Telecommunication Systems (2024) 85:277–288
https://doi.org/10.1007/s11235-023-01084-4

Quantitative measurement of link failure reaction time for devices with P4-programmable data planes

David Franco[1] · Marivi Higuero[1] · Eder Ollora Zaballa[2] · Juanjo Unzilla[1] · Eduardo Jacob[1]

Accepted: 21 November 2023 / Published online: 19 December 2023

Abstract
Quick handling of link failures remains a challenging issue in current communication networks, although it is crucial to many routing algorithms. Link failures are the leading cause of packet losses and delays, therefore, failure recovery is tied to stringent requirements for certain services, such as the sub-50 millisecond completion time for carrier-grade networks, which is sometimes difficult to achieve in traditional routing schemes. For this reason, fast recovery strategies are key pillars of modern communication networks. In this paper, we demonstrate the benefits of the devices with Programmable Data Planes (PDP) for fast reacting to link failures. We first review the link failure detection, reaction and recovery procedures and then we discuss the main fast failure recovery mechanisms employed by different types of devices in current communication networks. In addition, we present a novel method to measure the link failure reaction time of an Intel Tofino switch with PDP, as well as the results obtained when measuring such time using real hardware equipment. Our results show that such hardware devices provide a failure reaction time in the order of microseconds, with an average of **472.88** µs, which poses PDP as a key technology to achieve zero packet loss and zero delay failure recovery.

Keywords Failure reaction time · Failure recovery · Programmable data plane · P4 · Software-defined networking

1 Introduction

The ongoing evolution of Internet services and applications puts the focus on deploying reliable and efficient network architectures capable of providing five-nine availability rates. In this sense, link failures are the most frequent failures in a network [1] and thus the main cause of packet loss and delay that lead to service degradation or even disruption. In consequence, due to the stringent requirements demanded by some services, handling link failures and providing high network resilience has turned into a fundamental task of any routing algorithm. These resilience mechanisms can be implemented either in the control or the data plane. The control plane relies on a global view of the network and is responsible for determining the paths along which packets are sent, while the data plane is in charge of the packet processing logic in individual switches, thus the data plane moves packets from one interface to another based on control plane rules. Traditionally, network failure conditions have been addressed in the control plane, while the data plane was responsible for forwarding packets at line rate. Widely deployed routing schemes like Open Shortest Path First (OSPF) include control plane mechanisms which leverage control plane message exchanges and computation to determine how to recover from link failures.

However, the slow reaction of the control plane causes additional delay and packet loss and it is unacceptable for many services and applications [2, 3], mainly due to (1) high convergence delays of decentralized algorithms, or (2) high communication delay between the data plane devices and the centralized control plane that performs the path re-

✉ David Franco
david.franco@ehu.eus

Marivi Higuero
marivi.higuero@ehu.eus

Eder Ollora Zaballa
eoza@dtu.dk

Juanjo Unzilla
juanjo.unzilla@ehu.eus

Eduardo Jacob
eduardo.jacob@ehu.eus

[1] Department of Communications Engineering, University of the Basque Country, Plaza Ingeniero Torres Quevedo 1, 48013 Bilbao, Spain

[2] Department of Electrical and Photonics Engineering, Technical University of Denmark, Anker Engelunds Vej 101, 2800 Kongens Lyngby, Denmark

computation. Therefore, given the disparity in timescales between packet forwarding in the data plane —in the order of a microsecond— and control plane convergence —hundreds of milliseconds—, there is a trend to shift the responsibility for path reconfiguration to the data plane where line-rate packet processing can benefit the failure recovery time [4]. Indeed, most modern networks support different kinds of fast-recovery mechanisms which leverage pre-computed alternate paths that are directly activated in case of failure, without the interaction of the control plane. The control plane is hence just responsible for pre-computing the failover paths. For example, WAN networks leverage IP Fast Reroute (IPFRR) [5] or MPLS Fast Reroute (MPLS-FR) [6] mechanisms to deal with failures on the data plane, OpenFlow uses *FAST-FAILOVER* groups [7] to activate alternate routes when a link goes down, and BGP relies on BGP Prefix-Independent Convergence (BGP-PIC) [8] for a quick convergence after primary path failures.

The advent of high-speed switching devices with Programmable Data Plane (PDP) opens the door to more sophisticated recovery strategies to face network failures. They do not depend on fixed fast failure recovery techniques, such as OpenFlow's *FAST-FAILOVER* group, or fast rerouting protocols [5]. On the contrary, they provide the necessary tools to build customized fast recovery algorithms directly in the data plane. Therefore, PDP-based recovery strategies allow optimizing the response of the network by (1) implementing custom fast-recovery algorithms, and (2) running those algorithms at line rate. Programming Protocol-independent Packet Processors (P4) [9] is the most extended high-level hardware description language used to specify the packet processing pipelines of the PDPs. P4 is target independent, which means that it does not depend on low-level placement details, and it can be used both on hardware and software targets. It provides the required flexibility to build custom algorithms and the tools to guarantee their performance at line rate when using hardware targets.

Failure detection and recovery are widely used terms in the literature. Failure detection is the time that takes the node to be aware of the failure, while the failure recovery process implies the application of the necessary forwarding modifications to restore the communication path in the node —or in the whole network. However, devices with PDP allow us to measure a new phase, hereby named failure reaction time, which is defined as the time elapsed from the link failure occurrence until the first change is applied in the data plane as a response to it. In order to minimize packet loss, network devices need to optimize this time, which is usually hardware-dependent, so they can quickly react to link failures.

This paper focuses on characterizing the link failure reaction time of devices with P4-PDP as a key enabler for applying advanced fast recovery mechanisms in the data plane. We first study the reaction time when a link failure occurs, breaking down the process into a number of separate events. Then we evaluate the performance of a hardware Application-Specific Integrated Circuit (ASIC) on reacting to failure events, for which we measure the failure reaction time of an Intel Tofino switch. Our measurements show that Tofino hardware switches can react to failures directly from the data plane on a microsecond scale, which opens the door to implementing data plane based failure recovery strategies that improve the performance of data center and core networks operating at tens of Gbps. Therefore, the main contribution of this paper is threefold. On the one hand, we provide a thorough explanation and review of (1) the failure detection, reaction and recovery procedures and the existing fast recovery mechanisms to quickly recover from link failures in current communication networks. On the other hand, we (2) review the existing measurement techniques and present the designed method to measure the failure reaction time of an Intel Tofino switch with PDP and (3) show the test results. The tests are performed using real hardware equipment and the results show that the Intel Tofino switch with PDP reacts to link failures in the order of microseconds, with an average of 472.88 μs.

The remainder of the paper is organized as follows. First, Sect. 2 reviews the related work regarding link failure detection and recovery techniques. Next, Sect. 3 goes deeper into the concepts of failure detection, reaction and recovery. Then, Sect. 4 introduces the different fast failure recovery techniques implemented by the networking devices. After that, Sect. 5 describes the method to measure the link failure reaction time in hardware devices with PDP and shows the obtained results. Finally, Sect. 6 brings the main conclusions of this work to the fore.

2 Related work

The topic of failure reaction has not been widely addressed in the literature, however, it is present in many works as part of the more global concept of failure recovery, which involves not only detecting and informing about the failure but also the process of restoring the communication. Therefore, this section analyzes the literature regarding the data plane failure detection and recovery techniques, focusing on those solutions that decompose the failure recovery time and/or describe their approaches to measure it.

2.1 Approaches to link failure reaction

The literature describes the failure recovery as a process that involves detecting and reacting to the failure, and finally restoring the communication. Therefore, many works in the literature that look at the recovery time are useful to

understand the order of magnitude of the reaction time. For instance, authors in [1] present a survey that shows how fast recovery is addressed in the data plane. At layer 2, mechanisms such as RSTP, and other custom implementations, show recovery time values around 50 ms. However, those values strongly depend on the complexity of the topology. At layer 3, the authors highlight that increasing the number of rerouted LSPs in MPLS networks, or IGP prefixes in IP networks, causes higher packet losses and a non-linear increase of the recovery time, even beyond 50 ms. They also address fast recovery in programmable networks, where they describe several mechanisms based on OpenFlow and P4. They do not show specific recovery time values but they conclude that PDPs enable more efficient recovery mechanisms in terms of latency and throughput.

Furthermore, authors in [10] present a solution to speed up the failure reaction mechanism in RouteFlow [11], an OpenFlow framework to run traditional routing protocols. They implement a patch that allows notifying the corresponding process in the control plane —i.e., a virtual machine— when a link failure is detected in a physical OpenFlow node. Then, the routing protocol running in the virtual machine triggers the recovery procedure to overcome the failed link.

2.2 Measurement methods

Regarding the methods to measure the failure recovery time, there are several approaches that leverage **computer simulations** to evaluate the performance of their solutions. For instance, authors in the aforementioned [12] use computer simulations to evaluate their IPFRR OpenFlow implementation compared to the standard OSPF recovery process. Moreover, authors in [13] analyze the effect of the designated router's placement on overall convergence time in OSPF networks by elaborating a mathematical analysis of the problem and modeling the convergence time.

On the other hand, most of the analyzed solutions rely on **emulation platforms** to test their failure recovery techniques. Authors in [2] provide an improved data plane based fast rerouting mechanism to avoid inefficiencies when recovering from link failures. They use a version of the Mininet [14] emulation platform that supports behavioral model version 2 (bmv2) [15], a P4 software switch for testing and debugging P4 data planes, to evaluate their solution. They show an average link failure recovery time of one second compared to the 7 s obtained by a control plane based mechanism. Similarly, authors in [16] design and implement a P4-based fast failure recovery mechanism using Multiple Routing Configurations (MRC) [17]. They describe the design and the implementation of the failure detection and packet forwarding functions and they model their design in Mininet/bmv2, using the Scapy library of Python to measure the failure recovery time. P4Neighbor [18] is a multi-failure recovery system based on PDP that precomputes and encapsulates backup paths into the packet header and calculates backup paths for each link instead of each flow or host. To measure the link failure recovery time, they emulate the network using Mininet/bmv2, and they compare their proactive solution with a modified version of P4Neighbor that recovers from link failures in a reactive manner. Their solution provides recovery times from 19 ms to 51 ms depending on the number of switches in the alternate path. Additionally, authors in [19] develop a proactive fast failover mechanism in the data plane to deal with link failure and congestion problems in SDN/OpenFlow. They implement their fast failover mechanism using the Ryu controller and Open vSwitch software switches in Mininet. They show that the average recovery time of their solution is less than 40 ms, compared to the hundreds of ms of the control plane based fast restoration mechanism.

The literature also shows the use of **discrete event simulators** to evaluate the failure recovery time, such as in PURR [20]. PURR is a fast reroute primitive for programmable data planes that provides low failover latency and high switch throughput by avoiding packet recirculation. Simulations in ns3 are carried out to evaluate the impact on the flow completion time when using PURR with regard to another approach based on recirculating packets.

Finally, there are many works using **commercial hardware** to test their solutions. Authors in [21] propose the use of pre-provisioned alternate paths, as well as a control plane based failure detection scheme that leverages probe packets. They evaluate their solution using real Commercial-Off-The-Shelf (COTS) equipment, PCs running *ofsoftswitch13* [22], which shows an average recovery time of about 60 ms. Also, authors in [23] present a failover scheme with per-link BFD sessions and preconfigured primary and secondary paths computed by an OpenFlow controller. They perform their experiments on both software —Open vSwitch— and hardware —Pica8 P3920— switches using pktgen [24], a packet generator that operates from kernel space. When using Loss Of Signal (LOS) based failure detection, they show a recovery time of 4759 ms and 1697 ms in software and hardware respectively, which is unacceptable in carrier-grade networks. However, when using their BFD-based solution, only supported by software switches, they show recovery times between 3.4 ms and 42.1 ms depending on the BFD transmit intervals.

We conclude that none of the aforementioned mechanisms focuses on measuring the failure reaction time nor uses real switching equipment to measure it. Therefore, to the best of our knowledge, this is the first paper that provides a method to measure the failure reaction time in commercial hardware-based devices with PDP.

3 Link failure detection, reaction and recovery

The **failure detection time** is the time that takes for the local control plane or Operating System (OS) to identify a link as not operative to send or receive data —see Fig. 1. This time depends on the mechanism employed to identify the failure and can range from a few microseconds, in case of LOS failures, to tens of seconds if a *Hello*-based failure detection mechanism —also known as probe packets— is used. For instance, a LOS-based link down detection using packet over SONET might take tens of milliseconds, while detecting link down via lost of neighbor adjacency —e.g., sending Link Status Advertisement (LSA) messages in OSPF— might take up to 40 s [13, 25]. Another example is the Ethernet 10GBase-T, which according to its physical layer definition for 10 Gbps [26], a link will only be considered failed if the $link\_fail\_inhibit\_timer$ —from 2000 ms to 2250 ms— has expired and the link has still not gone into the link status OK state. Similarly, the commonly used Small Form-factor Pluggable (SFP) transceivers have their own mechanism to detect link failures via LOS. According to the SFF-8431 standard [27], the Rx_LOS signal indicates that the received signal strength is below the specified range, which typically points to non-installed or broken cables. Then, it defines a 100 µs delay from the LOS occurrence to the assertion of the Rx_LOS, which means that it waits that time before considering the link as failed. Link failures can occur at level 1/2, usually caused by LOS, or they can be at upper layers, affecting the inter-operation of higher-level protocols without the connection at lower layers having been interrupted. The OS of network devices is configured to generate OS-level interruptions as a response to local link failures caused by LOS. However, detecting errors at upper layers is often more difficult as keep-alive probe packets are required to monitor the liveliness of such applications and services. Another difficulty emerges when two devices are connected at layer 2 through a third device. Let's say two routers, R1 and R2, are connected through a switch S1 and they have a router adjacency. When the link R2-S1 goes down due to LOS, R1 will not detect the link failure event and it will keep sending packets to R2.

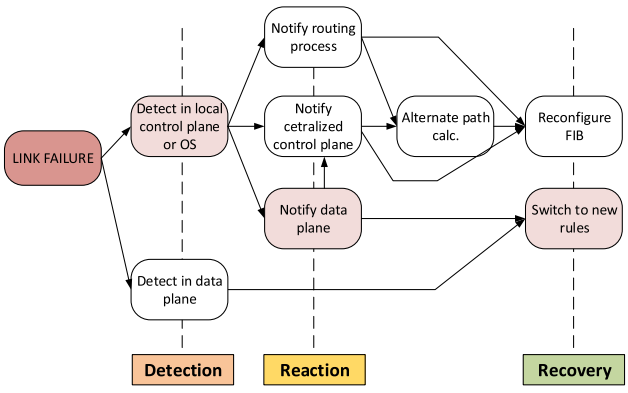


Fig. 1 Description of the link failure detection, reaction and recovery processes

According to [28], there are two families of failure detection mechanisms in IP networks: (1) lower layers failure notification and (2) *Hello*-based mechanisms. The former is used to overcome situations such as in the aforementioned example, in which an additional mechanism is required to notify R1 about the failure in the link S1-R2. The latter consists in sending periodical *Hello* messages to monitor the liveliness of a link or service. BFD [29] is a low-overhead and protocol-independent *Hello* mechanism that is oriented to quickly detect link failures in the forwarding plane. For instance, OpenFlow uses LOS and BFD detection to implement path restoration and path protection mechanisms respectively [3, 30]. In restoration, alternate paths can be either preplanned or dynamically allocated, but resources are not allocated until a failure occurs, thus additional signaling is required. LOS detection causes a port to change its state from up to down and vice versa. When there is a change in the state of the port, the OpenFlow switch sends a port-status message [7] to notify the controller. The reactive nature of path restoration makes LOS detection more suitable for that mechanism. However, for path protection, alternate paths are always pre-planned and reserved before the failure occurs. In these scenarios, BFD provides end-to-end link failure detection and allows for preconfiguring an alternate path at the ingress node —the node that usually has the alternate route for the flow. Both mechanisms can be linked to OpenFlow's *FAST-FAILOVER* group tables —explained in Sect. 4— in order to speed up the response time in case of link failure.

Once the failure is detected, the local control plane or OS needs to communicate with the entity responsible for taking the appropriate actions to overcome the error. Therefore, we define the **failure reaction time** as the time since the failure occurs until it is notified to the entities in charge of triggering the recovery process. In this case, there are up to three different options depending on the network device or configuration we are using.

- If it is a legacy device, the process in charge of the I/O devices will tell the routing process about the failure detection.
- In the case of centralized architectures, such as in SDN, the error will be either notified to the centralized controller using the southbound interface —e.g., port-status message in OpenFlow— or directly detected by the controller using Link Layer Discovery Protocol (LLDP) signaling —detection and notification at once.
- In other implementations, e.g., devices with PDP or OpenFlow *FAST-FAILOVER* groups, the local control

plane or OS directly notifies the data plane so it can trigger the process to recover the failure —precomputed alternate routes are common in these cases.

Some SDN implementations perform the failure detection directly in the centralized controller using $Hello$ packets. In such scenarios, the failure detection is limited by the time that is considered to mark a link as failed when $Hello$ packets are missing. Additionally, other custom implementations, usually for devices with PDP, perform $Hello$-based failure detection in the data plane [31]. These scenarios are out of the scope of this paper, as they tend to increment the failure detection time compared to LOS detection.

After the reaction, the entities in charge need to run their specific actions to overcome the failure. The **link failure recovery time** is the time elapsed since the failure occurs until the affected node performs all the changes to overcome the failed link. For the centralized control plane and the process in charge of routing, depending on whether they have precomputed alternate routes or not, they will directly reconfigure the Forwarding Information Base (FIB) —replace old entries— in the data plane or they will need to calculate the alternate path first. If the failure notification arrives at the data plane, the process is similar. The data plane can have precomputed alternate routes and directly switch to them, or if it does not, it will notify the centralized controller so it can perform the required actions. Again, the centralized controller can run the path recalculation or use precomputed alternate routes. Note that the data plane is not capable of reconfiguring a FIB, this process is delegated only to the control plane, so the data plane is limited to switching from one route to another according to predefined state information.

Network recovery is a more complex process in which multiple nodes of the network cooperate to reestablish the end-to-end paths when a link goes down. The **network failure recovery time** involves the propagation of the information about the failure along the network; therefore, it is highly dependent on the network topology and size. This time also depends on the network architecture, for example, in traditional distributed architectures, the routing algorithm needs to propagate the failure along the network, so every node can modify its routing tables. This is a time-consuming process that may cause transient loops until the convergence of the algorithm is achieved. On the contrary, in SDN/OpenFlow the centralized controller is responsible for configuring every node to overcome the failed link. Thus, the path computation is performed in a centralized entity that has the whole view of the network, which avoids the need to wait for the convergence of a distributed routing algorithm and allows configuring all nodes at once —through the southbound interface. This process reduces the global network recovery time as long as the Data-to-Control Plane Interface (D-CPI) latency is not high. In this sense, network architectures based on devices with PDP benefit from the centralized path computation capabilities while avoiding additional D-CPI communication delay. Thus, they allow the implementation of failure recovery directly in the data plane, without involving the control plane.

4 Fast recovery mechanisms

In Sect. 3, failure recovery is defined as the process to restore the connectivity in the network when a link —it can be generalized to a node— fails. As described before, the processes involved in detecting, reacting and recovering from link failures are time-consuming. Therefore, most of the existing network devices implement fast recovery mechanisms that aim at decreasing the time that takes to restore the connectivity in the network after a failure. The majority of them leverage precomputed alternate routes to create transient paths that surpass the failed link until the connectivity is fully restored.

In this way, this section aims at comparing the different fast recovery strategies that can be found in current network devices. According to the network architecture, we identify the following types of devices:

- Traditional devices: network devices based on traditional TCP/IP protocol stacks, where the control and data planes are located in the same physical device. The control plane of these architectures is distributed along all the network devices, which cooperate to make routing decisions.
- OpenFlow devices: they form the data plane of SDN-based networks. OpenFlow defines a static data plane that is controlled from a programmable control plane.
- Devices with PDP: devices that allow the definition of customized data plane algorithms, and which also have a separated control plane that can be local or remote.

In the following lines, the most significant aspects of fast recovery strategies developed for these three types of devices are described.

4.1 Traditional devices

Traditional devices usually have two tables: the Routing Information Based (RIB) table and the FIB table. The RIB is a control plane related table —or repository— where all the routing entries are stored, including primary and alternate paths for each destination. On the contrary, the FIB is the table used to forward packets, so it stores essential information to do that, like the best active path.

When a link fails, the hosting device's hardware/OS is designed to notify routing protocols about that event, so the process that manages the interfaces will communicate with

the process that runs the routing protocol to inform about the link-down event. Regardless of how the link-down event is detected —LOS or *Hello*-based—, if there is not any previous alternate route, the routing protocol's notification and convergence process searches for an alternate route. Then, the RIB is modified in the control plane to mark the primary route as failed, and the FIB is updated with the new route. The updating process is relatively fast as control and data planes are collocated in the same physical device. However, in order to converge to a new routing state, the information about the failure needs to be propagated along the network, which is the main bottleneck of the distributed routing strategies.

The failure detection and recovery processes in traditional devices are defined by the routing protocol, being OSPF the most extended one. The time that OSFP takes to restore the network is in the order of seconds and depends on many factors, such as the network topology, the network congestion or how many nodes detect the failure [32]. There are OSPF parameters, such as the router dead interval, that can be customized to improve this time. However, the process does not meet the sub-50 ms recovery time required for carrier-grade networks [3] and is prone to create loops due to inconsistencies during the propagation of the information about the link states along the distributed network.

Regarding fast recovery, the aforementioned BFD [29] is a *Hello*-based link-down detection mechanism standardized by the IETF, which reduces the time of detecting a link failure compared to other LSA-based strategies. On the other hand, IPFRR [5] is a pure IP-based mechanism that leverages precomputed routes to fasten the process of switching to an alternate, avoiding the routing protocol's convergence stage and thus reducing the recovery time. When a failure shows up, the affected routers switch to an alternate path instantly, letting the Interior Gateway Protocol (IGP) converge in the background. However, it does not guarantee the full failure coverage —80% of single link failures and 40–50% of node failures [1]—, and it leads to transient loops when affected nodes do not switch to the alternate route at the same time.

4.2 OpenFlow-based devices

OpenFlow-based network devices rely on a centralized controller to compute and install forwarding paths. In this case, the standard way to proceed when a failure occurs is to first detect the failure using either LOS-based or *Hello*-based —e.g., LLDP messages— mechanisms and then notify the centralized controller about the failure. However, OpenFlow devices have the *FAST-FAILOVER* group, a type of OpenFlow group that is designed specifically to quickly detect and overcome link failures. Like the *SELECT* and *ALL* groups, the *FAST-FAILOVER* group has a list of buckets —being a bucket a list of actions. In addition to this list of actions, each bucket has a watch port and/or watch group as a special parameter. The watch port/group will monitor the up or down status of the indicated port/group. The procedure to determine if a port is up or down can be LOS detection or it can be based on BFD, depending on the implementation. If the port/group is deemed to be down, then the bucket will not be used. If the port/group is determined to be up, then the bucket can be used. Only one bucket can be used at a time, and the bucket in use will not be changed unless the status of the currently used bucket's watch port/group transitions from up to down. When such an event occurs, the *FAST-FAILOVER* group will quickly select the next bucket in the bucket list with a watch port/group that is up.

There is no guarantee on the transition time that takes to select a new bucket when a failure occurs, it depends on the search time to find a suitable port/group that is up on the specific switch. However, the motivation behind using a *FAST-FAILOVER* group is that it is almost guaranteed to be quicker than reaching the control plane to handle the port-down event and inserting a new flow or set of flows [19]. Therefore, with *FAST-FAILOVER* groups, link failure detection and recovery take place without the intervention of the centralized control plane.

4.3 Devices with PDP

PDPs allow the implementation of custom data plane algorithms that process packets at line rate. This ability can be leveraged to program tailored fast recovery algorithms that automatically reconfigure the network in case of failure, without the intervention of the control plane. In this case, the failure recovery process is not architecture-specific but implementation specific, which means that every device has its own mechanism to recover from link failures. Once the data plane is notified about the event, it triggers the recovery algorithm to find the most suitable alternate route —preinstalled or not— for the specific packet or flow.

Although the recovery process is implementation-specific, devices with PDP implement custom mechanisms to detect and react to link failures. For instance, generic devices with PDP, such as bmv2 software switches, leverage *Hello*-based mechanisms to detect and react to link failures via packet loss. On the other hand, some devices implement hardware-based mechanisms to detect and notify the data plane when a link goes down, as it is the case with Tofino switches.

4.3.1 Fast reaction in generic devices with PDP

Among the generic devices with PDP, bmv2 [15] is a reference P4 software switch written in C++11 that is meant to be used as a tool for developing, testing and debugging P4 data planes. It takes as input a file generated from a P4 program by a P4 compiler and interprets it to implement the packet processing behavior. The bmv2 software switches do

not implement any mechanism to notify the data plane when a link goes down, they rely on self-implemented *Hello*-based mechanisms to detect link failures such as in [16]. One or both ends of a link send *Hello* packets to determine if the link is up or down. When an end detects that the arrival of such packets stops, the switch sets the affected port as down —see Fig. 2. In bmv2, *Hello* packets are generated as a response to other incoming packets, either from the control plane or directly cloning user space packets. It is worth noting that data plane *Hello*-based detection requires additional state information —e.g., registers— to store the status of each port [33]. This solution allows the data plane to know if a port is up or down, however, its cost in terms of latency is high, as the switch needs to specify the period and set the threshold of lost packets until the port is marked as down. Note that other P4-programmable hardware devices, such as the Netronome NFP-4000, also rely on implementation-specific Hello-based mechanisms to fast react to link failures.

4.3.2 Fast reaction in devices with specific hardware

Devices with specific hardware, such as Tofino switches, internally maintain the state of their ports so when a port goes down they automatically discard the packets destined to it. The signal comes from the network interface card and represents the state of the link. The same signal can be used to trigger the *Port Down* mechanism, which leverages a packet generator —specific hardware— that generates packets that enter the pipeline of the specific switch in which the port goes down —see Fig. 2. In this way, the link-down event is detected very quickly and the reaction can be triggered directly from the data plane at almost zero delay. Section 5 elaborates more on the failure reaction time of these types of devices.

5 Testing failure reaction time

This section evaluates the Failure Reaction Time (FRT) of a hardware switch with PDP that uses a Tofino ASIC. As previously explained, Tofino-based switches implement a failure recovery mechanism called *Port Down*, which leverages LOS detection to generate a packet that enters the ingress pipeline of the switch whenever a link goes down.

As previously explained, when a link failure occurs, first, the switch must be aware of that failure, and then, the error must be notified to the corresponding entity —control or data plane— in order to apply the backup configuration if any. Therefore, we distinguish between the time for detecting the failure and the time that takes to react to that failure. After the failure reaction, there is a recovery time, which is the period that takes the switch to apply the necessary changes in order to face the link failure and restore the alternate routing.

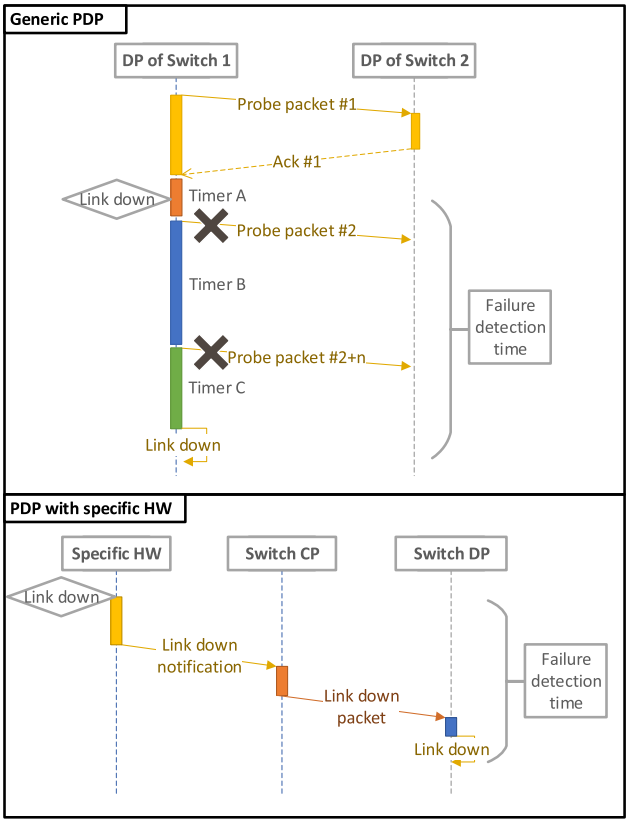


Fig. 2 Mechanisms to inform the data plane about a failure in devices with PDP

This section focuses on evaluating the FRT, the specific implementation of the failure recovery algorithm is out of the scope of this paper.

5.1 Measurement method

We hereby describe the method designed to measure the FRT of an Intel Tofino switch, the control and data plane code is fully available at [34] and the post that discusses the method is in [35]. Figure 3 shows the experimental scenario deployed to measure the FRT using real hardware equipment. We use two hosts, $h1$ and $h2$, and one Intel Tofino switch to connect the hosts through SFP+ 10 Gbps interfaces, and we configure Tofino's Packet Generation Tool in the port 17 of the switch to generate a port-down packet whenever the link goes down. This packet is delivered to the Collector, connected through port 1 at 1 Gbps, which receives the timestamps and computes the FRT. For the hosts, we use two Anritsu MT1000A transport modules, while the Collector is built on a commercial Supermicro server with an octa-core Intel(R) Xeon(R) D-2146NT CPU running Ubuntu 20.04. The first host $h1$ is in charge of sending packets to the second host $h2$. Every time a data packet arrives at the ingress pipeline of the switch a timestamp ts_1 is stored in a regis-

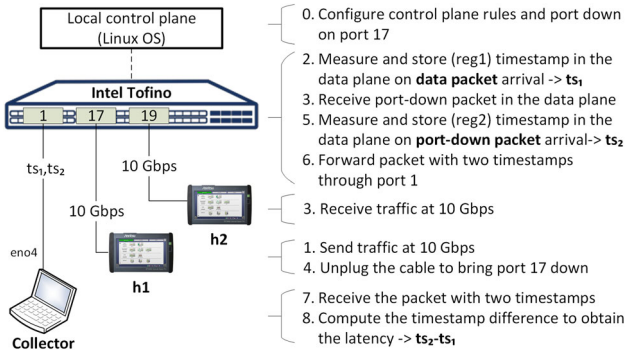


Fig. 3 Testing scenario to experimentally measure Tofino's failure reaction time

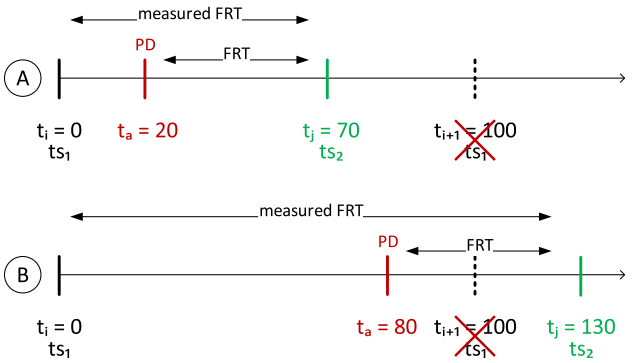


Fig. 4 Potential occurrences, **A** and **B**, when measuring the FRT

ter. Then, we physically disconnect the link between $h1$ and the switch —port 17— so a port-down packet is generated and sent to the data plane of the switch. When the port-down packet arrives at the ingress pipeline of the data plane, a new timestamp ts_2 and the previously stored ts_1 are attached to the packet. The resulting packet is finally forwarded to port 1, so it reaches the Collector where it is processed to compute the measured FRT as the difference between ts_2 and ts_1. Both timestamps are measured in the switch, using the Intel Tofino's timestamping engine, which provides a 48-bit counter that increments every nanosecond, thus getting a nanosecond-level precision. The timestamp used is the Tofino's $ig\_intr\_md.ingress\_mac\_tstamp$, which is taken on the arrival of the packet at ingress.

The following examples in Fig. 4 depict, from a theoretical perspective, how the FRT is measured using the method described in the aforementioned scenario. Let us assume that $h1$ is sending data packets at a constant rate, which arrive at the switch at $t_i, t_{i+1}, ..., t_{i+n}$. Let us also assume that, in the absence of any other external event, the FRT is constant for every port-down event. Then, Fig. 4 shows two potential occurrences, A and B, when a port-down event —shown as PD— occurs in the Tofino switch at the time t_a.

In occurrence A, the port-down event takes place closer to the first data packet arrival denoted as t_i. As explained before, the timestamp ts_1 is only stored when a data packet arrives at $t_i, t_{i+1}, ..., t_{i+n}$, therefore, the port-down packet that arrives at t_j makes ts_2 be stored before the scheduled arrival of the next data packet —which should have arrived at t_{i+1} if the port-down had not occurred. In this case, the difference between the real FRT and the measured FRT, named measurement error (ϵ), is lower. By contrast, in occurrence B, the port-down event happens just before the scheduled arrival of the next data packet at t_{i+1} —which in fact does not arrive due to the port-down—, so the ϵ is greater because the packet that should arrive at t_{i+1} is lost and thus the measured FRT is computed based on previous timestamp stored at t_i. The maximum and minimum values of the measured FRT can be theoretically computed as follows. We know that FRT is computed as:

$$FRT = t_j - t_a \tag{1}$$

where t_a is the time at which the port-down event occurs. We can also compute the measured FRT as:

$$FRT_{meas} = t_j - t_i \tag{2}$$

Finally, the error ϵ is calculated as the difference between the real and measured FRTs as follows:

$$\epsilon = FRT_{meas} - FRT = t_a - t_i \tag{3}$$

We derive from (3) that the minimum and maximum values of ϵ are obtained when $t_a \approx t_i$ and $t_a \approx t_{i+1}$ respectively, thus when the port-down event occurs just after and before the arrival of a new data packet.

According to the aforementioned occurrences, we also conclude that the packet inter-arrival time (Δt), which is the time elapsed between the arrival of two consecutive packets t_i and t_{i+1}, is relevant to reduce the ϵ. In fact, reducing Δt means increasing the sampling frequency, so it is desirable to be of the same or less order of magnitude as the FRT to get the most accurate measures. In consequence, we should generate packets with the lowest Δt. The packet inter-arrival time is theoretically computed as $\Delta t = {}^{pkt\_length}/_{bitrate}$, where pkt_lenght is in bits. In this test, we achieve a maximum bitrate of 7.619 Gbps with a 64-byte frame size, which results in an inter-arrival time $\Delta t = 67.2\ ns$.

The data plane implementation of the *Port Down* can influence the FRT, so we should pay attention to the elements in the pipeline that introduce latency when a packet is processed [36, 37]. First, in the parser block, represented as a state machine, the packet moves from one state to another depending on the packet header. As all the port-down packets have the same header, we assume this latency is constant —packets always traverse the same states. Then, in the control block, the processing logic can add a variable latency that depends

on the data plane complexity. Addressing a link-down event is considered a high-priority task, so it is recommended to process the port-down header at the beginning of the control block to avoid additional latency. The Listing 1 shows an example of this implementation that leverages an if/else statement to check if the packet is of type *Port Down*. If so, the actions corresponding to the link-down event are run, i.e., we update a register that stores the status of each port. If not, the rest of the control is executed. Note that, in terms of latency, the deparser block is considered equivalent to the parser. In this way, the additional latency introduced is constant and limited to a match-action table corresponding to the if/else statement.

```
const bit<32> REG_IDX   = 0x1;
register<psv_t>(16) portStatus;
apply{
  portStatusReg.read(meta.port_status, REG_IDX);
  if (hdr.portdown.isValid()) {
    bit<16> rcvPortStatus = 16w1 <<
      (hdr.portdown.port - 1);
    meta.port_status = meta.port_status |
      rcvPortStatus;
    portStatusReg.write(REG_IDX, meta.port_status);
    drop();
  } else {
    // Rest of the control logic
  }
}
```

Listing 1 Implementation of the port-down event in the control block.

Finally, it is worth noting that all the experiments are carried out without overloading the switch, which results in a much lower reaction time compared to state-of-the-art recovery time values reviewed in the literature. We estimate that under overloading conditions, although the reaction time can rise compared to not overloading the switch, it will be even more favorable if we compare it to other existing recovery mechanism under the same condition.

5.2 Results

Figure 5 plots the experimental results obtained from measuring the FRT according to the scenario defined in this experiment. Looking at the numerical results, FRT_{meas} shows an average of 472.88 µs and a standard deviation of 17.28 µs. There are several measured values under 450 µs, with 393.3 µs being the minimum value. The standard deviation is one order of magnitude less that the average and the 97% of the measured reaction times are within plus or minus the standard deviation, which ensures the statistical validity of the overall results.

These results show that the Intel Tofino switch with PDP achieves a FRT that is in the order of microseconds, which opens the door to the implementation of faster and more robust data plane based failure recovery strategies. This happens thanks to the ability of the devices with PDP to define custom packet processing pipelines, and their hereby

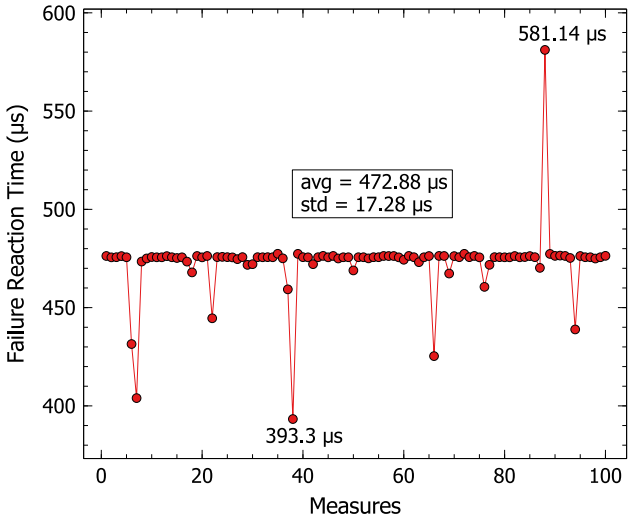


Fig. 5 Results of the measured FRT when using 7.619 Gbps bitrate and 64-byte frame size

demonstrated capacity to quickly react to link failures. These features allow for the implementation of failure recovery strategies that redirect every single packet through the alternate path, thus getting closer to the zero-packet loss target. Moreover, choosing the correct failure recovery strategy can lead us to avoid transitory routing states due to sub-optimal alternate routes such as tunnels. In fact, the alternate route can be as optimal as the primary one, so it could persist even when the link brings up again after the failure.

On another hand, devices with PDP provide the ability to combine decentralized and data plane based failure recovery strategies with the centralized view of the SDN controller. The former are required to optimize the fast reaction of the network while, as a second step, the latter provides an updated and more accurate view corresponding to the new state of the network after the fast recovery process.

Apart from the benefits in terms of improving the fast recovery strategies, we foresee other fields in which such a negligible reaction time could be beneficial. For instance, in time-sensitive networks (TSN) an extremely low latency —and low jitter— is required to synchronize all the entities involved in the communication. In this sense, recovering from link failures in a microsecond scale provides the required resiliency to both execute low-latency synchronization protocols and run ultra-reliable services over the TSN. In addition, real-time monitoring systems can benefit form the ability to detect link failures with a very low delay. Devices with PDP can be combined with complex NFV orchestrators in order to notify their monitoring systems whenever a link-down event occurs, activating alternate paths to reach the compute nodes or even deploying backup functions to restore the service.

6 Conclusions

In this paper, we study the different link failure recovery strategies implemented in current communication networks, due to their utmost importance in providing proper support for many services that demand stringent delay and packet loss requirements. We classify those communication networks according to the network devices that comprise them, including traditional distributed control plane based devices, SDN/OpenFlow based devices and devices with PDP. Our focus is on analyzing the fast recovery strategies that are implemented directly in the data plane, as their response, in terms of packet loss and delay, is more optimal than distributed or control plane based strategies.

In addition, we address the link failure detection and recovery times from a theoretical point of view, while introducing the new concept of failure reaction time, which refers to the time that takes a node to trigger the failure recovery process. We consider the analysis of the reaction time relevant to characterize the failure recovery process and to determine the network's ability to react fast to link failure events. In this sense, we provide a method to measure the FRT of Intel Tofino switches with PDP, which relies on the packet-based *Port Down* mechanism to inform the data plane about link failure events. Our promising results show that the FRT is in the order of microseconds, which combined with the ability of PDPs to define custom packet processing pipelines, enables the implementation of novel fast failover strategies that allow us to recover from link failures at line rate and thus optimize the overall network recovery time. Therefore, we conclude that such devices with PDP can improve the network's performance —reduce packet losses and delays— by implementing a failure recovery algorithm that leverages their fast reaction capability, which is indeed part of our future work.

Acknowledgements This research was supported by the Spanish Ministry of Science and Innovation (Grant number PID2019-108713RB-C54) under the project "Towards zero touch network and services for beyond 5 G (TRUE5G)".

Author Contributions DF—wrote the main manuscript. MH—contributed to the design of the testing method (Sect. 5). All authors reviewed the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Ministry of Science and Innovation (Grant number PID2019-108713RB-C54).

Declarations

Conflict of interest The authors declare no competing interests.

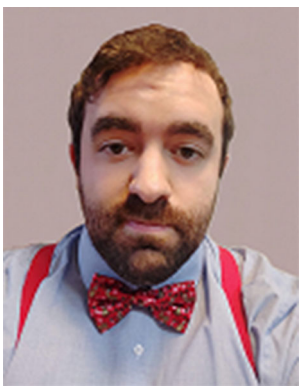
References

1. Chiesa, M., Kamisiński, A., Rak, J., Retvari, G., & Schmid, S. (2021). A survey of fast-recovery mechanisms in packet-switched networks. *IEEE Communications Surveys & Tutorials, 23*(2), 1253–1301. https://doi.org/10.1109/COMST.2021.3063980
2. Shukla, A., & Foerster, K. T. (2021). Shortcutting fast failover routes in the data plane. In Proceedings of the symposium on architectures for networking and communications systems (pp. 15–22). https://doi.org/10.1145/3493425.3502751
3. Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2013). OpenFlow: Meeting carrier-grade recovery requirements. *Computer Communications, 36*(6), 656–665. https://doi.org/10.1016/j.comcom.2012.09.011
4. Liu, J., Panda, A., Singla, A., Godfrey, B., Schapira, M., & Shenker, S. (2013). Ensuring connectivity via data plane mechanisms. In 10th USENIX symposium on networked systems design and implementation (NSDI 13) (USENIX Association) (pp. 113–126). https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_junda
5. Atlas, A. K., Zinin, A., Torvi, R., Choudhury, G., Martin, C., Imhoff, B., & Fedyk, D. (2008). Basic specification for IP fast reroute: Loop-free alternates. RFC 5286, RFC Editor. https://www.rfc-editor.org/rfc/rfc5286
6. Swallow, G., Pan, P., & Atlas, A. (2005). Fast reroute extensions to RSVP-TE for LSP tunnels. RFC 4090, RFC Editor. https://www.rfc-editor.org/rfc/rfc4090
7. ONF. (2012). OpenFlow switch specification version 1.3.0. OpenFlow Spec, Open Networking Foundation. http://opennetworking.wpengine.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf
8. Cisco. (2014). BGP PIC edge for IP and MPLS-VPN. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-3s/irg-xe-3s-book/irg-bgp-mp-pic.html. Accessed 07 Mar 2023.
9. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., & Walker, D. (2014). P4: Programming Protocol-independent Packet Processors. *SIGCOMM Computer Communication Review, 44*(3), 87–95. https://doi.org/10.1145/2656877.2656890
10. Sharma, S., Colle, D., & Pickavet, M. (2020). Enabling fast failure recovery in openflow networks using routeflow. In 2020 IEEE international symposium on local and metropolitan area networks (LANMAN (IEEE) (pp. 1–6).
11. CPqD. (2023). RouteFlow: Virtual IP routing services over OpenFlow networks. https://routeflow.github.io/RouteFlow/. Accessed 27 Sept 2023).
12. Kamamura, S., Shimazaki, D., Hiramatsu, A., & Nakazato, H. (2013). Autonomous IP fast rerouting with compressed backup flow entries using OpenFlow. *IEICE Transactions on Information and Systems, 96*(2), 184–192. https://doi.org/10.1587/TRANSINF.E96.D.184

13. Waqas, M., Malik, S. U. R., Akbar, S., Anjum, A., & Ahmad, N. (2019). Convergence time analysis of OSPF routing protocol using social network metrics. *Future Generation Computer Systems, 94*, 62–71. https://doi.org/10.1016/J.FUTURE.2018.11.003
14. Mininet Team. (2023). Mininet main page. https://mininet.org/. Accessed 20 April 2023.
15. P4 community. Behavioral Model version 2 (bmv2). The reference P4 software switch. https://github.com/p4lang/behavioral-model. Accessed 01 May 2023.
16. Miura, H., Hirata, K., & Tachibana, T. (2022). P4-based design of fast failure recovery for software-defined networks. *Computer Networks, 216*, 109274. https://doi.org/10.1016/j.comnet.2022.109274
17. Kvalbein, A., Hansen, A. F., Cicic, T., Gjessing, S., & Lysne, O. (2008). Multiple routing configurations for fast IP network recovery. *IEEE/ACM Transactions on Networking, 17*(2), 473–486. https://doi.org/10.1109/TNET.2008.926507
18. Xu, J., Xie, S., & Zhao, J. (2021). P4Neighbor: Efficient link failure recovery with programmable switches. *IEEE Transactions on Network and Service Management, 18*(1), 388–401. https://doi.org/10.1109/TNSM.2021.3050478
19. Lin, Y. D., Teng, H. Y., Hsu, C. R., Liao, C. C., & Lai, Y. C. (2016). Fast failover and switchover for link failures and congestion in software defined networks. In 2016 IEEE international conference on communications (ICC) (IEEE) (pp. 1–6). https://doi.org/10.1109/ICC.2016.7510886
20. Chiesa, M., Sedar, R., Antichi, G., Borokhovich, M., Kamisiński, A., Nikolaidis, G., & Schmid, S. (2019). PURR: A primitive for reconfigurable fast reroute: hope for the best and program for the worst. In Proceedings of the 15th international conference on emerging networking experiments and technologies (pp. 1–14). https://doi.org/10.1145/3359989.3365410
21. Lee, S. S., Li, K. Y., Chan, K. Y., Lai, G. H., & Chung, Y. C. (2014). Path layout planning and software based fast failure detection in survivable OpenFlow networks. In 2014 10th international conference on the design of reliable communication networks (DRCN) (IEEE) (pp. 1–8). https://doi.org/10.1109/DRCN.2014.6816141
22. Fernandes, E. L., Rojas, E., Alvarez-Horcajo, J., Kis, Z. L., Sanvito, D., Bonelli, N., Cascone, C., & Rothenberg, C. E. (2020). The road to BOFUSS: The basic OpenFlow userspace software switch. *Journal of Network and Computer Applications, 165*, 102685. https://doi.org/10.1016/j.jnca.2020.102685
23. Van Adrichem, N. L., Van Asten, B. J., & Kuipers, F. A. (2014) Fast recovery in software-defined networks. In 2014 3rd European workshop on software defined networks (IEEE) (pp. 61–66). https://doi.org/10.1109/EWSDN.2014.13
24. Turull, D., Sjödin, P., & Olsson, R. (2016). Pktgen: Measuring performance on high speed networks. *Computer Communications, 82*, 39–48. https://doi.org/10.1016/j.comcom.2016.03.003
25. Goyal, M., Soperi, M., Baccelli, E., Choudhury, G., Shaikh, A., Hosseini, H., & Trivedi, K. (2011). Improving convergence speed and scalability in OSPF: A survey. *IEEE Communications Surveys & Tutorials, 14*(2), 443–463. https://doi.org/10.1109/SURV.2011.011411.00065
26. Institute of Electrical and Electronics Engineers (IEEE). (2006). Standard for information technology–telecommunications and information exchange between systems–LAN/MAN–specific requirements part 3: CSMA/CD access method and physical layer specifications–amendment: Physical layer and management parameters for 10 Gb/s operation, type 10GBASE-T. IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005) (pp. 1–181). https://doi.org/10.1109/IEEESTD.2006.231802
27. SFF Committee. (2013). SFP+ 10 Gb/s and low speed electrical interface. SFF-8431 Rev 4.1 (Addendum) (pp. 1–137). http://www.snia.org/sff/specifications
28. Vasseur, J. P., Pickavet, M., & Demeester, P. (2004). *Network recovery: Protection and restoration of optical, SONET-SDH, IP, and MPLS*. Elsevier.
29. Katz, D., & Ward, D. (2010). Bidirectional forwarding detection (BFD). RFC 5880, RFC Editor. https://www.rfc-editor.org/rfc/rfc5880.html
30. Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P. (2013). Fast failure recovery for in-band OpenFlow networks. In 2013 9th international conference on the design of reliable communication networks (DRCN) (IEEE) (pp. 52–59). https://ieeexplore.ieee.org/document/6529882
31. Cascone, C., Sanvito, D., Pollini, L., Capone, A., & Sanso, B. (2017). Fast failure detection and recovery in SDN with stateful data plane. *International Journal of Network Management, 27*(2), e1957.
32. Siddiqi, A., & Nandy, B. (2005). Improving network convergence time and network stability of an OSPF-routed IP network. In NETWORKING 2005. Networking technologies, services, and protocols; performance of computer and communication networks; mobile and wireless communications systems: 4th international IFIP-TC6 networking conference, Waterloo, Canada, May 2-6, 2005. Proceedings (vol. 4, pp. 469–485). Springer. https://doi.org/10.1007/11422778_38
33. Sedar, R., Borokhovich, M., Chiesa, M., Antichi, G., & Schmid, S. (2018). Supporting emerging applications with low-latency failover in P4. In Proceedings of the 2018 workshop on networking for emerging applications and technologies (pp. 52–57). https://doi.org/10.1145/3229574.3229580
34. Franco, D. (2023). daviddvs/tofino-frt: Ready to publish v2. https://doi.org/10.5281/zenodo.7890921
35. Intel. (2022). Intel connectivity research program forum–port down reaction time. https://community.intel.com/t5/Intel-Connectivity-Research/Port-down-reaction-time/m-p/1432179. Accessed 03 May 2023.
36. Harkous, H., Jarschel, M., He, M., Priest, R., & Kellerer, W. (2019). Towards understanding the performance of p4 programmable hardware. In 2019 ACM/IEEE symposium on architectures for networking and communications systems (ANCS) (pp. 1–6). https://doi.org/10.1109/ANCS.2019.8901881
37. Harkous, H., Jarschel, M., He, M., Pries, R., & Kellerer, W. (2020). P8: P4 with predictable packet processing performance. *IEEE Transactions on Network and Service Management, 18*(3), 2846–2859. https://doi.org/10.1109/TNSM.2020.3030102

David Franco received his MSc degree in Telecommunication Engineering from the University of the Basque Country (UPV/EHU) in 2018. He joined the Communications Engineering Department of the UPV/EHU as a researcher in the I2T (Engineering and Research on Telematics) research lab in 2016. His research is focused on software-defined networking and network function virtualization applied to traffic engineering and secure communication systems. He is also a PhD student in Mobile Network Information and Communication Technologies at UPV/EHU.

Marivi Higuero obtained her BS and MS degrees in Telecommunication Engineering, in the Faculty of Engineering in Bilbao, in the University of the Basque Country (UPV/EHU), and the PhD degree in Engineering in Information and Communications Technologies, in the same University, in 2005. She worked in Sarenet (Bilbao), an Internet Service Provider, as a member of the technical department in this company. Since 1997 she works as assistant professor in the Communications Engineering Department in the University of the Basque Country. She is also member of the I2T (Engineering and Research on Telematics) research lab in the same University, where she has participated in several national and European R&D projects. Her research interests include SDN and NFV, network security, and ITS (Intelligent Transport Systems) and network mobility protocols.



Eder Ollora Zaballa obtained his BSc at the University of the Basque Country (EHU) and the MSc at the Technical University of Denmark (DTU). He also recei ved his PhD degree in the Network Technologies and Service Platforms group at DTU. He has previously been involved in the European project NGPaaS, Bandwidth-on-Demand (GEANT) and X2Rail. His research interest focuses on control and data plane programming (ONOS, Openflow, P4, and P4Runtime), distributed control planes, slicing, telemetry and network security in SDN networks. As a Postdoc at the same university (DTU), he is currently involved with P4 programming at a later project with GEANT

Juanjo Unzilla received the BS and MS degrees in electrical engineering and the Ph.D. degree in communications engineering from the UPV/EHU, in 1990 and 1999, respectively. He was the Head of the Electronic and Telecommunications Department, from 2001 to 2004, and the Vice-Chancellor of UPV/EHU, from 2004 to 2013. He is currently a Professor with the Communications Engineering Department, UPV/EHU, where he teaches subjects related to telecommunication networks and services. He is a member of the I2T Research Laboratory, where he participates in several regional, national, and European Research and Development projects. His research interests include SDN and NFV in 5G networks, its applications to industrial communications, and cybersecurity in distributed systems. He is one of the co-founders of the spin-off Keynetic focused on cybersecurity services to SMEs, in 2017.

Eduardo Jacob (Senior Member, IEEE) eceived the BSc degree in industrial engineering and the MSc degree in industrial engineering, and industrial communications and electronics respectively in 1987 and 1991, and the PhD degree in communications engineering in 2001 from the University of the Basque Country (UPV/EHU). During two years he worked in a public telecommunications research and development enterprise (currently Tecnalia). He spent several years as the IT Director in the private sector. Since 1994 then he has been at full time with the Faculty of Engineering in Bilbao, UPV/EHU, where he was elected as Head of the Department of Communications Engineering from 2012 to 2016. He is currently Full Professor and leads the I2T (Engineering and Research on Telematics) Research Laboratory. He also has directed several PhD theses and managed several research projects at the local, national, and European levels. His research interests include applying software-defined networks to industrial communications, cybersecurity in distributed systems, software-defined wireless sensor networks, and in-network processing.