

RESISTING: A New Fast-Reroute Mechanism with Packet Distribution on P4-Programmable Switches

Daniel De Lima, Francisco Vogt, Alan Teixeira da Silva, Fabricio Rodriguez Cesen, Christian Esteve Rothenberg
✉ *Universidade Estadual de Campinas (UNICAMP), Campinas, Brazil*

Abstract—Equal Cost MultiPath (ECMP) is a routing solution that optimizes bandwidth efficiency in network infrastructure by using a hash algorithm on the switch’s data plane to distribute packet flows across multiple paths. However, the hash-based approach alone does not address link failures, making the deployment of a failure recovery solution critical to prevent service disruption and degradation. Fast Reroute (FRR) mechanisms enable rapid recovery when a link fails on the data plane, and integrating FRR with ECMP provides a more resilient solution. In this paper, we introduce **RESISTING**, a novel ECMP-FRR mechanism for programmable data plane devices. We evaluate our method against the state-of-the-art FRR mechanism on programmable switches by simulating up to three link failures. The results demonstrate that **RESISTING** achieves equal distribution of packet flows across available links, with no packet losses.

Index Terms—Fast Reroute, FRR, ECMP, P4.

I. INTRODUCTION

Modern data center applications are characterized by stringent performance requirements, including high throughput, minimal packet loss, and reliable failover mechanisms to address link and device failures [1]. Within this context, two pivotal mechanisms — Equal Cost MultiPath (ECMP) and Fast-Reroute (FRR) — are crucial in optimizing network performance and ensuring rapid failure recovery across diverse network architectures, including modern data centers. The ECMP mechanism comprises two components: (i) routing protocols running on the control plane pre-compute the best and shortest paths, which are then installed into the ECMP group on the switch’s data plane, and (ii) the ECMP group utilizes a hash algorithm and employs a method to distribute traffic based on the 5-tuple flow hash digest [2]. This approach selects a single path for flow forwarding from a group of multiple paths computed by routing protocols.

The failure recovery strategies that utilize routing applications within distributed data planes or centralized architectures demonstrate significantly slower recovery processes than rerouting mechanisms employed directly on the data plane [3]. To mitigate long periods of packet interruptions and service degradation caused by slow solutions, FRR has emerged as a solution on the data plane to enable quick failure recovery in networking. This method reroutes traffic to a backup path during unexpected failures [4]. It can be used with ECMP instead of slower solutions to reroute traffic from an affected path within the ECMP group to an operational path on the data plane. That results in faster recovery times, reducing packet flow degradation during failure events.

In P4-based architectures, a straightforward solution for FRR [3] entails continually redirecting all packet flows from the failed port to a backup port using packet recirculations. However, this approach can result in increased packet latency and decreased throughput, particularly as the number of failures rises, leading to a gradual escalation of packet recirculations within the data plane [5]. The works [4], [6] propose more efficient recovery methods that maintain latency and throughput while minimizing memory usage on programmable devices. However, both simple and advanced FRR mechanisms may be unsuitable for ECMP link recovery, as they redirect all affected packet flows to a backup path without distributing them across other operational links in the ECMP group. This can result in an uneven flow distribution during failures, overloading the backup link and reducing overall network resilience.

The present work proposes **RESISTING** (*datacenter Equally coSt multiIpath faSt re-rouTINg*): a new FRR mechanism integrated with ECMP for programmable switches, aiming to offer resilience, robustness, and packet flow distribution in the data center after failure events recovery. Our mechanism can recover from one or multiple simultaneous failures in the links used by ECMP traffic balancing. Upon detecting a failure in an ECMP balancing link, the architecture’s FRR recovery process initiates removing the output port of ECMP associated with the failed link through a port reordering routine of the remaining operational links in the balancing. After the recovery process, the packet flows previously affected by the failure are redistributed evenly among the operational balancing links. The main contributions of this paper can be summarized as follows:

- The design of an FRR mechanism integrated with ECMP for rapid failure recovery and optimal flow distribution;
- A P4 prototype implementation for BMv2 and Tofino hardware with open-source artifacts for reproducibility¹;
- An experimental evaluation showcasing **RESISTING**’s performance and resilience under simultaneous failures.

II. BACKGROUND & RELATED WORK

A. Programmable networks

Traditional computer network architecture [7], built upon conventional devices made with “closed” and proprietary technologies, has been essential for reliable data transmission

¹Github repository - <https://github.com/ecmpfrr>

through the past few decades. However, the addition of approximately 3 billion internet users over the past 15 years² caused a dramatic rise in network requirements (i.e., new protocols, functionalities, and capabilities) to address customer demands. However, the traditional architecture is complex to adapt and innovate [8], and it depends on vendors to change networks. The SDN [9] architecture emerged as a response to previously mentioned limitations, introducing concepts such as separation of data and control plane and programmable networks.

Despite advances in SDN architecture, the data plane had a challenging intrinsic inflexibility. The authors in "Reconfigurable Match Tables" (RMT) [10] proposed a solution to address this limitation, introducing chip technology applied in physical switch architecture able to enable data plane programmability, in addition to the emergence of P4 language.

B. Fast-Reroute

Network resilience is a crucial attribute that ensures the ability of a communication infrastructure to deal with failure scenarios while trying to maintain an acceptable level of service quality [11]. In our work, network resilience is directly related to the mechanisms and strategies for fault recovery, which can be implemented in both network devices' control and data planes of network devices. In general, resilience mechanisms in the data plane operate via hardware, enabling recovery processes in milliseconds or even microseconds [3]. The FRR mechanism provides quick connectivity restoration without using the control plane [12], minimizing packet degradation caused by network interruptions. Conversely, software-based recovery in the control plane is slower, potentially taking tens to hundreds of milliseconds, depending on the network scenario and method employed.

C. ECMP load balancing

ECMP is a routing scheme that distributes packet flows among links with equal cost [13]. Initially, it learns routes to a destination in the control plane or SDN controller. Then, multiple alternate paths are installed within the data plane's ECMP mechanism. ECMP randomly distributes packet flows among links upon route provisioning using a hash algorithm. To address failures, integrating ECMP with FRR can enable ECMP fast recovery while maintaining the load balancing [3].

D. Related Work

Concerning the P4 language, there isn't a native FRR primitive in software and hardware, and the network operator is responsible for developing the FRR mechanism. An approach involves using packet recirculation functionality: when a failure event is detected, the structure implemented through tables or conditionals recirculates affected packets in the data plane, allowing them to start a new processing cycle to find a backup path. This technique increases overhead and latency due to the continuous packet recirculation on the device [4].

The work in [6] introduces bit-indexed explicit replication (BIER), a multicast routing concept proposed by the

IETF RFC 8279 [14]. BIER simplifies router multicast routing, reducing resource (CPU and Memory) consumption compared to protocol-independent multicast (PIM). The authors emphasize implementing BIER in P4 and integrating two recovery mechanisms: IP-FRR for unicast traffic protection and BIER-FRR for multicast. These mechanisms redirect affected packet flows to backup paths. In 1:1 contingency scenarios, this solution can be effective. However, they do not distribute affected packet flows among multiple operational paths in scenarios with traffic balancing.

PURR [4], [5] is a FRR primitive applied to the programmable data plane in P4 that efficiently recovers one or multiple simultaneous failures without compromising throughput and latency and implements a table in the ingress pipeline that conducts a ternary search for the next operational output port. Then, it redirects packet flows affected by the failure to the available port, regardless of the number of simultaneous failures. However, the adopted strategy involves redirecting all packets from the affected path to a single backup path.

The authors in [15] proposed an approach for failure recovery that reroutes traffic directly in the data plane and configures a set of forwarding rules in the control plane to handle failure scenarios. Although considering alternative rules associated with a failure scenario in the control plane, the suggested approach lacks an ECMP scenario and an actual device implementation, such as programmable switches or SmartNICs to verify the performance.

In environments with heavy traffic where load balancing services are necessary to prevent congestion and ensure efficient link usage, the recovery strategy, which redirects affected packet flows to a single backup path, may result in an imbalance in the distribution of packet flows among the links. Furthermore, it could exacerbate saturation conditions on the backup path during failure events.

III. RESISTING

Our work encompasses the development of two P4 implementation models: one for the BMv2 platform and another specifically tailored for Tofino Native Architecture (TNA). While the P4 algorithm developed is consistent across both platforms, differences arise due to distinct architectural types.

A. Architecture & Implementation

The architecture proposed offers a set of P4 features, including an FRR mechanism dedicated to ECMP recovery as a primary capability. Additionally, we provide packet forwarding based on IPv4 and TAG, an ECMP mechanism for traffic load balancing, a temporary packet recirculation, and a technique for transmitting data via additional headers inserted into packets between the ingress and egress pipeline. These P4 components are incorporated into four types of data plane functionality: *Packet forwarding*, *Load balancing*, *Recalculation*, and *Recovery*.

1) *Packet Forwarding*: This functionality consists of components responsible for handling packet processing definitions on the data plane, such as method parsing, IPv4 routing, and

²ITU DATA HUB 2024 - <https://datahub.itu.int/>

failure detection. Under normal conditions, without failures, the packet goes through parsing stages to analyze the headers and extract all relevant fields from layers 2-4, which will be used in the pipeline process.

The routing component decides where to forward the packet by analyzing the destination IP address. Consider an example where the next-hop is a local host connected to its own switch; in this case, the routing stage sets an appropriate outgoing port. After that, the packet forwarding flow proceeds to the port status table, which controls the operational status of the outgoing port by determining whether the port is up or down. This mechanism is controlled by rules installed via the control plane. In another context, if the packet needs to reach a remote host in another switch, a TAG, an identification number, is added to the destination MAC in the Ethernet frame [16]. This TAG labels the destination switch and enables routing all packets between top-of-rack switches within the infrastructure, as seen in a Clos leaf-spine topology [17]. Then, the packet forwarding process proceeds to the load balancing step.

2) *Load Balancing*: In general, load balancing approaches in P4 are based on tables that perform lookups for keys from header fields or metadata. The result of this operation can be an outgoing port, which represents a static rule installed in the data plane. Additionally, the number of next hops parameters controls how many paths the hash algorithm can balance. Thus, during process failure recovery, when there is a need to update outgoing ports and the number of next hops parameters from the load balancing tables, the control plane acts as the failure recovery mechanism by modifying rules in the data plane. However, this process is slower than the FRR mechanism. To address this challenge, we designed a solution based on registers responsible for storing outgoing ports and the number of next hop parameters, enabling any changes through a failure recovery mechanism applied in the data plane.

In our work, *load balancing* utilizes a table to perform lookups for key values, similar to the P4 load balancing approaches mentioned earlier. However, the operation results in link values instead of outgoing ports. The outgoing ports are stored in another set of components, built by tables and registers, denominated as `forwarding_tag`. Each link number is mapped to a dedicated `forwarding_tag` component, which stores an outgoing port used by ECMP.

Another adaptation involves creating a specific register, named `max_links`, to store and control the number of next-hop parameters. Thus, the FRR implementation can update outgoing ports and the `max_links` value during process failure recovery. Figure 1 depicts the ECMP Hash and `forwarding_tag` abstraction. In this context, packet flows incoming from the routing stage ingress the Hash table for load balancing processing, taking into account the installation of all rules into components, including TAG routes, four links (0-3), four outgoing (P1-P4), and defining the total number of next-hops as 4.

The algorithm first performs a TAG lookup on the table to identify the destination switch for the packet flow; in this example, the switch is identified as 10. This value represents

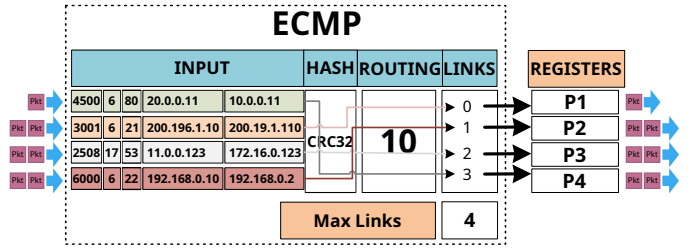


Fig. 1. ECMP hashing and registers.

an ECMP group associated with four links and ports. Then, the hash algorithm uses packet fields as input parameters to calculate the hash result, which maps to a specific link among the four balancing options and is associated with an outgoing port. Lastly, the flows are directed to the respective interface.

The `port_status` component is a table that stores the operation status of outgoing ports from the switch. This approach is a solution [12] that allows simulating port failures in the BMv2 and Tofino switches. The packets proceed to the last stage when an outgoing port is up. Then, the packet header fields are rebuilt, and the packets are directed to the physical port of the switch. However, if the port is down, it triggers the beginning of the FRR process. At this stage, the affected packet flows are not dropped; instead, they are redirected to the FRR process. The flows traverse the ingress and egress pipelines within this process through the recirculation method.

Highlighting that packets are recirculated twice, the FRR header is added to all packets to indicate their status in recovery. Only the first packet detected after a failure assumes the responsibility for transmitting data between the ingress and egress pipeline components during the FRR process. This process does not affect other flows, and those unaffected by the failures continue to follow the standard packet forwarding.

3) *Recalculation*: Once a port has failed, the architecture must take action to remove the port from the operation, reorganize available ports, and adjust the number of links within the ECMP mechanism. To accomplish this, we implemented a similar ECMP group structure, incorporating the same links associated with outgoing ports. In our design, we refer to these tables and registers as `FRR_Port_Out`.

A scenario depicting **RESISTING** process is shown in Figure 2. The port 3 (`P3`) fails, with the following link-to-port associations: ($L0 \rightarrow P1$), ($L1 \rightarrow P2$), ($L2 \rightarrow P3$), ($L3 \rightarrow P4$), ($L4 \rightarrow P5$) and ($L5 \rightarrow P6$). During the recalculation process, when the first packet arrives from the ECMP step, carrying information about the link position failure, the `FRR_Port_Out` tables identify the link positions as $L2$ with a port down by inspecting the appropriate FRR header field and triggering the recalculation processes.

4) *Recovery*: Once the algorithm identifies the port that needs recovery, it initiates a port migration process, which involves moving ports from the last link position to the next link position until they reach the link position corresponding to the port failure. The process concludes with the following assignments: ($L2 \rightarrow P4$), ($L3 \rightarrow P5$), ($L4 \rightarrow P6$), and

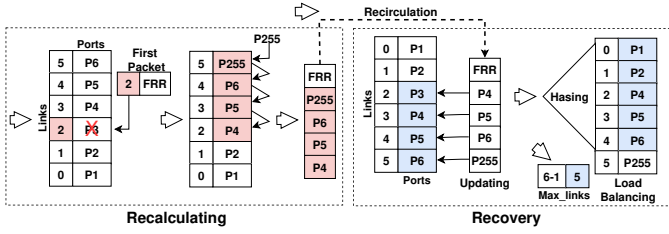


Fig. 2. ECMP-Fast-Reroute mechanism.

($L5 \rightarrow P255$) — the ports of links $L0$ and $L1$ do not require updates in this case. In sequence, the algorithm writes a new port order in the header FRR fields of the first packet. Subsequently, the packet follows to the next stage, referred to as *updating*, through the recirculation method. This component comprises a set of tables responsible for updating ECMP within the ingress pipeline by copying the new port order from the packet to the appropriate registers `forwarding_tag` while simultaneously subtracting the value of `max_links` to reduce the number of link operations applied in hash calculation. After this update, the ECMP group begins operating with the link-to-port associations ($L0 \rightarrow P1$), ($L1 \rightarrow P2$), ($L2 \rightarrow P4$), ($L3 \rightarrow P5$) and ($L4 \rightarrow P6$), excluding the port down ($P3$), resulting in a total of five links ($L0 - L4$) in the hash calculation.

The recovery process in the Tofino prototype is similar to that applied in BMv2, as illustrated in Figure 2. However, when the hash function results in traffic being forwarded to the last link (or last links if there are more failures) associated with port 255 ($P255$), an additional routine is triggered to redirect the traffic to an operational port. The `default_path` table, located following the `forwarding_tag` forwarding tables in the ingress pipeline, intercepts traffic to direct it to a port that always remains operational, as long as at least one port is operational in the ECMP. This approach ensures that traffic is not discarded, even in cases of subsequent new failures.

B. Tailoring for Tofino

The main functional constraint encountered in the TNA platform was the inability to reduce the number of operational links used in the hash function through the data plane. This occurs because the ECMP mechanism available on the platform does not allow storing the parameter value defining the number of links in ECMP via registers - which is crucial for the recovery process as it is manipulated to reduce the outgoing paths used in the modular operation of the hash algorithm. Thus, we developed a second prototype with an additional structure for the Tofino implementation that redirects traffic from the link that cannot be subtracted in ECMP to an available output port.

IV. EXPERIMENTAL EVALUATION

In order to assess the performance, resilience, and overhead of the proposed mechanism, we performed different evaluations in the simulated and real environment. Our experimental evaluation has the following objectives:

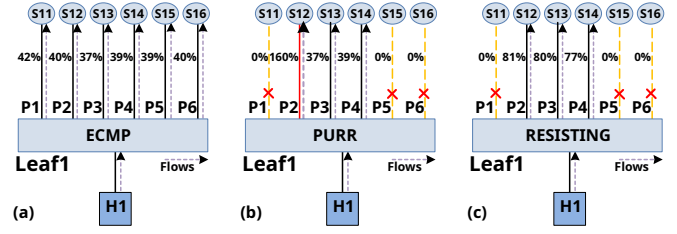


Fig. 3. Resilience simulation scenarios.

- 1) Evaluate the resilience of RESISTING against PURR [4], the state-of-the-art in FRR, when subjected to simultaneous failures.
- 2) Evaluate the overhead introduced by RESISTING in terms of recirculations during the recovery process and in terms of resource utilization on the Tofino Switch.

To achieve these objectives, we set up two experimental environments using P4-programmable Ethernet switches: one with Mininet and the software switch BMv2 (simulated environment) and another with an Intel Tofino switch (real environment). Next, we discuss these two environments, the experiments performed, and the results obtained.

A. Resilience Experiment

This experiment evaluates the proposed architecture's resilience in simulated ECMP link failures compared to PURR, the current state-of-the-art.

Experimental Setup. The experiments use a Mininet emulator featuring a *Clos* topology. The Mininet environment was established to run the network topology for each FRR mechanism. The simulated network balances packet flow among operational uplinks while maintaining an occupancy of less than 50% per link. Thus, starting from this baseline scenario, failures are applied to the uplinks of switch leaf 1, and subsequently, the **RESISTING** and PURR mechanisms are activated for recovery. Each of the six uplinks can support a maximum of 1000 flows, totaling 6000 packet flows. The host simultaneously generates 2400 distinct TCP packet flows, aiming to allow ECMP to achieve a balanced distribution among the six uplinks with an approximate occupancy of 400 flows on each uplink (i.e., about 40% occupancy).

In this scenario, PURR employs a recovery strategy that redirects all packet flows from unavailable links to a single backup link, disregarding the balance of affected packets. On the other hand, our solution utilizes a recovery method that removes the links affected by failures from the ECMP mechanism, enabling the continued distribution of packet flows among ECMP operational paths. Equation (1) on Traffic Delivery (TD) calculates the resilience for **RESISTING** and PURR in all failure simulations, computing the difference between Value Demanded (VD) and LOSS, where VD represents the total packet flow sent, and LOSS represents the packets discarded.

$$TD = \frac{VD - LOSS}{VD} \quad (1)$$

TABLE I
AVERAGE TRAFFIC DELIVERY TD_{AVG} RATES

Mechanism	TD_{AVG} (%)		
	1 Failure	2 Failures	3 Failures
PURR	100%	96%	86%
RESISTING	100%	100%	100%

Equation (2) computes the average of TD (TD_{Avg}) values obtained for all simulations by dividing the sum of TD results by the total number of evaluated network topologies (ENT). In the experimental context, network topology refers to a measured traffic scenario.

$$TD_{Avg} = \frac{\sum TD}{ENT} \quad (2)$$

Figure 3 demonstrates the execution of TD calculations by examining three emulated. Initially, in Figure 3a, the host connected to leaf 1 generated a demand for 2400 distinct flows. This corresponds to the VD base for all tests. Subsequently, the ECMP performed a distribution of packet flows among the uplinks in the following proportions: 429 (42%) for $P1$, 406 (40%) for $P2$, 371 (37%) for $P3$, 393 (39%) for $P4$, 399 (39%) for $P5$, and 402 (40%) for $P6$. Each link supports a maximum of 1000 flows. If the link exceeds 100%, we consider the overrun discarded.

In Figure 3b and Figure 3c, simultaneous failures were applied to ports $P1$, $P5$, and $P6$. After the failure event, the PURR recovery mechanism redirected packet flows from the three uplinks to port $P2$, as illustrated in Figure 3b. As a result, port $P2$ exceeded 100% occupancy, resulting in a TD of 0.735. This means only 73% of the VD (2400 flows) were delivered, with a LOSS of 648 (27%) flows. In Figure 3c, after detecting the failures, **RESISTING** removed ports $P1$, $P5$, and $P6$ from the ECMP balancing, allowing the demand from the three unavailable uplinks to be redistributed among the remaining links, avoiding packet flow overload and delivering 100%.

Resilience Results. For both **RESISTING** and PURR mechanisms, 1, 2, and 3 failures were applied to 6, 15, and 20 simulations, respectively. Initially, after the completion of TD calculations, the results were applied to (2), which calculates the Average TD of the experiment. Table I shows the results of both strategies. Regarding a single failure, **RESISTING** and PURR achieved a resilience level of 100%, delivering all packet flows in the simulations, regardless of which uplink port suffered unavailability in leaf 1.

In the scenario with two failures, PURR showed loss results in 6 port combinations: ($P1, P2$), ($P1, P6$), ($P2, P3$), ($P3, P4$), and ($P4, P5$). The average TD obtained was 96%. Therefore, 4% of the average flows suffered a loss. In this scenario, our architecture delivered 100% of the demand for all possible port combinations during the simulations.

Finally, in the drastic scenario with three failures, only two failure combinations did not suffer loss by PURR: ($P1, P3, P5$) and ($P2, P4, P6$). The other simulations showed a loss; for this reason, the average degradation result was 14%, while the **RESISTING** did not present packet flow loss in the

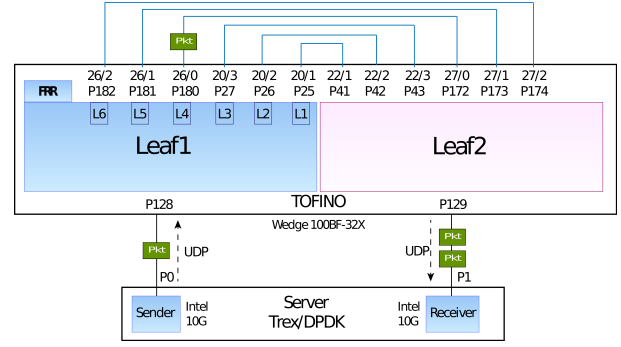


Fig. 4. Experimental topology for overhead experiments.

20 evaluated topologies. **RESISTING** demonstrated superior resilience capacity, especially during three-failure events.

B. Overhead Experiment

The overhead experiment measures the total number of packets recirculated from the packet flows affected by failures in ECMP links during the **RESISTING** recovery process.

Experimental Setup. These simulations use a server to generate UDP packets of large (1514 B), medium (814 B), and small (114 B) sizes for 10 seconds from network interface $P0$ (h1) to network interface $P1$ (h2), covering four different transmission rates: 100 Mbps, 1 Gbps, 5 Gbps, and 9 Gbps while applying 1, 3, and 5 simultaneous failures. This experiment uses the Intel Tofino switch hardware and one physical server to create a physical topology as illustrated in Figure 4. The Tofino is a 3.2 Tbps Wedge 100BF-32X running Intel’s *Software Development Environment* (SDE) version 9.9.0. The server is a 2.20 GHz Intel Xeon CPU running DPDK and contains 62 GB of RAM and an X552 10 GbE SFP network interface. The switch is a single physical switch emulating two leaves (leaf 1 and leaf 2). We benefit from Tofino’s multi-pipeline support with appropriate table entries. The leaves are physically connected with six 10 Gbps uplink cables. Traffic is generated by a Trex/DPDK on the server side with two 10 Gbps network cards that simulate host transmitting (Tx) and receiving (Rx). Our prototype enables leaf 1 to perform ECMP balancing and FRR, configured with six links ($L1$ - $L6$), each associated with logical ports $P25$, $P26$, $P27$, $P180$, $P181$, and $P182$.

Overhead Results. A correlation between packet size and the number of applied failures can be observed in Figure 5. When traffic is subjected to fault conditions, as packet size decreases and the number of failures increases, the number of recirculated packets increases. It is important to note that increasing the transmission rate from 100 Mbps to 1 Gbps, 5 Gbps, and 9 Gbps resulted in a proportional increase in the total number of generated packets. However, this variation in transmission rates did not affect the number of recirculated packets, which remained the same across all tested rates. For example, with 114-byte packets, 1 million packets were generated at 100 Mbps and 95 million at 9 Gbps, yet the number of recirculated packets remained the same during each failure event. The overall resource usage of **RESISTING** P4

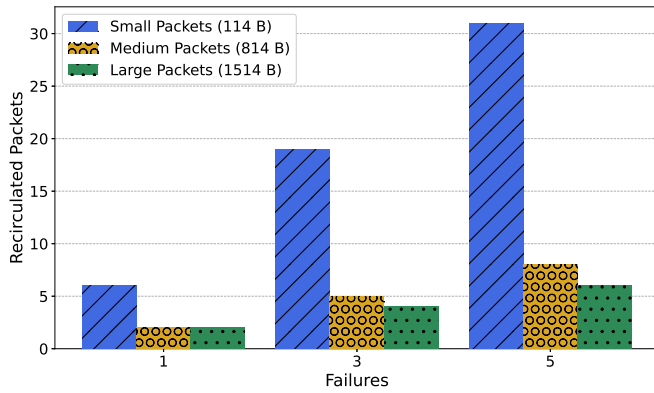


Fig. 5. Relationship between number of failures and recirculated packets.

code running in the Tofino hardware is depicted in Figure 6. It is possible to observe the distribution of resources along the pipeline stages. Notably, none of the resources overlap the 30%. Having this in mind, the resisting code can be allocated to other network functions or implementations that do not restrict the use of the network device.

C. Discussion

The resilience results in Section IV-A show that while both strategies effectively handle a single failure, only RESISTING maintains performance in scenarios with multiple failures. In contrast, PURR experiences packet loss with two and three failures. This occurs because RESISTING efficiently distributes flows across multiple links, whereas PURR redirects them to a single link, leading to congestion and packet loss in multi-failure scenarios. This highlights not only the superior resilience of RESISTING but also its greater scalability compared to PURR.

Although RESISTING uses recirculations in its recovery process, experiments in Section IV-B show that recirculations account for only $3 \times 10^{-7}\%$ of the total packets in the worst case. Additionally, Each packet can be recirculated up to twice, and with each recirculation taking $\approx 500ns$, the maximum added delay for a packet is $\approx 1000ns$. Finally, RESISTING demonstrates low resource usage, enabling it to coexist with other P4 solutions like QoE monitoring [18] or microburst detection [19].

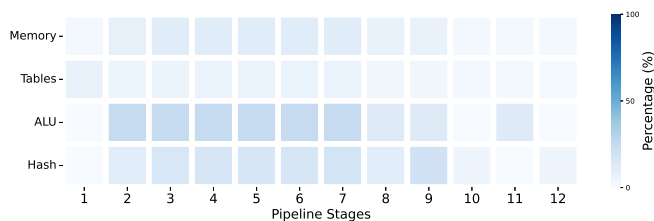





Fig. 6. RESISTING resources usage in the Tofino Switch.

V. CONCLUSIONS AND FUTURE WORK

This study proposes a new FRR method that works in conjunction with ECMP on programmable switches using the

P4 language, removing the link affected by a failure by reallocating the ports of the remaining links in the ECMP. After the failure recovery, packet flows are evenly distributed among multiple operational links in the balancing. As future work, we plan to implement **RESISTING** in different architectures and devices (e.g., SmartNICs), introducing more sophisticated balancing mechanisms that consider packet flow size and/or transmission rate as additional criteria for flow distribution, integrating methods for detecting the operational state of ports or links with the recovery architecture.

ACKNOWLEDGMENT

This work was supported by Ericsson Telecomunicações Ltda. , and by the Sao Paulo Research Foundation , grant 2021/00199-8, CPE SMARTNESS .

REFERENCES

- [1] F. Rhamdani *et al.*, “Equal-cost multipath routing in data center network based on software defined network,” in *6th ICoICT*, 2018, pp. 222–226.
- [2] L. Champagne and B. Donnet, “Smoothie: Efficient and flexible load-balancing in data center,” in *NOMS. IEEE/IFIP*, 2024.
- [3] M. Chiesa *et al.*, “A survey of fast-recovery mechanisms in packet-switched networks,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.
- [4] M. Chiesa, R. Sedar *et al.*, “PURR: a primitive for reconfigurable fast reroute,” in *Proceedings of the 15th CoNEXT '19*, pp. 1–14, 2019.
- [5] R. Sedar *et al.*, “Supporting Emerging Applications With Low-Latency Failover in P4,” in *Proceedings of the 2018 NEAT Workshop*. ACM, 8 2018, pp. 52–57.
- [6] D. Merling *et al.*, “P4-based implementation of BIER and BIER-FRR for scalable and resilient multicast,” *Journal of Network and Computer Applications*, vol. 169, p. 102764, nov 2020.
- [7] Cisco, “Cisco visual networking index: Forecast and trends, 2017-2022,” Cisco Systems, Tech. Rep., 2019.
- [8] L. Csikor *et al.*, “High availability in the future internet,” in *The Future Internet*. Berlin: Springer Berlin Heidelberg, 2013, pp. 64–76.
- [9] L. Yang *et al.*, “A survey on network forwarding in software-defined networking,” *Journal of Network and Computer Applications*, vol. 176, p. 102947, 2021.
- [10] P. Bosshart *et al.*, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn,” in *Proceedings of the ACM SIGCOMM 2013 Conference*, 2013, p. 99–110.
- [11] J. Sterbenz *et al.*, “Evaluation of network resilience, survivability, and disruption tolerance: Analysis, topology generation, simulation, and experimentation: Invited paper,” *Springer Telecommunication Systems*, 12 2011.
- [12] G. V. Luz *et al.*, “InFaRR: Um algoritmo para roteamento rápido em planos de dados programáveis,” in *SBRC*, 2022.
- [13] C. Hopps, “Rfc2992: Analysis of an equal-cost multi-path algorithm,” USA, 2000.
- [14] I. Wijnands *et al.*, “Multicast Using Bit Index Explicit Replication - BIER,” RFC Editor, RFC 8279, November 2017.
- [15] J. A. Marques *et al.*, “Responding to Network Failures at Data-plane Speeds with Network Programmability,” in *NOMS 2023 IEEE*, 2023, pp. 1–10.
- [16] A. Greenberg *et al.*, “VL2,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 51–62, aug 2009.
- [17] M. Alizadeh and T. Edsall, “On the data path performance of leaf-spine datacenter fabrics,” in *2013 IEEE 21st HOTI*, 2013, pp. 71–74.
- [18] F. Vogt, F. R. Cesen, A. G. deCastro, M. C. Luizelli, C. E. Rothenberg, and G. Pongrácz, “QoEyes: towards virtual reality streaming QoE estimation entirely in the data plane,” in *2023 IEEE NetSoft*. IEEE, 2023, pp. 267–271.
- [19] F. G. Vogt *et al.*, “Innovative network monitoring techniques through in-band inter packet gap telemetry (IPGNET),” in *Proceedings of the 5th EuroP4*, 2022, pp. 53–56.