InFaRR: In-Network Fast ReRouting

Fábio L. Verdi and Gustavo V. Luz

***Abstract*—InFaRR (In-network Fast ReRouting) is an algorithm for fast rerouting in programmable data planes. Implemented in P4, InFaRR is free of additional headers and network state heartbeats. InFaRR has four essential features not jointly found in other recovery mechanisms: Loop prevention, Pushback, Recognition and Restoration and Return to the main route. Tests in standard Fat-Tree and AB Fat-Tree topologies with failures in different scenarios showed positive results when compared to state-of-the-art algorithms in the literature. In scenarios in which the other algorithms were able to recover, InFaRR presented fewer hops to bypass the failure when the *Pushback*, Loop Prevention and Recognition and Restoration mechanisms were used. In scenarios with multiple failures, InFaRR successfully rerouted where the others algorithms in some cases looped. The mechanism for returning to the main route allows to verify failures in remote links, making it possible to return to the main route without intervention from the control plane. Several evaluations were done comparing the results of InFaRR with the state-of-the art mechanisms, showing the capability of our fast rerouting algorithm in dealing with failures at the line-rate.**

***Index Terms*—Data center networks, dataplane programmability, fault management, software-defined networking, fast rerouting.**

I. Introduction

DATACENTERS increasingly centralize the processing of applications that require real-time responses, with a high degree of interactivity or that are sensitive to latency variation. Such applications are present today in telemedicine services, autonomous vehicles, AR/XR, gaming, among others. To meet these characteristics of current applications, many of them are available in the cloud as they have fault tolerance mechanisms since their conception. Examples of these mechanisms include the usage of redundant equipment and multiple paths as well as algorithms that allow the choice of the main routing plan, packet prioritization and recovery approaches.

Fast rerouting solutions are characterized by the local ability of the switch to circumvent failures without the help of control plane and without dependence on signaling from other external elements, such as announcements of new routes, *heartbeat*, *keep alives* or overflow timeout of a dynamic routing protocol session [1].

Rerouting solutions supported by centralized control plane in Software Defined Networking (SDN) have a slower recovery mechanism when compared to fast rerouting in the data plane [1]. The centralized control plane has a delay to start the recovery, as it is necessary to detect the failure in a remote equipment through a *pooling* process, or to receive the failure notification, which is contrary to the fast rerouting proposal [2]. However, the recovery mechanisms executed in the data plane provide fast rerouting, improving the agility and speed of recovery during periods of failure, as there is no need to consult the control plane or dependence on some type of external agent signaling [3].

In this context, InFaRR benefits from the programmable data plane and proposes a fast rerouting algorithm capable of bypassing up to three simultaneous failures in *Fat-Tree* networks. InFaRR was implemented in the P4 language (*Programming Protocol-Independent Packet Processors*) and compared with the main solutions currently found in the literature [4]. InFaRR has four essential features not jointly found in other mechanisms: Prevention of *Loops* in the network, *Pushback*, Recognition and Restoration Mechanism and Return to the Main Route [5], [6].

The *loop* control proposed in InFaRR acts before the traditional TTL (*Time-To-Live*), preventing packets from circulating on the network. The *Pushback* Mechanism, also found in [7], returns packets to the previous switch in case there are no alternative routes from the point of failure. The Recognition and Restoration enables the discovery of which routes are at fault, thus avoiding sending further traffic through them. Finally, Return to the Main Route is a mechanism added to InFaRR capable of detecting if the original route has recovered from the failure and then resume sending back the affected flows through its primary route.

InFaRR was evaluated using the datacenter Standard Fat-Tree and AB Fat-Tree network topologies with k=4 in an environment emulated in Mininet. Different failure scenarios were exercised by comparing the proposed algorithm with related works which were minimally adapted to work in P4 [8], [9]. The algorithms were compared based on the evaluation of the success of recovery in the face of failures, configuration method, recovery scope, recovery domain, packet loss occurrence during the recovery process, packet transmission time and number of hops on the network after recovery from the failure.

InFaRR was firstly presented in [10] showing its main features and preliminary results. In this paper, InFaRR was extended and evaluated not only in Fat-Tree topologies but also in AB Fat-Tree to support up to three failures simultaneously. We also defined the taxonomy, the Finite State Machine (FSM) and compared it with other related and similar works.

Manuscript received 18 November 2022; revised 27 March 2023 and 31 May 2023; accepted 3 June 2023. Date of publication 6 June 2023; date of current version 9 October 2023. This work was funded by the São Paulo Research Foundation (FAPESP) under grant number 2021/14297-1 and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001." The associate editor coordinating the review of this article and approving it for publication was M. Tornatore. *(Corresponding author: Fábio L. Verdi.)*

Fábio L. Verdi is with the Computer Science Department, Federal University of São Carlos (Sorocaba Campus), Sorocaba 18052780, Brazil (e-mail: verdi@ufscar.br).

Gustavo V. Luz is with Claro Telecom Operator, São Paulo, Brazil.

Digital Object Identifier 10.1109/TNSM.2023.3283459

TABLE I
CLASS OF SERVICES DEFINED BY ITU-T

Service	Type of application	IPTD	IPDV	IPLR	IPER
Class 0	Real-time, jitter-sensitive, highly interactive	100 ms	50 ms	1 x 10^{-3}	1 x 10^{-4}
Class 1	Real-time, jitter-sensitive, interactive	400 ms	50 ms	1 x 10^{-3}	1 x 10^{-4}
Class 2	Transactions Data, highly interactive	100 ms	unfixed	1 x 10^{-3}	1 x 10^{-4}
Class 3	Transactions Data, interactive	400 ms	unfixed	1 x 10^{-3}	1 x 10^{-4}
Class 4	Tolerating low loss	1000 ms	unfixed	1 x 10^{-3}	1 x 10^{-4}
Class 5	Typical applications of IP networks	unfixed	unfixed	unfixed	unfixed

Therefore, the main contributions of this work are: 1) design and implementation of a fast rerouting algorithm for programmable networks, with the ability to recover from up to three failures in Fat-Tree networks; 2) definition of a taxonomy for InFarr which may be useful for other similar works; 3) execution of a proof of concept in a virtual environment from the perspective of Standard Fat-Tree and AB Fat-Tree topologies for analysis, interpretation and comparison with the state-of-the-art; 4) availability of the *dataset* collected during the proof of concept for the purpose of study replication and future comparisons.

This paper is organized as follows: next section presents some fundamental concepts about resilience in computer networks. Section III is dedicated to the related works. Section IV details InFaRR, the taxonomy and the FSM of the algorithm. Section V depicts the evaluation and the comparison with the state-of-the-art. Section VI shortly summarizes the lessons learned and limitations of InFaRR. Finally, Section VII concludes the paper.

II. RESILIENCE IN COMPUTER NETWORKS

Networks that need high levels of availability must be planned in order to circumvent possible ruptures in any of their elements and keep specific level of Quality of Service (QoS). In programmable networks, there is the possibility of building recovery policies based on each type of flow or clients, guaranteeing greater flexibility for different levels of service present in the network.

The Telecommunication Standardization Sector (ITU-T) has defined parameters related to IP network performance that quantify the required levels of QoS for the different classes of existing services, which are: IP Packet Transfer Delay (IPTD), IP Delay Variation (IPDV), IP Packet Loss Ratio (IPLR) and IP packet Error Ratio (IPER) [11], [12]. The requirements for each type of traffic can be seen in Table I and will be a reference for analysis and interpretation of the results of the algorithms evaluated in this work.

The first four classes of services (0 to 3) described in Table I are sensitive to transmission times (IPTD) and require end-to-end transmission paths of less than half of a second (400 ms), as are related to applications that require real-time or interactive communication. Also, classes 0 and 1 require a latency variation (IPDV) of less than 50 ms [13] to work properly. Therefore, a recovery mechanism in order to meet these described requirements must provide a rerouting within the IPDV limit, and the new contingency path must not exceed the required IPTD limit.

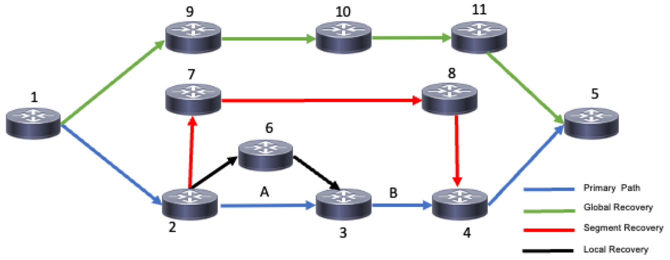

Fig. 1. Recovery scopes.

Dynamic routing protocols are among the tools used to provide fault tolerance and generate network reliability, so that when a link fails, the network can converge to another path. Traditional and widely used protocols, such as Border Gateway Protocol (BGP) or Open Shortest Part First (OSPF), can take tens of seconds to converge, while the requirements of real-time classes of services require this convergence to occur significantly in less time [14], therefore they do not meet the established QoS levels for computer networks.

Traditionally, in programmable networks, when the link fails, the switch needs to detect the failure and communicate the control plane to reconfigure the forwarding paths of the affected flows. This convergence process can take about 100ms if there is still connectivity between the remote switch and the control plane [15]. This work will present an alternative for rerouting in the data plane in order to meet the QoS requirements for classes of service which are constrained to IPTD, IPDV and packet loss.

1) Recovery Methods: Recovery methods can be classified by various criteria [16]. Four of these criteria are extremely relevant. The first is associated with the backup path configuration method, which can be pre-configured or reactive. The pre-configured method in SDN networks enables faster recovery, as the recovery strategy is already pre-defined to be used when necessary in the data plane, as a secondary routing table. In the reactive method, a recovery process is started in the control plane as soon as a failure is encountered. All routing table reconstruction processing is done by the control plane and distributed to the data plane.

The second criterion is associated with the scope of the recovery, which can be global, local or segmented. In the global policy, protection is offered to all elements of the path (end-to-end). Local policy ensures that there is a way to punctually bypass the faulty link. In this scenario, traffic will only be forwarded from the point where the fault was found. Finally, the segment recovery policy offers protection to specific segments involving switches and links, and will provide a punctual bypass to the problem segment [16].

In Figure 1, the main path between switch 1 and 5 is represented by the links in blue, passing through equipment 2, 3 and 4. The policy with global recovery scope, represented by the links in green, provides protection from failure of any

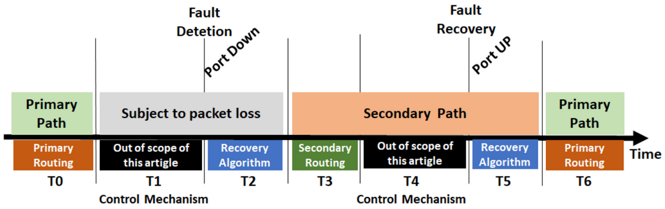

Fig. 2. Failure detection and recovery phases.

switches and links that compose the main path. The recovery scope per segment, represented by the links in red, together with switches 7 and 8, provide failover protection for switch 3 and links A and B. Finally, the local protection scope, provides protection only for link A through the black links passing through switch 6.

The third criterion refers to how the resources are used to provide recovery, which can be classified as dedicated or shared. In architectures that use dedicated resources, also known as the 1+1 approach, all elements that require a recovery method are duplicated. The shared approach, in turn, allows multiple elements to use the same backup resource, assuming (statistically), that there will not be more than one failed element requiring the use of this backup resource simultaneously.

The last criterion deals with the characteristics related to the operation and recovery domain. The operation domain attribute can be observed in SDN networks by the number of existing control planes. The more operation domains, the more complex the process of synchronism between the different control planes [17] will be. If we look at the Standard Fat-Tree and AB Fat-Tree networks, object of this study, we can see a PoD (Point of Delivery) as a recovery domain, depending if the rerouting process occurs exclusively in the same PoD or if there is a dependency from switches of other PoDs. InFaRR adopts a pre-configured recovery method that classifies it as a fast rerouting algorithm. The recovery scope adopted in InFaRR is per segment. The operation domain will be internal to *PoD*, bringing independence to the rerouting process, preventing packets from traveling through unnecessary equipment at the time of failure recovery.

2) Recovery Process: Initially, it is important to define the basic concept regarding the existing routing table types, in which the "main routing table" refers to the primary process used by the switch to define the output port for forwarding packets, using the destination IP address as key to lookup. The "secondary routing table", or backup, will only be used if the port suggested by the main routing table is in trouble.

Figure 2 describes the existing steps related to the failure detection and recovery process. In the initial step *T0*, the network is functioning with traffic sent through an optimal forwarding path, determined by the control plane. The failure detection, which happens in step *T1*, defines the moment when the port went to the *down* state, starting the recovery process in step *T2*. In step *T3*, the network operates through an alternative path until the control mechanism can re-establish connectivity, which happens in step *T4*. The trigger to change state to *up* from the previous step, starts the process of updating the routing tables (step *T5*), returning to the initial forwarding path in *T6*.

The control mechanism that occurs in steps *T1* and *T4* are not the scope of this work. In general terms, it means that our solution is independent of how the switch detects the failure of a given link. We assume that the hardware is capable of detecting a physical failure in the interfaces. However, changing the port state to *down* or *up* is the trigger to start the recovery algorithm. For evaluation purposes in this paper, the port state was arbitrarily imposed in the failure simulation and respective recovery process.

One of InFaRR's goals is to reduce the recovery time, which is the sum of the duration of steps *T1* and *T2*. In this range, from *T1* to *T2*, it is conjectured that packet loss may occur, so the shorter this time, the faster the convergence process [16]. Since in step *T1*, the link with status *DOWN* is detected, it is important to evaluate the duration of step *T2*, given that it may be directly influenced by the convergence algorithm used. Performing measurements related to the performance of the network during normal operation at the time *T0*, and comparing them with the operation at the time of contingency *T3*, will make it possible to evaluate the effectiveness of the contingency algorithm. It is conjectured that even in the process of convergence, performance measurements remain within the limits presented in Table I.

The network conditions at the moment of contingency will be equivalent or worse if compared to the normal operating state, assuming that the control plane offered the optimal route to normal operation in step *T0*. Therefore, the network conditions will be equivalent if the contingency structure has the same conditions of use and network latency, found in topologies with dedicated protection (1+1). In protection scenarios with shared resources, the contingent forwarding plan will be subject to the use of segments with greater occupancy, packet loss, greater latency or paths with greater number of hops that will directly impact the performance parameters of the contingent flow. For the purpose of this study, we will consider that the links that will be part of the redundancy process will be dedicated and will present the same network conditions (latency and utilization), so it will not be necessary to consider any process of prioritization or packet discard in this study.

III. Related Works

This section is divided into two parts: Section III-A presents the main works in the literature related to rerouting that inspired the creation of the InFaRR algorithm; and Section III-B will specifically deal with works related to rerouting proposals which are free of additional headers that we will use to compare with InFaRR in Section V.

A. Rerouting Solutions Inspiration for InFaRR

As already mentioned, dynamic routing protocols are among the tools used to provide fault tolerance and ensure network reliability. However, protocols such as BGP or OSPF can take tens of seconds to converge.

F10 [18] highlights the need for a physical and logical approach to implement a highly resilient environment. The discussion about points of failure and convergence process suggests an adaptation of the *Fat-Tree* topology, so that we move to a connection scheme between devices called *AB Fat-Tree*. The logical proposal of these authors contemplates a local protection scope and requires that the rerouting be executed in another PoD in a way that does not have an independent recovery domain. Despite its renowned scientific relevance, and serving as inspiration for this work, F10 proposal is conditioned to a reformulation of the connections between the equipment, decharacterizing the *standard Fat-Tree* topology as well as requires a process of announcements to all its neighbors in the event of failures. Finally, there is no P4 implementation for F10.

The P4-Protect algorithm [19], implemented in the P4 language, presents a 1+1 global recovery scope solution for IP networks. The algorithm requires a topology with two disjoint paths to the destination. The first switch in the network is responsible for forwarding a packet to each path (clone), adding a control header to the packet. The last switch removes the header and forwards the first received packet to the destination, discarding the cloned packet. Although this algorithm does not perform rerouting, it offers an interesting layer of resilience and presents excellent results in reducing *jitter*, but still requires studies in hierarchical datacenter topologies such as *Fat-Tree*.

Derived from Graph Theory, Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms enable fault recovery even without having a predefined secondary routing table [20]. When the packet cannot be forwarded to the destination port, the mechanism starts the search process for a viable path, systematically forwarding the packet according to the chosen algorithm. Implementing the DFS or BFS algorithms in P4 adds a header to the structure of packets, so that switches can track where the packet has gone. However, although there is no loop, the absence of a Recognition and Restoration Mechanism makes all packets go a long way to the destination [21].

RoLPS [22] proposes Loop-free alternates as a mechanism to detect loops in the network and presents a solution for FRR in the dataplane. The paper is very related to ours and has similar ideas, as for example, the point of local repair (PLR) concept defined in RoLPS is similar to our Pivot switch (explained in the next section). However, there are some key differences in which we believe make our approach simpler. The RoLPS needs to have a few bits in the header to implement what they call Advanced Loop Detection (ALD) so that packets are dropped when rerouted a third time. Our approach does not need any extra bits in the header. They also need to support tunneling for certain recovery scenarios. Our approach does not depend on tunneling to recovery, which may bring extra cost/overhead to the solution. Finally, InFaRR uses a Recognition and Restoration Mechanism that enables the switch to learn that a given path is no longer useful for the flow and proceeds to the rerouting plan or Pushback.

The comprehensive *survey* on fast data plane rerouting mechanisms by [14] presents the most current study on the different approaches to implementing these mechanisms. A chapter dedicated to SDN is included, in which strategies in Openflow environments and in programmable data planes (P4) are discussed. The survey highlights the implementation of the algorithm *Data-Driven Connectivity* (DDC) in P4 that uses additional headers to store the number of failures along the path taken by the packet, introducing the concept of link - reversal [7]. Such an approach generates a signaling *overhead* in the packet.

The P4Neighbor [23] and P4Resilience [24] are implementations in the P4 language of fast rerouting algorithms and use additional headers in the packet structure to encapsulate backup path information; thus they optimize rerouting by the switch, since the rerouting proposal is contained in the packet header. The P4Neighbor algorithm, when unable to forward a packet over a faulty link, adds a recovery header to the packet and recirculates it within the same switch, so that the same packet is rerouted through a working port. The recovery mechanism is predefined, so the multipath backup is calculated by the control plane [23]. P4Resilience has the differential of guaranteeing the creation of loop-free paths; however, the multiple backup paths need to be stored in the data plane and have linear growth in relation to the number of streams [24].

PURR [8], implemented in P4, presents a solution with two queries to match-action tables: the first query performed to the main routing table, returns the output port through which the packet must be forwarded. The second query uses the output port and the status array of the switch ports as the search key, and provides a port with the status active as the output port. This proposal optimizes the size of the second table but makes it impossible to handle rerouting by flows or other type of segregation.

B. Rerouting Algorithms That Are Free of Additional Headers

The main works in the literature which do not require network status announcements and do not use additional headers[1] in the packet structure were selected as directly related works. Such works were adapted to the P4 language and used for comparison purposes with the InFaRR algorithm.

The first fast rerouting algorithm, called **Static**, relies on pre-configured routes and proposes a secondary routing table to provide a bypass for failures when the main routing table is not able to forward the packets. The second algorithm, called **Rotor**, implements a target search mechanism using the active ports of the switch sequentially.

We also selected a traditional approach to programmable networks running in the **control plane** to prove the efficiency and benefits of fast rerouting.

1) Static Rerouting: In the work entitled "*Scalability and Resiliency of Static Routing*" [25], an extensive study is presented on the feasibility of implementing fault tolerance through the use of static routing and its variations. The most basic approach of static routing as a recovery mechanism consists of providing a main routing table and a secondary routing

[1] Extra headers should always be avoided since they cause extra overhead and fragmentation.

table to be used in the event of failures, both previously configured by the control plane. This approach offers local recovery scope and, when implemented in the core switch, needs a convergence domain from another PoD to find an alternate path to its destination.

All routing and rerouting actions are performed exclusively in the data plane through queries to previously defined routing tables. As soon as the packet passes through the switch's *Ingress* phase, the main routing table is consulted. If there is a *match*, an output port is returned to forward the packet, otherwise the packet is dropped. If the outgoing port is set to status "OK,"[2] the packet is forwarded to *Egress*. If the port is in *loop status* or *down*, the next step will be to consult the secondary routing table, where a new output port will be defined or the *Drop* is executed if there is no predefined secondary route. Finally, a new *status* check of the outgoing port (IF 2) is performed. If it has the status "OK", the packet is forwarded to *Egress*, otherwise the packet is dropped.

2) Rotor Rerouting: The work presented in [9] proposes a programmable data plane implementation without the need to use additional headers. The Rotor algorithm performs a lookup in the main routing table and, once the outgoing port is in trouble, the packet will be forwarded to the next available active port. The recovery scope is local, and its implementation on the core devices require the packet to be routed through another PoD. The configuration method was previously defined, since there are no queries to the control plane. Although there is no preconfigured backup routing table, the algorithm takes responsibility for finding an alternate path to the destination [26].

The workflow of the Rotor algorithm starts with the verification of the type of packet that will be processed. The packets, when processed for the first time, are classified as normal and proceed to the traditional lookup of the main routing table and check the status of the output port. When "Ok", the packet is sent to *Egress*; otherwise the packet is recirculated inside the switch, passing through *Ingress* again; however, now the packet will be flagged as "recirculated" and will be forwarded to the next port ("Output port + 1"). Once again, the status of the output port is checked, repeating the process until an output port with status "Ok" is found.

3) Control Plane: In a centralized control plane architecture, it is possible to provide global recovery scope, reactive configuration mechanism, and independent operating domain. One of the tasks of the control plane is to constantly assess the network conditions and, whenever necessary, update the routing tables of switches. The evaluation process may involve recurrent queries (pooling) of the control plane to each of the network equipment, causing considerable use of the network for its own monitoring. The update cycle interval is decisive for the management bandwidth consumption and for the problem detection time, and can become a big challenge [2].

It is up to the Control Plane to make a query to a routing table to check the status of the output port. However, there is no pre-programmed action in the data plane for rerouting. The Control Plane needs to update the routing table with viable routes whenever necessary. The process of detecting the failure and updating the routing table is the big challenge of this algorithm, and while this update does not happen, the packets will be dropped.

TABLE II
SUMMARY OF THE ALGORITHMS

Algorithm	**Control Plane**	**Static**	**Rotor**	**InFaRR**
Fast Rerouting	No	Yes	Yes	Yes
Place of the Action	Control Plane	Data Plane	Data Plane	Data Plane
Setup Method	Reactive	Preplanned	Search Path	Preplanned
Recovery Procedure	Global	Local	Local	Segment
Domain Recovery	Single	Single	Multiple	Single
Recovery Path	Optimized	No	No	Optimized
Pooling process	Yes	No	No	No

Table II summarizes the main characteristics of the related works that will be part of the experimentation to compare with the InFaRR algorithm.

IV. In-Network Fast ReRouting - InFaRR

The fast rerouting functionality starts as soon as the switch detects that it is not possible to forward packets through the port indicated by the main routing table. InFaRR proposes a fast rerouting algorithm for programmable networks capable of bypassing up to three faults in Fat-Tree networks without using additional packet headers and without link state monitoring mechanisms (heartbeats or keep alive messages).

The support for up to three link failures is because the fat-tree topology has three layers, and only one failure per layer is supported: from the ToR switch to the aggregation switch, from the aggregation switch to the core router/switch, and from the core router/switch back to the aggregation switch. The first two layers are managed using the backup route. In the core, Pushback is used. In fact, InFaRR may support more than three link failures at the same time. The general rule would be to support up to $k/2-1$ backup routes per layer (up to $k/2-1$ link failures). Such approach will require $k/2-1$ forwarding tables in the switch which may consume too much memory. At the end of the day, supporting up to three failures simultaneously should be quite enough for the control plane to assume and deal with rerouting.

InFaRR was implemented in the P4 language and its basic recovery principle is to query a predefined secondary routing table to be used in failure situations. Advanced functionalities complement its operation, such as: 1) Pushback mechanism; 2) Prevention of loops in the network; 3) Recognition and restoration mechanism; and 4) Return to the main route. It is worth mentioning that although such functionalities are found in other similar solutions, they are supported individually, not being implemented in P4 as a joint mechanism. Putting all the features together as a single solution is exactly what InFaRR aims to do.

A. InFaRR Taxonomy

The taxonomy of the InFaRR algorithm is important to understand the terminology used to describe the functionality presented in the routing and rerouting process. Such taxonomy is somehow found in similar works, but several other terms

[2] "OK" here means that the port is physically UP and not in loop.

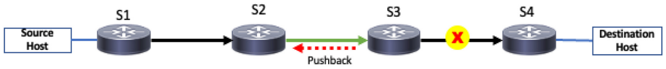

Fig. 3. Pushback sends the packet to the predecessor switch.

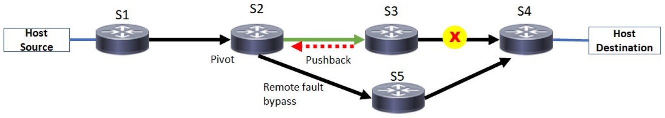

Fig. 4. Switch Pivot.

are newly defined in this paper and may be used to further data plane rerouting solutions.

- *Loop Prevention Mechanism:* It is the switch's ability to prevent packets from being sent on the same port they arrived. Although its detection is a simple activity (comparison if the input port is the same as the output one), it requires that the switch has mechanisms to handle this situation; InFaRR proposes rerouting or using the Pushback Mechanism for this purpose. One key aspect of InFaRR is that there are not backup routes in the core part of the network. This design choice was done since we do not want to send traffic to other PoDs, different from the destination PoD. If we allow having backup routes in the core, the other PoDs will end up serving as transit for traffic not being sent to them, causing a downstream/upstream movement of traffic inside the datacenter. In summary, in case of link failures in the core routers, InFaRR will always make Pushback towards the aggregation switches.
- *Pushback Mechanism:* it is ability of the InFaRR algorithm to forward packets to the predecessor switch in case of failures in the routing and rerouting process. This forwarding option prevents the packet from being dropped and forwards it to the predecessor switch which is an active and known path. Figure 3 shows the execution of the Pushback Mechanism on switch S3: unable to forward the packet to switch S4 and, in the absence of a secondary route, the packet is forwarded to its predecessor, S2.
- *Predecessor:* the predecessor concept differs from the previous switch concept as it is implemented during the Pushback Mechanism. Routing consists of a series of switches through which the packet needs to pass until it reaches its destination. The term predecessor applies to switch $n - 1$ considering n the current switch that is processing the packet. See an example with $n = 4$, as shown in Figure 3. The packet generated in the source host is forwarded through the network, passing sequentially through switches $S1 \rightarrow S2 \rightarrow S3$, until it finds a failure between the links S3-S4; switch S3 (n = 3) needs to perform the Pushback Mechanism as it does not have a secondary routing. Thus, the packet is forwarded to switch S2 ($n = 2$). Switch S2 has as its predecessor the switch S1 ($n - 1$), while switch S3 is the previous switch that the packet passed through.
- *Recognition and Restoration Mechanism:* the switch, when detecting a loop situation, learns that such a path is not useful for the flow being treated and then proceeds to the rerouting plan or Pushback. This feature enables the flow to use an optimized routing table since the packet does not need to go through unnecessary paths caused by the loop, and makes the switch a Pivot in the rerouting process to the flow.
- *Pivot:* every switch on the network that has a backup routing table can become a Pivot. The switch that has the port disabled for the flow by the Recognition and Restoration Mechanism, and has the rerouting port working is called a Pivot switch. This identification is important because it will determine the functioning of the Return to the Main Route mechanism. Figure 4 shows the operation of Pivot on switch S2. The packet follows the main routing through switches $S1 \rightarrow S2 \rightarrow S3$. Since the link between S3-S4 is failed, switch S3 does a Pushback returning the packet to S2. The Loop Prevention Mechanism causes switch S2 to forward packets through switch S5, while the Recognition and Restoration Mechanism causes the next packets in the flow to be forwarded along the fault-free route via S5. The Pivot switch makes it possible to bypass a remote fault and is where the Return to main route Mechanism will be implemented.
- *Return to the Main Route Mechanism:* this mechanism allows to return from the backup path to the main route once it is recovered. The idea behind the mechanism is essentially to duplicate (clone) a packet and send the original one to the backup path and the cloned one to the main route. If the packet sent through the main route does not return within a certain pre-defined time,[3] the switch assumes that the main route is recovered and should return sending of packets through it. Packet duplication ensures that one of them is delivered to the destination via the main or backup paths.
- *Primary routing table:* it is the first routing table to be used by the forwarding algorithm. The table is predefined by the control plane and may have different routing policies for each type of destination or flow.
- *Secondary routing table:* also called the *backup* routing table, it is used by the recovery mechanism when the primary routing table was not able to forward the packet. This table is also predefined by the control plane and only consists of entries for destinations or flows that require a recovery mechanism. The usage of a secondary (backup) routing table recalls the idea of traffic deflection [27] which is similar to our approach. However, some works use deflection to deal with persistent congestion and microbursts. In our case, we divert the flow to a preconfigured backup route because of a link failure which is a different use of deflection.

Figure 5 will be used as an example to visualize the terms and mechanisms that have been described. The topology[4]

[3] One TTL is enough to wait for the packet to get back.

[4] This topology was chosen just for sake of explanation. InFaRR is deeply evaluated in/for data center topologies.

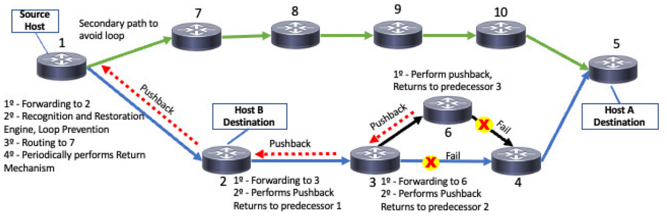
Source Host
1
Secondary path to avoid loop
7
8
9
10
5
1º - Forwarding to 2
2º - Recognition and Restoration Engine, Loop Prevention
3º - Routing to 7
4º - Periodically performs Return Mechanism
Pushback
Host B Destination
Pushback
2 1º - Forwarding to 3
2º - Performs Pushback Returns to predecessor 1
1º - Perform pushback, Returns to predecessor 3
Pushback
6
Fail
Fail
3 1º - Forwarding to 6
2º - Performs Pushback Returns to predecessor 2
4
Host A Destination

Fig. 5. Pushback, Loop Prevention, Recognition and Restoration mechanisms.

presents the communication between a source host and some destination hosts: hosts A and B.

The main routing table (highlighted in blue) between the source host and destination host A has two failures and we will use them to illustrate the Pushback Mechanism on switches 2, 3 and 6. We will also depict the Loop Prevention Mechanism on switches 1, 2 and 3, as well as the Recognition and Restoration Mechanism and the Return to the Main Route Mechanism on switch 1.

Switches 1 and 2 use the main routing table to forward packets to host A. Switch 3 detects the failure in link 3-4 and performs rerouting via switch 6. However, there is also a failure in the link 6-4 and, in this case, InFaRR triggers the Pushback Mechanism, returning the packet to its predecessor (switch 3). It is observed that switch 3 does not have viable forwarding options, as it has a DOWN port and another port in loop. In this case, the Pushback Mechanism forwards the packet to its predecessor (switch 2). Switch 2 also needs to use the Pushback Mechanism and forward the packet to its predecessor (switch 1).

Switch 1, upon receiving back the packet forwarded by switch 2, also detects the loop, thus triggering fast rerouting via switch 7. At this moment, switch 1 uses the Recognition and Restoration Mechanism to store the information that the path through switch 2 is unavailable, that is, the subsequent packets of this flow must be forwarded via switch 7. After that, switch 1 eventually starts the Return to the Main Route Mechanism to check the availability of forwarding packets via switch 2. In this case, the original packet (forwarded to switch 7) must be duplicated, having its copy sent to the output interface that connects it to switch 2. If the failures have not been corrected in switch 2, the duplicated packet will return to switch 1 according to the Pushback Mechanism already described. If the duplicate packet does not return, it is assumed that the faults have been recovered and this route becomes the main route for the routing process again.

There is a slight difference between the execution of InFaRR on switch 1, 2 and 3. On switch 2 and 3, the Recognition and Restoration Mechanism was not used, since the Pushback Mechanism sent the packet back to switch 1. Switch 1, on the other hand, benefited from the Recognition and Restoration Mechanism as the backup route is operational, becoming a Pivot in the rerouting process. Therefore, it is observed that the Recognition and Restoration Mechanism should only be used when the secondary route is operational, which does not occur with switch 2 and 3 for this scenario. InFaRR is able to detect this event by triggering the Recognition and Restoration Mechanism at different points in the network when necessary.

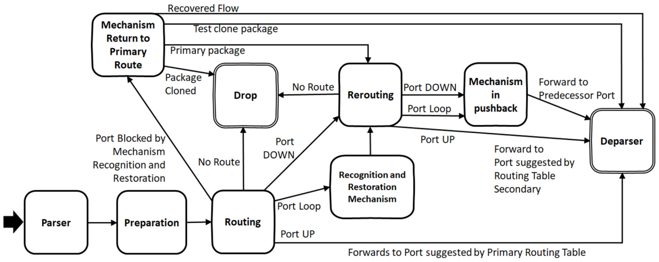
Recovered Flow
Mechanism Return to Primary Route
Test clone package
Primary package
Package Cloned
Drop
No Route
Rerouting
Port DOWN
Port Loop
Mechanism in pushback
Forward to Predecessor Port
Port UP
Deparser
Forward to Port suggested by Routing Table Secondary
Port Blocked by Mechanism Recognition and Restoration
No Route
Port DOWN
Recognition and Restoration Mechanism
Port Loop
Port UP
Parser
Preparation
Routing
Forwards to Port suggested by Primary Routing Table

Fig. 6. FSM for the InFaRR algorithm.

B. Finite State Machine - FSM

We have designed a simplified FSM for InFaRR which is shown in Figure 6. We will focus on the following states: Routing, Recognition and Restoration, Return to the main route, Rerouting and Pushback. Such states are the main ones necessary to showcase the features of InFaRR. We will not discuss the Parser and Deparser states since they are well-known for P4-based data plane programs. We will also not discuss the Preparation state since it basically consists in initializing the counters, registers and variables necessary to run InFaRR. Finally, the Drop state represents the state of a dropped packet.

1) Routing State: The Routing state is responsible for performing the basic packet switching function and is present in all programmable network switches. We evaluated the feasibility of selecting priority flows that would require different routing policies. We consider the source IP address, destination IP address and destination TCP ports as the key for the lookup, used individually or jointly. The implementation of the routing table with this lookup key structure proved to be viable and easy to implement. Flows that do not have routes defined in the routing table are transitioned to the *Drop* state.

There are five possible transitions to the next state, which are chosen according to the physical (UP/DOWN) or logical (loop or blocked) status of the output port:

- Output port with status UP: transition will take place to Deparser state;
- Output port with status DOWN: the transition will take place to the Rerouting state;
- Output port not specified: in cases where there was no match in the routing table, the transition will occur to the DROP state;
- Output port in loop: in cases where the output port is the same as the input port, the transition will take place to the state of Recognition and Restoration Mechanism;
- Outgoing port blocked by the Recognition and Restoration Mechanism: the transition will take place to the Return to the main route state.

2) Rerouting State: Once the packet is assigned to the rerouting state, a process similar to that of routing will be carried out, however with a query to a secondary routing table (backup). The secondary routing table is defined in advance and can be adjusted as needed by the control plane.

There are four possible transitions to the next state which are chosen according to the physical (UP/DOWN) or logical (loop) status of the output port:

- Output port with status UP: the transition will take place to the Deparser state;
- Output port with status DOWN: the transition will take place to the Pushback Mechanism state;
- Output port not specified: in cases where match does not occur in the secondary routing table, the transition will occur to the DROP state;
- Output port in loop: in cases where the output port is the same as the input port, the transition will take place to the Pushback state.

In the rerouting state, it is not necessary to manage the logical state of the port since once it is not possible to forward to the recommended port, the Pushback will occur and consequently the Recognition and Restoration Mechanism of the predecessor switch will prevent packets from this flow from reaching this switch again.

3) Recognition and Restoration State: The Recognition and Restoration state is transitioned when the Routing state detects a loop. In this step, InFaRR will record that the output port is inactive for this flow. Subsequent packets of this flow will not be forwarded through this port. The timestamp at which the status of the flow was changed is also stored for future use by the Return to the main route mechanism. After completing all the steps described, the FSM will be transitioned to the Return to the main route state (arrow not shown in the FSM for sake of simplicity).

4) Return to the Main Route State: The Return to the Main Route Mechanism can be divided into two functions: 1) periodically send verification packets. This action duplicates a data packet and sends it to the main route, in addition to sending the original packet via the backup path; and 2) validate the return of the cloned packet. If the cloned packet does not return within the pre-established period, for example one RTT as explained above, InFaRR assumes that the main route has been re-established. From this moment on, packets can be sent via the main route. If the packet returns, it is concluded that the failure still persists.

There are four possible transitions to the next state which are chosen according to the physical status (UP/DOWN) of the output port:

- The time to clone the packet and send it to the main route has not arrived: the transition will take place to the Rerouting state;
- The time to clone the packet and send it to the main route has arrived: the transition will take place to Deparser and Rerouting state simultaneously;
- The cloned packet does not return within one TTL: the main route is recovered and the transition will take place to the Deparser state;
- The cloned packet returns: the main route is still failed, then transition to Drop state.

5) Pushback State: The Pushback Mechanism is transitioned when the Rerouting state cannot forward packets because of the DOWN status or port in loop. Faced with this condition, the switch will send the packet to the predecessor switch through the mapped port in the Preparation state. After all the described steps are completed, the FSM will be transitioned to the Deparser state.

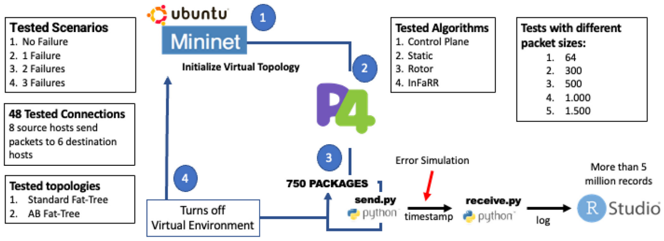

Fig. 7. Setup for the evaluation.

V. Evaluation

The proposed algorithm, as described in Section IV, was implemented using the P4 language. For the evaluation, a BMv2[5] Mininet environment was created. The source code of InFaRR can be found in the public repository.[6]

The experiment was performed using standard Fat-Tree and AB Fat-Tree topologies with k=4. In order to evaluate the network recovery mechanism in different failure scenarios between hosts of different PoDs, flows with different packet sizes were created to measure the impact of packet losses versus the packet size. Different sizes of payloads in the IP layer were evaluated[7]: 64, 300, 500, 1000 and 1500 bytes. Each execution was performed independently for InFaRR as well as each of the three algorithms mentioned in Section III-B. As described, the InFaRR algorithm has the ability to recover up to three failures simultaneously, so all algorithms were subjected to the following scenarios:

1) *Fail-free scenario:* to create a baseline and validate the connectivity between the network elements, a test was carried out in a environment without failures;
2) *Scenario with 1 failure*[8]*:* a failure was simulated in the main link at the destination PoD;
3) *Scenario with 2 failures:* a failure was simulated in the main link and another in the secondary link at the destination PoD;
4) *Scenario with 3 failures:* failures were simulated in three sequential links in the routing process (main, secondary and tertiary) at the destination PoD.

Figure 7 represents the experimentation scenario used to evaluate this work. Each test consists of starting the virtual environment that simulates the (AB) Fat-Tree network with k=4 in Mininet (step 1), and loading the respective P4 algorithm on the switches referring to the test to be evaluated (step 2). Once the environment is ready, the IPDT calculation tool is started on the source and destination hosts (step 3). The information for each packet is stored in a log file for further

[5] https://github.com/p4lang/behavioral-model.

[6] https://github.com/dcomp-leris/InFaRR.

[7] Refers to the size of the "IP.len" field.

[8] The failures always occur between the Core and Aggregation equipment of the destination *PoD*.

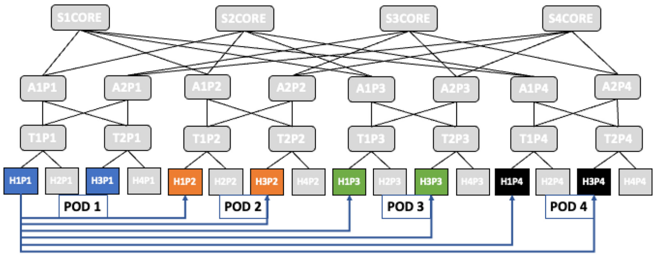

Fig. 8. Setup for the evaluation.

analysis. The test ends with the deactivation of the virtual environment and cleaning the variables (step 4). The data collected in all tests and scenarios were analyzed using *R Studio*.

The topology used is shown in Fig. 8 with k=4. As an example, the topology demonstrates that host H1P1 will send traffic to other six hosts, as shown in blue lines. Thus, following this same concept, all selected hosts are taken as source and will connect to hosts in other PoDs.

Tests with hosts of the same PoD were not considered since failures in such a scenario may be solved using intra-PoD solutions. For optimization of the experimentation, we considered only one host of each ToR switch, since all hosts connected to the same ToR have the same routing table. Thus, in a k=4 topology, two hosts were chosen from each PoD, each one connected to a different ToR, totalizing eight hosts. Each host will make a connection to six other neighbors hosts of other PoDs, totalizing 48 tests per round (8 source hosts x 6 destination hosts).

Each test was structured to collect logs in the different existing stages, as shown in Fig. 2. To carry out the tests, we arbitrarily chose to send 750 packets between a source host and a destination host. Such number of packets is just an example for our evaluations so that we could trigger the failure events. Any other flow sizes may be used constrained to the time for collecting the results.

In the initial stage of each *T0*, 249 packets are sent through the main path to the destination. This first batch generates information about the normal working state of the network. Failures occur on the 250th packet (step *T1*); when we simulate that the ports are *DOWN*, the 250th packet is unable to be routed through the port suggested by the main routing table, so the recovery algorithm is immediately triggered (step *T2*). Monitoring the 250th packet is an important metric for us to discern whether the algorithm was able to bypass the failures without losing packets. During step *T3*, the rerouting algorithm has already managed to bypass the failure and packets are forwarded through the secondary path (backup); at this stage, packet losses are no longer expected and it is possible to assess how much the network has been degraded by the use of the secondary path, either by increasing the number of hops or by increasing the IP Packet Transfer Delay (IPTD) or IP Delay Variation (IPDV). Failure recovery takes place during the 500th packet, featuring step *T4*. In step *T5*, we evaluate the algorithm's ability to return to the main path; subsequent packets travel normally along the main path and belong to step *T6*. The experimentation process ends after the 750th packet.

From the analysis of the results, it was possible to obtain the following measurements: 1) packet loss during the recovery process; 2) IPTD; 3) number of hops (TTL) during the different stages; and 4) IPDV. Packet loss was obtained through the difference between the number of packets sent and received. The IPDT was computed through the difference between the timestamp at which a packet enters the network, and the time timestamp when the packet leaves the network. The number of hops was measured from the Time to Live (TTL) field of the IPv4 header. The IPDV is calculated by the differences between the average of IPTD: the phase in which the packets are forwarded without failure and the phase in which the packets need to bypass failures. Without compromising the comprehension, and since IPDV is calculated from IPDT, IPDV is not shown in this evaluation due to lack of space.

After performing the initial tests where we measured the TTL in our topology, we defined the waiting interval for the return of the cloned packet to 1 second. This value must be at least equivalent to the RTT of the packet up to the failed switch.[9] There is no background traffic in the network.

The centralized Control Plane, described in Section III-B3, was implemented in *Python* and works on a server segregated from the switches, performing periodic pooling on each of the P4 switches. The objective of this algorithm is to detect the occurrence of failures in the network, recalculate the routes and update the routing tables in the switches. In this case, it was observed that the average time to execute queries on all switches in our experimental environment, sequentially, varied between 3 and 5 seconds. Therefore, for the purposes of standardization and data collection, we added a delay to the algorithm, so that queries have a fixed interval of 5 seconds.

The interval for testing the return to the main route and clone the packet was defined as half (2.5 secs) of the Control Plane monitoring interval time, in order to provide a significant reduction in recovery time through the Return to the main route mechanism.

A. *Fat-Tree Results*

Table III shows a summary of the results obtained for the number of hops needed during the steps for the recovery process for 1, 2 and 3 failures. *T0* shows the step without failures. It is observed that all the algorithms (Static, Rotor and InFaRR) were able to use a backup path in the scenario with only 1 failure. During the recovery step (*T2*), all the algorithms needed 7 hops to circumvent the point of failure. However, InFaRR converges to 5 hops after the recovery when using the backup path (instant *T3*), while the Static and Rotor algorithms still need 7. In instant *T6* (not shown in the table), all algorithms return to 5 hops, which is expected. The Control Plane algorithm is not able to forward packets until a Control Plane update is sent to the data plane. In this case, as indicated, there is packet loss regardless of the number of failures.

As can be seen in Tab. III, the algorithms Static and Rotor were not able to deal with multiple failures resulting in loss of

[9] The monitoring of the RTT is not the scope of the InFaRR algorithm. However, there are currently available P4 implementations for RTT monitoring [28].

TABLE III
SUMMARY OF EVALUATION: *Standard Fat-Tree*

Algorithms/ ***N.º of failures***	**Control Plane**	**Static**	**Rotor**	**InFaRR**
1 failure				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packet	7	7	7
N.º of hops Secondary path ($T3$)	5	7	7	5
Meet ITU requirements?	No	Yes	Yes	Yes
2 failures				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packets	No recovery	No recovery	11
N.º of hops Secondary path ($T3$)	5	No recovery	No recovery	5
Meet ITU requirements?	No	No	No	Yes
3 failures				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packets	No recovery	No recovery	13
N.º of hops Secondary path ($T3$)	5	No recovery	No recovery	5
Meet ITU requirements?	No	No	No	Yes

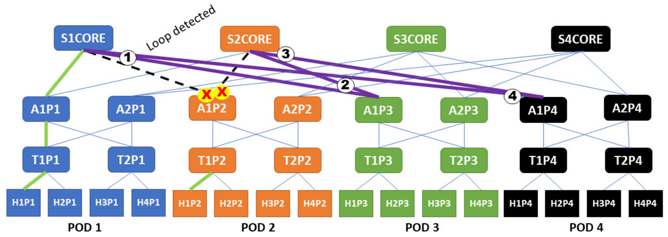

Fig. 9. Occurrence of looping not managed by Static and Rotor algorithms.

packets. InFaRR, on the other hand, could manage the simultaneous failures needing 11 and 13 hops during the recovery step (*T2*), respectively for 2 and 3 failures, returning to 5 hops when using the backup path (*T3*). Note that for the Static and Rotor algorithms, the only solution for recovering from multiple failures is dependent on the control plane which will take seconds for rerouting, causing several damages to applications.

For sake of clarification, we will show in Fig. 9 why the algorithms Static and Rotor are not able to deal with more than one failure. Figure 9 shows a scenario where H1P1 wants to send traffic to H1P2 and needs to deal with two failures, one between link S1CORE and A1P2 and another one between A1P2 and S2CORE. Following the sequence of alternative links available for circumventing the failures, both algorithms will use the link between S1CORE and A1P3 (step 1). A1P3 then uses the link to S2CORE (step 2) which in turn has only the link from S2CORE to A1P4 available (step 3). A1P4 will then send the packet back to S1CORE creating a loop in that equipment. Unfortunately, such situation is quite common in Fat-Tree topologies making the Static and Rotor algorithms not appropriate to deal with more than one failure at the same time.

Fig. 10 shows the same scenario as above, however now with InFaRR in action. The Pushback Mechanism used on the Core switches prevents the loop from occurring and enables recovery. The S1CORE switch performs Pushback (step 1) to A1P1 due to the failure of the link that connects it to PoD2 (A1P2); A1P1 will send the packet to S2CORE, which is also unable to send the packet directly to PoD2 and, again, a Pushback is performed (step 2); the A1P1 switch, given the infeasibility of the main and secondary routing table, sends the packet to its predecessor (step 3). Rerouting from the T1P1 switch takes the packet along a viable path to H1P2, passing through A2P1, S3CORE, A2P2, and T1P2 switches (highlighted in purple). Note that InFaRR does not use other PoDs to circumvent the failure avoiding longer paths in the topology, thus minimizing the usage of network resources.

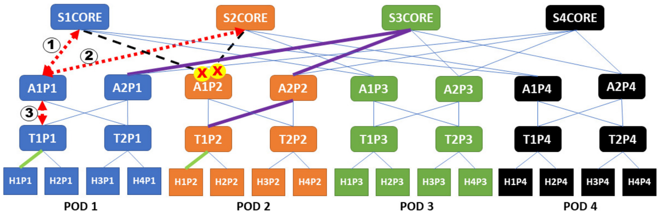

Fig. 10. InFaRR: Pushback mechanism in action to avoid loops.

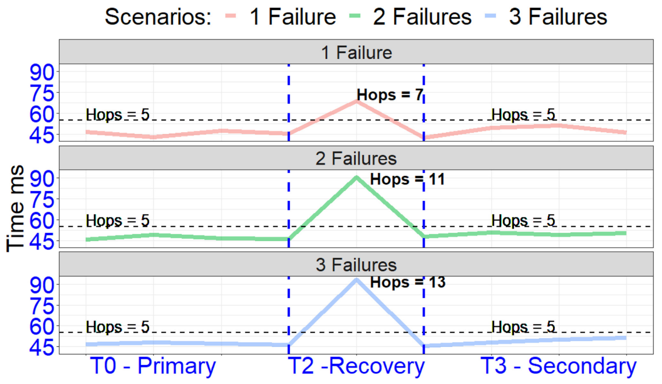

Fig. 11. IPDT time during step *T2*.

Clearly, if a third failure happens between the link from S3CORE to A2P2, no alternative paths are available at all and no recovery is possible. Any other combination of failures is possible to be managed by InFaRR.

Figure 11 presents the IPTD time during step *T2*, specifically for the InFaRR algorithm in different failure scenarios. Step *T2* consists of looking for an alternative path to the destination and, thanks to the Recognition and Restoration Mechanism, there is an increase in the number of hops and IPDT only in this step. It is important to highlight that, in InFaRR, step *T2* is executed only to one packet, and during step *T3* (using the backup path), the other packets will have the same number of hops and IPDT equivalent to the initial step *T0*.

The process to return to the main route associated with step *T5* happens immediately after the port returns to the *UP* state for the Static and Rotor algorithms, as these algorithms act locally on the switches where the fault existed. The InFaRR and Control Plane algorithms promote fault bypass by optimizing the routing table so that packets do not need to go to the faulty switch. InFaRR uses Pushback, Loop Prevention, and Recognition and Restoration mechanisms to provide this

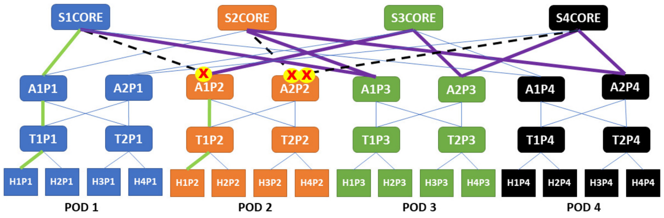

Fig. 12. Rotor algorithm circumventing three failures.

optimization while the Control Plane recalculates routes and sends updates to the switches' routing tables. The InFaRR algorithm uses the Return to Main Route Mechanism to anticipate the use of the main routing table before the recalculation of routes by the Control Plane. The Return to Main Route Mechanism, as it sends packets (cloned packets) at predefined time intervals, does not guarantee return to the main route immediately like the Static and Rotor algorithms.

B. AB Fat-Tree Results

This evaluation presents the results obtained with the AB Fat-Tree topology, which is characterized by the relocation of the connection of some links between the core switches and the aggregation switches.

With the AB Fat-Tree topology, the Rotor algorithm was able to recover from two and three failures, while the Static algorithm is yet not capable of dealing with more than one failure. Figure 12 represents the recovery through the Rotor algorithm in the face of three consecutive failures, in which there was an increase of 120%, from 5 hops to 11 hops to bypass the failures. The line highlighted in purple shows the necessary hops during the recovery process so that the Rotor algorithm can work around the failures.

The rearrangement of the links in the AB Fat-Tree enabled the Rotor algorithm to recover in scenarios where it was previously not possible using the Standard Fat-tree topology. Although the Rotor algorithm was able to manage the failures, the number of hops obtained during the recovery process (step *T2*) remained equal to step *T3* in which the flow to bypass failures uses secondary path. Thus, the Rotor algorithm requires longer paths and extra hops using other PoDs to circumvent the failures. Table IV shows the summary of the results for the AB Fat-Tree. As can be seen, for both *T2* and *T3*, the number of hops are the same: going from 5 to 7 in the scenario with 1 failure, to 9 hops in the scenario with 2 failures and reaches 11 hops in the scenario with 3 failures.

The InFaRR algorithm obtained the same results as with the Standard Fat-Tree topology; thus, the rearrangement of the links between the equipments to support AB Fat-Tree is not necessary for the perfect functioning of InFaRR, thus reducing extra costs for the data center.

VI. Lessons Learned and Limitations

The InFaRR algorithm is designed to exploit up to 3 simultaneous failures in Fat-Tree networks with k=4. The growth of k does not enable greater capacity for simultaneous fault tolerance in InFaRR, since in this case the limit for the Pushback Mechanism is the number of layers. An alternative to increase resilience and use more links between PoDs would be the inclusion of a tertiary routing table.

TABLE IV
Summary of Evaluation: *AB Fat-Tree*

Algorithms/ ***N.º of failures***	**Control Plane**	**Static**	**Rotor**	**InFaRR**
1 failure				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packet	7	7	7
N.º of hops Secondary path ($T3$)	5	7	7	5
Meet ITU requirements?	No	Yes	Yes	Yes
2 failures				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packets	No recovery	9	11
N.º of hops Secondary path ($T3$)	5	No recovery	9	5
Meet ITU requirements?	No	No	Yes	Yes
3 failures				
N.º of hops Normal path ($T0$)	5	5	5	5
N.º of hops Recovery process ($T2$)	Loss of Packets	No recovery	11	13
N.º of hops Secondary path ($T3$)	5	No recovery	11	5
Meet ITU requirements?	No	No	Yes	Yes

Compared to the algorithms adapted from the literature (Control Plane, Static and Rotor), InFaRR obtained a better result because it recovered correctly in all failure scenarios evaluated. The existing programmability in the data plane enabled the coding of InFaRR allowing the detection of the loop and the optimization of the recovery domain, not making it necessary to route the traffic through an external PoD. This feature made it possible to optimize the number of hops during step *T3*- secondary path.

The addition of links to the AB Fat-tree topology actually promotes greater network resilience. The Rotor algorithm, which was unable to recover in scenarios with two and three failures in the Standand Fat-Tree topology, was able to bypass the failures in the AB Fat-Tree topology. The InFaRR algorithm showed no distinction between the topologies, and delivered a forwarding plan that was always optimized thanks to the Pushback and Recognition and Restoration mechanisms.

The use of programmable network cards is a current trend in datacenter networks and can enable the use of the InFaRR algorithm at the end-host [29]. A host with a programmable card with two or more interfaces could use the rerouting features and enable new protection paths for priority flows, extending the routing capability to the host [30].

Currently, InFaRR supports only link failures and future works need to be done to adapt the solution to deal with switch failures as well. The structures allocated inside the P4 switch to each flow needs 288 bits (36 bytes). Thus, 1.47 Mbytes of memory were allocated for the treatment of 40960 different flows. The scalability of the number of flows handled is directly associated with the available memory capacity. In this implementation, we did not perform any process of optimization, grouping and selection of which flows would be protected by the InFaRR algorithm, either through the hosts involved (IP addresses), type of service (TCP ports) or any other QoS classification mechanism.

VII. Conclusion

InFaRR is a fast rerouting algorithm for programmable networks, developed in the P4 language, free of additional management headers and heartbeats. The basis of its operation explores the characteristic of the P4 language match-action on a main table and a secondary table, which can be built with personalized search keys. The InFaRR algorithm has four essential features not found, together, in other recovery mechanisms: Loop Prevention, Pushback, Recognition and Restoration, and Return to the Main Route.

The InFaRR algorithm obtained excellent results, highlighting the success of the recovery process in all evaluated scenarios with the absence of packet loss after the link failure (step *T2*). The experiments were done with one, two and three simultaneous failures on Standard Fat-Tree and AB Fat-Tree topologies.

Given the results obtained, it is understood that the InFaRR algorithm would increase its recovery capacity in hierarchical structures with more layers, since it becomes possible to explore more adjacencies during the execution of routing, rerouting and Pushback Mechanism. The Return to the Main Route Mechanism executed in the data plane makes it possible to recover the routing in the face of remote failures before the intervention of the control plane.

References

[1] L. Kiranmai, L. Lakshmi, and S. P. Kumar, "IP fast rerouting framework with backup topology," *Int. J. Comput. Eng. Res. Trends*, vol. 1, no. 2, pp. 96–103, 2014.

[2] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," *J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, Sep. 2013.

[3] J. Ali, G.-M. Lee, B.-H. Roh, D. K. Ryu, and G. Park, "Software-defined networking approaches for link failure recovery: A survey," *Sustainability*, vol. 12, no. 10, p. 4255, 2020.

[4] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[5] A. Kamisinski, "Evolution of IP fast-reroute strategies," in *Proc. 10th Int. Workshop Resilient Netw. Des. Model. (RNDM)*, 2018, pp. 1–6.

[6] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "D2R: Policy-compliant fast reroute," in *Proc. ACM SIGCOMM Symp. SDN Res. (SOSR)*, 2021, p. 148–161.

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Proc. 10th USENIX Symp. Netw. Syst. Des. Implement.*, 2013, p. 113.

[8] M. Chiesa et al., "PURR: A primitive for reconfigurable fast reroute: Hope for the best and program for the worst," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, 2019, pp. 1–14.

[9] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, and S. Schmid, "Supporting emerging applications with low-latency failover in P4," in *Proc. Workshop Netw. Emerg. Appl. Technol.*, 2018, pp. 52–57.

[10] G. Luz, A. Rocha, L. Almeida, and F. Verdi, "InFaRR: Um algoritmo para reroteamento rápido em planos de dados programáveis," in *Proc. Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBC)*, 2022, pp. 154–167.

[11] R. Chodorek, "QoS measurement and evaluation of telecommunications quality of service [book review]," *IEEE Commun. Mag.*, vol. 40, no. 2, pp. 30–32, Feb. 2002.

[12] *Network Performance Objectives for IP-Based Services*, Rec. Y.1541, ITU-T, Geneva, Switzerland, 2002, pp. 7–9.

[13] R. Stankiewicz, P. Cholda, and A. Jajszczyk, "QoX: What is it really?" *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 148–158, Apr. 2011, doi: 10.1109/MCOM.2011.5741159.

[14] M. Chiesa, A. Kamisinski, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1253–1301, 2nd Quart., 2021.

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

[16] P. Cholda, A. Mykkeltveit, B. E. Helvik, O. J. Wittner, and A. Jajszczyk, "A survey of resilience differentiation frameworks in communication networks," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 4, pp. 32–55, 4th Quart., 2007.

[17] J. Rak, *Resilient Routing in Communication Networks*. Cham, Switzerland: Springer, Nov. 2015.

[18] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implement.*, 2013, pp. 399–412.

[19] S. Lindner, D. Merling, M. Häberle, and M. Menth, "P4-protect: 1+1 path protection for P4," in *Proc. 3rd P4 Workshop Eur.*, 2020, pp. 21–27. [Online]. Available: http://arxiv.org/abs/2001.11370

[20] V. Krishna, N. N. R. R. Suri, and A. Gopalasamy, "A comparative survey of algorithms for frequent subgraph discovery," *Current Sci.*, vol. 100, no. 2, p. 190, 2011.

[21] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing OpenFlow graph algorithms," in *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 121–126.

[22] D. Merling, S. Lindner, and M. Menth, "Robust LFA protection for software-defined networks (RoLPS)," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2570–2586, Sep. 2021.

[23] J. Xu, S. Xie, and J. Zhao, "P4Neighbor: Efficient link failure recovery with programmable switches," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 388–401, Mar. 2021.

[24] Z. Li, Y. Hu, J. Wu, and J. Lu, "P4Resilience: Scalable resilience for multi-failure recovery in SDN with programmable data plane," *Comput. Netw.*, vol. 208, May 2022, Art. no. 108896.

[25] I. Nikolaevskiy, "Scalability and resiliency of static routing," M.S. thesis, School Sci., Aalto Univ., Espoo, Finland, 2016. [Online]. Available: http://urn.fi/URN:ISBN:978-952-60-7194-7

[26] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.

[27] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Burst-tolerant datacenter networks with vertigo," in *Proc. 17th Int. Conf. Emerg. Netw. EXp. Technol.*, 2021, p. 1–15.

[28] S. Sengupta, H. Kim, and J. Rexford, "Continuous in-network round-trip time monitoring," in *Proc. ACM SIGCOMM Conf.*, 2022, pp. 473–485.

[29] Y. Yan, A. F. Beldachi, R. Nejabati, and D. Simeonidou, "P4-enabled smart NIC: Enabling sliceable and service-driven optical data centres," *J. Lightw. Technol.*, vol. 38, no. 9, pp. 2688–2694, May 1, 2020.

[30] Y. Shan, W. Lin, Z. Guo, and Y. Zhang, "Towards a fully disaggregated and programmable data center," in *Proc. 13th ACM SIGOPS Asia–Pac. Workshop Syst.*, 2022, pp. 18–28.

Fábio L. Verdi is an Associate Professor with the Computer Science Department, Federal University of São Carlos (Sorocaba Campus). He has been working with data centers, cloud computing, SDN, and dataplane programmability, leading several national and international projects.

Gustavo V. Luz received the master's degree in computer science from the Federal University of São Carlos. Since 2002, he has been working with Claro Telecom Operator as a Solutions Architect.