \ell^{c-misc}}{\ell_{S2}} \right\rfloor$.

Figure 6 also illustrates the importance of choosing the right subset of flows to reroute. The problem of finding the minimum subset of flows that makes the solvable MCP problem is intractable. Therefore, we present a heuristic algorithm to find the best subset, named *nearest neighbor* (Adaptive-NN). This algorithm uses the notion of distance between links. Specifically, let us assume a link (u, v) experiences a fault. Then, all the links starting from node u (except (u, v)) have a distance of 1 with the faulty link (u, v). Then, all the links that are connected with distance-1 links have the distance of 2, except the incoming links of node u. This way, we can compute the distance of each link to the faulty link. Then, the distance of flow i is computed by the minimum distance of links that make up the path Φ_i. We make the subset of flows to reroute by the top-N^*_{route} minimum distance flows. For those flows that have the same distance, we select flows with the large bandwidth first. This is because re-routing the large bandwidth flows is more likely to help the path calculation.

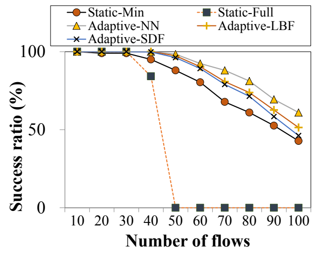
VIII. Evaluation

In this section, we evaluate FR-SDN by answering the following questions.

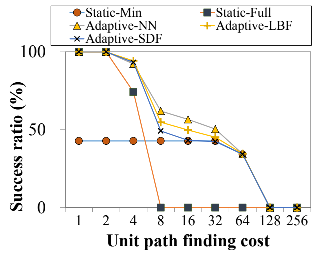
- How effective is our adaptive path recalculation approach in improving the success ratio of the path restoration (Section VIII-A)?
- How effective is FR-SDN in reducing the path restoration delay (Section VIII-B)?
- How does the path restoration delay affect the consecutive message losses of real-time flows (Section VIII-B)?

A. Path Finding Simulation

Simulation setup. We evaluated the performance of the path finding with varying the number of flows and the unit path finding cost. We estimated the success ratio of path restoration in a 5×5 grid network, with synthesized flow sets. A flow set is said to be *fault-resilient* for a link failure if its path restoration process correctly finds alternative paths for all flows without compromising any timing and loss requirements. The *success ratio* is defined as the percentage of the number of fault-resilient flow sets of the total number of generated flow sets. Each link had a delay of 40, and its bandwidth was randomly chosen between 50 and 100. We also randomly set the source and destination of each flow as well as the delay and bandwidth requirements between 160 and 240, 10 and 15, respectively. We generated 500 network and flowsets for a given number of flows and evaluated whether each path finding algorithm gives the solution within the delay budget. To exclude the unsolvable cases, we only considered cases in



(a) With varying the number of flows



(b) With varying the unit path finding cost

Fig. 7: Path restoration success ratio

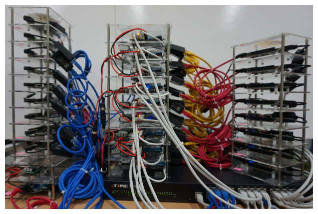
which Static-Full succeeds in finding the solution before and after the link fault, without considering its path recalculation time budget.

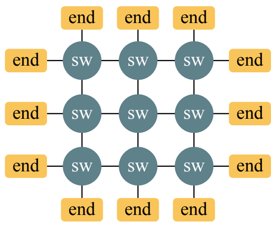
Baseline algorithms. To compare the path restoration performance with baselines, we considered various MCP approaches as follows:

- Static-Min reroutes the flows that are directly affected by the fault (i.e., flows that go through the faulty link).
- Static-Full reroutes the entire flows in the network.
- Adaptive-MCP limits the number of re-routing flows to be within available delay budget with the following heuristics.
  - Adaptive-NN selects the flows that go through the link nearest to the faulty link first (see Section VII).
  - Adaptive-SDF selects the flows with the smallest delay requirement first.
  - Adaptive-LBF selects the flows with the largest bandwidth requirement first.

Varying the number of flows. Figure 7(a) shows the path restoration performance between each algorithm with varying the number of flows and with a fixed unit path finding cost of 5. The figure shows that the success ratio of Static-Full drops significantly when the number of flows exceeds a certain threshold (40-50 in Figure 7(a)); this is because it requires a long computation time which exceeds the path recalculation time budget given by the flow requirements. On the other hand, the success ratio of Static-Min decreases gradually as the number of flows increases. This is because Static-Min does not suffer from the path recalculation time budget excess; however, since Static-Min only seeks to restore the minimal necessity flows, it could fail to find the feasible paths that satisfy the delay and bandwidth requirement of all flows.

Meanwhile, the success ratio of our adaptive approach, Adaptive-NN, dominates both Static-Full and Static-Min. Adaptive-NN performs better than Static-Min since it accommodates more flows for path recalculation within the delay budget; and it also performs better than Static-Full since it considers the available delay budget to finish the path restoration process. In addition, Adaptive-NN also shows the higher success ratio than other baseline heuristics, Adaptive-SDF and Adaptive-LBF in general. This result implies that the flows selected first by Adaptive-NN, which go through links nearest to the faulty link, are critical to the successful path restoration. In contrast, since Adaptive-SDF and Adaptive-LBF respectively select flows according to their given delay and bandwidth requirements, they may choose flows that go through the links far away from the faulty link, which are not critical to the successful path restoration.


(a) Testbed with 21 single-board computers



(b) Grid topology of the testbed

Fig. 8: Network testbed and its topology

Varying the unit path finding cost. Figure 7(b) shows the path restoration performance of each algorithm with varying the unit path finding cost of the MCP algorithm (l_{S2} in Section VI). The number of flows was fixed to 100 throughout the simulation. As the unit path finding cost changes, the number of flows that can be restored by the MCP algorithm within the path recalculation time budget may vary. However, the static MCP approaches, Static-Min and Static-Full, do not consider the time budget. Static-Full works well if the unit cost is fairly low, but always fails if the cost is greater than or equal to 8 since it exceeds the time budget. Static-Min works better than Static-Full when the unit cost is larger than 8 since it can finish within the time budget, but it always restores only the minimal set of flows which go through the faulty link.

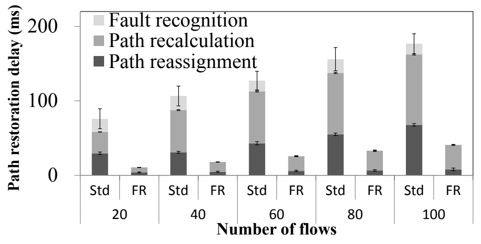
Our adaptive approach adjusts the number of flows to reroute according to the available budget. Therefore, Adaptive-MCP shows better performance than both of Static-Min and Static-Full regardless of the unit cost of path finding. This is expected since Adaptive-MCP considers the available time budget to determine how many flows to recalculate the paths; it can finish even if the unit cost is high, while Static-Full cannot. Likewise, Adaptive-MCP performs better than Static-Min since it always accommodates more flows than Static-Min within its time budget. Note that Adaptive-NN outperforms Adaptive-SDF and Adaptive-LBF in general[4]; this shows Adaptive-NN is a better strategy than Adaptive-SDF and Adaptive-LBF.

B. Experiment on Network Testbed

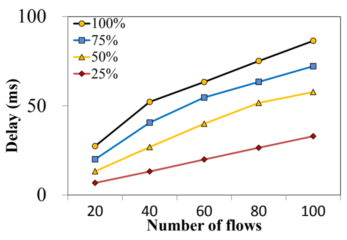
We implemented FR-SDN on top of Open vSwitch [17], the de facto standard software switch for OpenFlow implementation. We evaluated the performance of FR-SDN on a network testbed of 21 single-board computers.

Experiment setup. Experiments were performed on a network testbed (see Figure 8(a)), which consists of 12 end nodes (*Beaglebone-Black* [27] boards), 9 software switches (*Odroid-XU4* [28] boards), and a SDN controller (*Odroid-XU4*). To increase the connectivity of switch nodes, we equipped each switch node with additional 4 USB Ethernet interfaces (Realtek r8152 [29]) by using a USB2.0 hub. All nodes were connected via 100 Mbps Ethernet in a grid topology as shown in Figure 8(b), and each switch node had a dedicated Ethernet interface to communicate with the remote SDN controller. To realize the bandwidth allocation, we used HTB (*Hierarchical Token Bucket*) Linux Queueing Discipline with 8 sub-queues. Since the current OpenFlow does not

[4]In Figure 7(b), the success ratio of Adaptive-NN is slightly lower than that of Adaptive-SDF and Adaptive-LBF, when unit cost is 4. This is possible since we use a heuristic MCP algorithm, instead of an optimal MCP algorithm.



(a) Path restoration delay breakdown. Note that FR-SDN has the path recalculation time budget to calculate 25% of flows



(b) Path recalculation delay of FR-SDN with varying the number of path recalculating flows over the total number of flows

Fig. 9: Path restoration delay

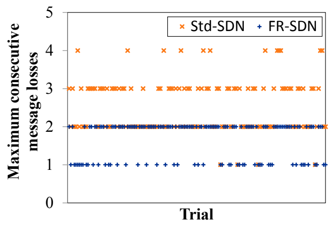


Fig. 10: The maximum number of consecutive message losses with random generated flowset

support dynamic queue configuration, we set each queue with a fixed reserved bandwidth. In addition, all switches and end nodes were synchronized by NTP (Network Time Protocol) with an accuracy of less than 1 ms.

Metric and baseline. We measured *path restoration delay* (L_{PR}) as an elapsed time from the instant at which a switch detects a fault to the instant at which all switches update their forwarding tables with new path rules. We measured each of *fault recognition delay* (L_{S1}), *path recalculation delay* (L_{S2}) and *path reassignment delay* (L_{S3}) as a breakdown of path restoration delay. In addition, we also measured *the maximum consecutive message losses* when the path restoration took place while real-time flows generate periodic messages.

To compare the performance with a baseline, we evaluated two distinct systems as follows:

- Std-SDN: A controller-driven path restoration with the Static-Full MCP algorithm.
- FR-SDN: A switch-driven path restoration with the Adaptive-NN MCP algorithm.

Path restoration delay. We measured L_{PR} with varying the number of flows, and broke it down into L_{S1}, L_{S2}, and L_{s3}. Figure 9(a) shows the result; note that each box plot and each error bar represent the average and the standard deviation, respectively, derived from 30 repeated trials. The result shows that L_{PR} of Std-SDN rapidly increases than that of FR-SDN, as the number of flows increases. This is because Std-SDN requires heavy OpenFlow communication and the path finding computation for all flows, while FR-SDN effectively eliminates OpenFlow communication and adjusts the amount of computation. Although L_{S1} involves a single OpenFlow communication on Std-SDN, it shows a long (up to 49 ms) and widely fluctuating delay. In contrast, FR-SDN effectively bypasses OpenFlow communication and reduces L_{S1} to less than 3 μs. L_{S2} of FR-SDN is also around a quarter of time compared to Std-SDN, since Adaptive-NN effectively adjusts the number of flows for path recalculation. Note that we set the recalculation time budget L^*_{S2} to cover 25% of the flows. L_{S3} is reduced with the highest ratio, because FR-SDN effectively replaces heavy OpenFlow communication for rule update with inter-switch path propagation.

Figure 9(b) shows L_{S2} according to the time budget with varying the total number of flows. Each legend indicates different time budget which can accommodate p% of flows for the path recalculation, where p denotes the number of flows to reroute over the number of total flows. This result is expected; as the number of flows increases, L_{S2} linearly increases; and as the time budget decreases, FR-SDN effectively reduces L_{S2} by adjusting the number of flows to recalculate the path.

Note that, unlike FR-SDN, it is difficult to apply Adaptive-MCP to Std-SDN because the safe time budget for path recalculation (L^*_{S2}, defined in Eq. (8)) cannot be derived due to the unpredictable delays (L_{S1} and L_{S3}) in Std-SDN. Instead, we can manually set the percentage of the number of path calculating flows and compare how the path recalculation delay (L_{S2}) of Std-SDN changes with varying the number of calculating flows. Note that a similar trend can be seen for Std-SDN as in the case of FR-SDN shown in Figure 9(b); L_{S2} is linearly decreased for all $|F|$ values as the percentage of the number of path calculating flows decreases from 100% to 25% (e.g., 94 to 35 ms when $|F| = 100$). However, in Std-SDN, limiting the number of path calculating flows has no effect on reducing L_{S1} and L_{S3}. On the other hand, FR-SDN significantly reduces L_{S1} and L_{S3} via the proposed switch-driven path restoration approach and enables adaptive path recalculation with tight upper-bounds on the delays, resulting a much smaller and predictable path restoration delay compared to Std-SDN as shown in Figure 9(a).

Effect on real-time flows. To show the effectiveness of FR-SDN to guarantee the maximum tolerable consecutive message loss requirement of real-time flows, we undertook an experiment with randomly synthesized real-time flows. For each experiment, we generated a set of up to 14 real-time flows. For each flow, the source and destination nodes are randomly selected, the period is randomly chosen from 25 ms to 100 ms according to the log-uniform distribution, the bandwidth is randomly selected between 5 Mbps to 10 Mbps, and the message size is randomly determined to be transmitted within 80%-90% of the period according to the given bandwidth. Note that since the reserved bandwidth of the queue is fixed, each flow was associated with the queue that can support higher bandwidth than the flow requirement. The maximum tolerable consecutive message loss requirement of each flow was fixed as 2, for ease of presentation. Note that all flowset used in the experiment passed the feasibility test in Theorem 1 with a conservative delay bound of each delay factor obtained by measurement (i.e., L_{S1} and L_{S3} of 3 μs and 10 ms, respectively, and the bounded L_{S2} with l_{S2} and l_{c-misc} of 1.4 ms and 0, respectively). After 2 seconds from the start of each experiment, we emulated a link fault by taking the interface down (i.e., `ifdown` system command) on a randomly selected switch port that is connected to another switch. Since the link status monitor takes a while to detect a link fault (i.e., up to 15 ms in our setup), we consider this delay to derive the path

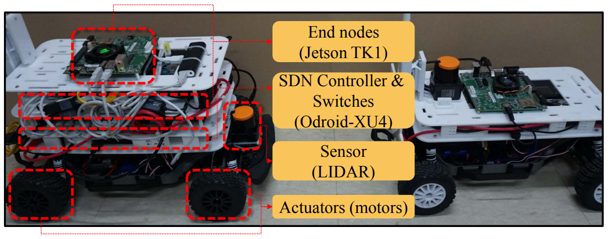


Fig. 11: 1/10 scale autonomous vehicles

recalculation budget.

Figure 10 shows the maximum number of consecutive message losses of all flows in each flowset. The figure shows that Std-SDN results in up to 4 consecutive message losses which violate the flow requirement. This is because of the long path restoration delay imposed by Std-SDN. In contrast, FR-SDN effectively reduces the maximum consecutive message losses compared to the Std-SDN; this is because of the reduced delay for the path restoration. Besides, FR-SDN strictly limits the maximum consecutive message losses by 2, thanks to the bounded delay of the path restoration.

IX. Case Study: Autonomous Vehicle

In order to show how real world systems benefit from FR-SDN, we conducted a case study, supporting the *Adaptive Cruise Control* (ACC) system on a 1/10 scale autonomous vehicle [30].

Adaptive Cruise Control. Adaptive Cruise Control is a system that automatically adjusts the speed of a car to continuously maintain the safe distance, by constantly measuring the distance from the car driving ahead. Since it alleviates the driving effort, it is considered as the Level 1 automation in the SAE J3016 standard [31].

Experiment setup and scenario. We implemented a 1/10 scale autonomous car, which is extended from the F1/10 autonomous racing platform [30], as shown in Figure 11. The car consists of the Traxxas Rally 1/10 body with actuators (i.e., motors) [32], a LIDAR sensor (Hokuyo UST-10LX [33]), Jetson TK1 [34] boards for end nodes, and Odroid-XU4 [28] boards for switch/controller nodes.

In each experiment, two cars ran in a way that a *following* car follows a *leading* car while aiming to maintain a regular distance (i.e., the reference distance of 1.5 meters). Note that the two cars used the PID controller to drive and keep the distance. The leading car drove itself along the straight corridor of 15 meters while alternating accelerating and braking at the interval of one second. Figure 12(a) depicts the network topology of the vehicle system, and only the following car is equipped with the networked system in the figure.

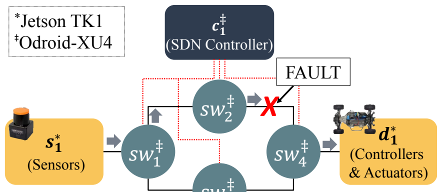
In each experiment, in the following car, the sender node s_1 transmitted the LIDAR sensor data flow to the destination node d_1 through the switches sw_1, sw_2, and sw_4. In the middle of each experiment (i.e., 6 seconds after departure), we emulated a fault on the link between sw_2 and sw_4 by taking the interface down (i.e., `ifdown` command). The switch sw_2 detected the fault and sought to change the route with FR-SDN and Std-SDN (see Section VIII-B for more details). Note that the LIDAR flow generated messages with the period of 25 ms, the amount of 8 Kbytes, and the bandwidth of 3.0 Mbps. To reflect a real-world situation that multiple flows share the network, the switches have 100 background flows sharing the network. Note that each flow required the period of 20 ms, the amount of 1Kbytes, and the bandwidth of 0.5 Mbps. When the fault occurs, the LIDAR flow and 50 out of 100 background flows are re-routed to the new path of sw_1, sw_3, and sw_4.

To compare the control performance of each system, we used three different scenarios: No-Fault, Std-SDN and FR-SDN. In No-Fault, no link fault happened; in Std-SDN and FR-SDN, a link fault happened 6 seconds after from the departure time of the following car.

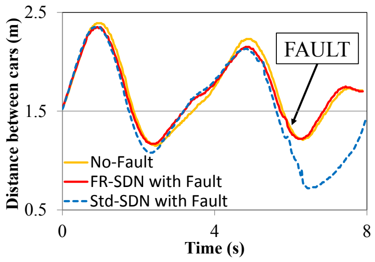
Implication. Figure 12(b) shows the distance between two cars over time. Note that the distance samples were measured by the LIDAR with the sampling rate of 40 Hz. We consider the case of No-Fault as a reference control performance case. The figure shows that Std-SDN cannot keep the reference distance after it suffers from the fault with the distance reduced down to 0.7 meters. On the other hand, No-Fault and FR-SDN maintain the distance more than 1.1 meters. This is because it takes longer to perform path restoration on Std-SDN against the link fault. During the path restoration process, the LIDAR messages cannot reach the destination; thereby, the PID controller cannot have enough sensor data to properly control the car. In contrast, FR-SDN shows stable control behavior comparable to No-Fault despite the link fault. This is because FR-SDN incurs a much shorter path restoration delay than Std-SDN and is thus able to limit consecutive message losses during the path restoration process.

To evaluate how well each path restoration mechanism preserves the control stability, we repeated experiments 20 times. We evaluate the control performance by comparing the control behavior with a reference case, which is selected from the No-Fault trials. Note that the reference has the median control performance among all No-Fault trials. Then, we quantify the control performance of each trial as the performance index of the integral of the absolute error (IAE) criterion [35], defined as $\int_A^B |s(t) - r(t)| dt$, where $s(t)$ and $r(t)$ denote the control output and reference value at time t, respectively. The performance index represents an accumulated state error, i.e., a deviation from the desired state. Thus, a large value indicates a larger deviation from the desired states, or worse control performance. We used the measured distance between two cars and the desired safety distance as the control output and the reference value, respectively; to measure the effect after the link fault, we set A and B to 6 and 7.5, respectively.

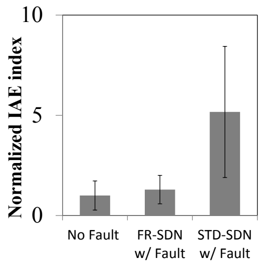
Figure 12(c) shows the average and standard deviation of the performance index for No-Fault, FR-SDN and Std-SDN, normalized to the average of No-Fault. Note that No-Fault also has non-zero IAE. This is because we controlled the car using a simple PID controller; hence, control performance was slightly different for each trial even with the same setup in all trials. The figure shows that FR-SDN results in the small IAE comparable to No-Fault; it implies that FR-SDN effectively preserves the control performance despite the link fault, thanks to the minimized LIDAR message losses with the short path restoration delay. Yet, FR-SDN has a slightly higher IAE than No-Fault since some LIDAR message losses (1-2 message losses) are inevitable during the path restoration process. On the other hand, Std-SDN has an IAE of more than 5 times higher than No-Fault. This implies that Std-SDN cannot support stable control performance when the link fault happens, due to



(a) Network topology in the vehicle system



(b) Distance changes over time



(c) Normalized IAE index

Fig. 12: Case study with the 1/10 scale autonomous vehicles: Adaptive Cruise Control

the considerable amount of LIDAR message losses during the path restoration. In the worst case, Std-SDN results in the IAE 9.8 times larger than No-Fault (i.e., the absolute value of 42.03); it represents that the worst trial shows the average difference of 0.7 meters with the reference trial in each control tick.

The case study implies that the path restoration delay is critical to preserve the control stability; and the fast path restoration thanks to FR-SDN helps to improve the safety of real-world control systems.

X. Related Work

Fault tolerant Ethernet-based CPS network. Ethernet standards for CPS, such as IEEE TSN [7] and AFDX [1], support *fault-free* communication by employing redundant packet transmission which simultaneously transmits replicated packets through multiple routes. In addition, IEEE TSN also supports *fault-resilient* communication relying on path protection approach [36]; upon detecting a link fault, it can change the path to the pre-allocated backup path. Although they provide fault tolerance for packet transmission, they incur inefficient resource use for the duplicated packet transmission and proactively allocated bandwidth for the backup path. Instead, our path restoration approach helps to efficiently utilize the network resources while providing fault resiliency.

Fault tolerant SDN-enabled networks. To support fault resilient packet transmission through SDN, it employs two major ways of fault recovery: *path restoration* and *path protection*. In studies for *path restoration* [15], [16], a fault detecting switch reports it to the controller. The controller then seeks to find new paths of the flows that go through the faulty link; although it can generate alternative paths considering run-time network status including the fault situation, it yet imposes a long and unpredictable delay to communicate with the controller, which is not acceptable for the real-time flows. FR-SDN changes the path restoration way from the controller-driven to the switch-driven, and consequently reduces and bounds the delay. Note that, although FR-SDN applies a switch-driven approach so as to enable fast and predictable path restoration, it respects most of essential SDN principles, such as dynamic network management and global network view.

In studies for *path protection* [37]–[40], each switch stores pre-computed alternative paths for fault cases; it immediately updates its forwarding table with the alternative paths upon a link fault, while minimizing the fault recovery delay. However, it incurs inefficient resource use to store alternative paths of every single fault case; moreover, it should recalculate all the alternative paths upon any network status change (i.e., adding a new flow). In contrast, FR-SDN can efficiently utilize the resource, thanks to the on-demand path recalculation. A hybrid approach [14] proposes a two-step fault recovery; it first applies pre-computed alternative paths that only consider end-to-end connectivity, and later, it replaces the rules with the paths computed by the controller considering QoS requirements. This approach still incurs a long delay; the timing requirements of flows can be guaranteed after the second step is done.

Switch-driven network management on SDN. MC-SDN [41] proposes switch-driven network management to support a fast and predictable mode change for mixed-criticality scheduling. When a mode change happens, switches notify the event to each other through inter-switch signal propagation, and immediately update their forwarding table with proactively stored rules. Although MC-SDN enables the fast and predictable rule update, it is not suitable for the path restoration, because it requires to store alternative forwarding rules in advance of the event (i.e., link failure). In contrast, FR-SDN effectively supports the path restoration, by generating alternative rules after the event.

XI. Conclusion and Future Work

We presented the design and implementation of FR-SDN that supports fast and predictable path restoration to meet the fault tolerance constraints of firm real-time flows. This is the first work on adaptive path restoration in SDN networks, aiming at safe fault recovery from link failure without violating any loss requirement. FR-SDN was designed to completely change the way of performing path restoration from SDN controller-driven to switch-driven, enabling path restoration operation with minimal and bounded delays. FR-SDN not only provides a feasibility test for meeting the loss requirement of individual firm real-time flows, but also employed an adaptive MCP technique that selectively reroutes only some flows so as to perform path recalculation effectively within a limited time budget. We developed the prototype of FR-SDN to evaluate its effectiveness in a real-world system such as a 1/10 scaled autonomous vehicle; and the extensive evaluation and case study demonstrated that FR-SDN significantly improves the reliability of real-time communications.

As future work, an important aspect is scalability to support large network systems, such as smart factories and autonomous vehicles. In order to make FR-SDN more scalable, it requires to restrict signal propagation range and path finding search space. A hierarchical design approach would be helpful to effectively bound signal propagation range and path finding search space within each sub-network. In addition, we would like to extend FR-SDN to support multiple link failures.

Acknowledgements

This work was supported in part by BSRP (NRF-2015R1D1A1A01058713), IITP (2014-0-00065, Resilient Cyber-Physical Systems Research), ERC (NRF-2018R1A5A1059921), MSIT (DGIST Start-up Fund Program, 2018080005), MSIT (2019R1A2B5B02001794), and MSIT (2017M3A9G8084463, Bio & Medical Technology Development Program).

References

[1] J.-P. Moreaux, "Data transmission system for aircraft," 2005, US Patent 6,925,088.

[2] "Open Alliance," http://www.opensig.org/.

[3] "AUTOSAR Classic Platform Standard 4.3.0," 2016.

[4] IEEE, "IEEE standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks," *IEEE Std 802.1AS*, pp. 1–292, March 2011.

[5] ——, "IEEE standard for local and metropolitan area networks– bridges and bridged networks - amendment 24: Path control and reservation," *IEEE Std 802.1Qca*, pp. 1–120, March 2016.

[6] ——, "IEEE standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv*, pp. 1–57, March 2016.

[7] ——, "IEEE standard for local and metropolitan area networks - frame replication and elimination for reliability," *IEEE Std 802.1CB*, 2017.

[8] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44(12), pp. 1443–1451, 1995.

[9] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in *IEEE Real-Time Systems Symposium (RTSS)*, 1995.

[10] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50(4), pp. 308–321, 2001.

[11] J. Lee and K. G. Shin, "Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective," *Journal of Systems and Software*, vol. 126, pp. 45–56, 2017.

[12] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and V. Verdugo, "A scheduling model inspired by control theory," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2017.

[13] H. S. Chwa, K. G. Shin, and J. Lee, "Closing the gap between stability and schedulability: A new task model for cyber-physical systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018.

[14] 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.

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

[16] S. Hegde, S. G. Koolagudi, and S. Bhattacharya, "Path restoration in source routed software defined networks," in *Proceedings of the International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017.

[17] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[18] 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 *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[19] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, pp. 95–116, 1984.

[20] S. Shigang Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 1998.

[21] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD)," RFC 5880, Jun. 2010.

[22] "Cisco NX-OS Software," https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/nx-os-software/data_sheet_c78-652063.pdf.

[23] "PICOS®-Pica8," https://www.pica8.com/wp-content/uploads/PICA8-Datasheet.pdf.

[24] Open vSwitch, "An Open Virtual Switch," http://openvswitch.org/.

[25] 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.

[26] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[27] "BeagleBone Black," https://beagleboard.org/black.

[28] "Odroid-XU4," https://magazine.odroid.com/odroid-xu4.

[29] "RealTek RTL8152," http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PNid=14&PFid=55&Level=5&Conn=4&ProdID=323.

[30] "F1/10 Autonomous Racing Competition," http://f1tenth.org/.

[31] "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems ," *SAE International Standard J3016*, 2014.

[32] "Traxxas Models," https://traxxas.com/products/showroom.

[33] Hokuyo Automatic Co., "UST-10LX Specification," http://www.senteksolutions.com/application/files/2414/7196/1936/UST-10LX_Specifications.pdf.

[34] NVIDIA, "NVIDIA Jetson TK1 developer kit product page," http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html.

[35] P. Sarkar, *Advanced process dynamics and control*. PHI Learning, 2014.

[36] IEEE, "IEEE Draft Standard for Local and metropolitan area networks–Link Aggregation," *IEEE P802.1AX*, 2014.

[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] X. Zhang, Z. Cheng, R. Lin, L. He, S. Yu, and H. Luo, "Local fast reroute with flow aggregation in software defined networks," *IEEE Communications Letters*, vol. 21, no. 4, pp. 785–788, April 2017.

[39] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sansò, "Fast failure detection and recovery in sdn with stateful data plane," *Int. Journal of Network Management*, vol. 27, 2017.

[40] Y. Lin, H. Teng, C. Hsu, C. Liao, and Y. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2016.

[41] K. Lee, T. Park, M. Kim, H. S. Chwa, J. Lee, S. Shin, and I. Shin, "MC-SDN: Supporting mixed-criticality scheduling on switched-ethernet using software-defined networking," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2018.