# A Survey on Rerouting Techniques with P4 Programmable Data Plane Switches

Ali Mazloum, Elie Kfoury, Jose Gomez, Jorge Crichigno
College of Engineering and Computing, University of South Carolina, Columbia, U.S.A
Email: amazloum@email.sc.edu, ekfoury@email.sc.edu, gomezgal@email.sc.edu, jcrichigno@cec.sc.edu

***Abstract*—Traditionally, the networking industry has been dominated by closed and proprietary hardware and software. Vendors have been controlling the network by hard-coding how packets should be processed and providing the network operators with a set of predefined protocols. Recently, the industry, operators, and the research community have started to pay special attention to data plane programmability, which allows the user to define the packet processing behavior. Allowing the network operators and programmers to define, deploy, and test new forwarding behaviors in a relatively short time paved the way for a significant wave of innovation and experimentation. With the emergence of programmable data planes, traffic rerouting has been used by the research community. Rerouting approaches are deployed to mitigate various network issues. Despite the considerable number of works that deploy innovative rerouting mechanisms using programmable switches, the literature lacks a comprehensive survey. To this end, this paper provides an in-depth overview, detailed analysis, and unique categorization of the recent programmable data plane-based rerouting approaches. The survey explains the need for rerouting by highlighting the promising results while dealing with link/node failures, load imbalance, and congestion. It then discusses the challenges and considerations and presents future perspectives and open research issues.**

***Index Terms*—Traffic rerouting, Programmable data planes, P4 language, Challenges and solutions in P4**

## 1. Introduction

As the network scale is increasing, failures are becoming more frequent [1]. After link/node failures, the network enters a reconvergence state, re-establishing the affected paths. During the reconvergence period, some traffic may have an invalid route, at which a reliable path between the sender and the receiver may not be available. Under failures, real-time applications might suffer from significant performance degradation caused by packet loss during network re-convergence.

Besides availability, low latency becomes a key requirement for the ongoing evolution of the Internet. For example, Tactile Internet is the result of the giant leaps the Internet has made over the years [2]. The International Telecommunication Union (ITU) describes the Tactile Internet as the combination of ultra-low latency with extremely high availability, reliability, and security [3]. If a network node receives more data than the outgoing link can process, the node's buffer will be congested. The more packets accumulated inside the node's buffer, the higher the queuing delay. The typical latency needed by Tactile Internet applications is less than 1 millisecond (ms) [4]. To achieve such low latency, Tactile Internet flows should not suffer from any delay nor be dropped by any node on the path [5].

Furthermore, congestion may result from unfair load balancing of traffic. Load imbalance is a major problem in cloud computing [6]–[8]. Load balancing can be achieved by minimizing the maximum link utilization while minimizing resource consumption [9], [10]. In data centers and cloud architectures, applications are replicated over multiple application instances. All the application instances providing the same service share the same virtual IP (VIP). An average 44% of cloud and data center traffic is VIP traffic [11], [12]. After connection initialization, a load balancer is required to map the incoming packets only to the application instance serving it. This property is known as per connection consistency (PCC). In case PCC is broken, the connection will be reset.

**Limitations of Legacy Approaches.** Upon link/node failures, networking nodes forward the traffic based on the routes computed by the control plane. Unfortunately, the average convergence time of the control plane upon remote failures is above 30 seconds [13]–[15]. One reason behind this slow reaction is that the control plane is responsible for achieving convergence through the propagation of routing updates on a per-router and per-prefix basis.

Moreover, congestion control mechanisms of traditional transport protocols should not be deployed over Tactile applications because modifying the sending rate is not allowed at the Tactile source [5]. An alternative approach to these congestion control mechanisms is the use of congestion-signaling mechanisms (e.g., ECN [16]). Congestion-signaling mechanisms avoid congestion as the receiver notifies the sender to reduce the data rate before packet loss happens. However, these mechanisms might not function if they are not implemented on all the network devices between the sender and the receiver [17]. Furthermore, traditional switches partially support load-aware traffic splitting [18]. The most deployed data plane load-balancing technique is equal-cost multi-path routing (ECMP) [19]. ECMP does not adapt to the state of the network and splits the traffic randomly among the available paths, congesting the network under some circumstances [20].

Traditional switching Application Specific Integrated Circuit (ASIC) is not capable of holding a load balancer [11]. Instead, a software load balancer (SLB) is usually implemented, which can easily keep PCC at the cost of unfair

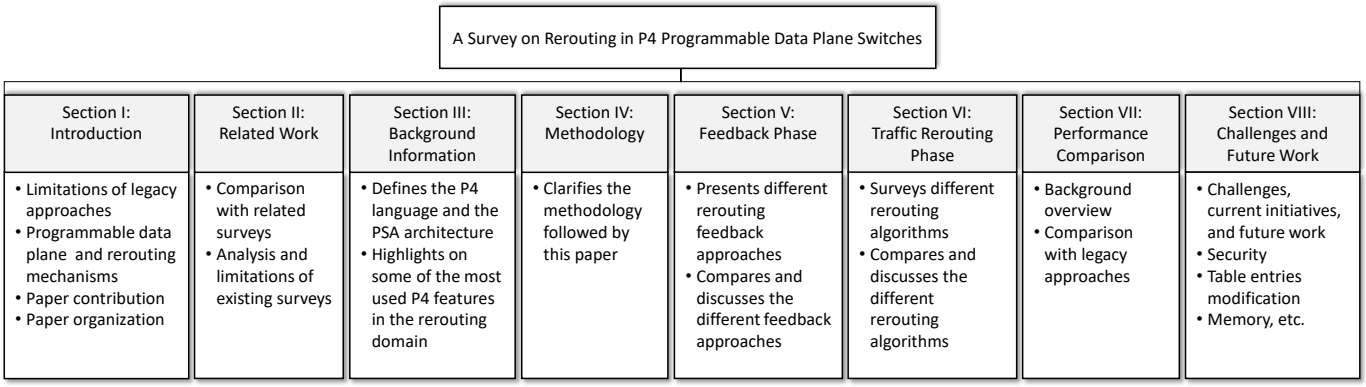


Fig. 1. Paper roadmap.

load balancing. Compared to the characteristics required by the tactile applications, SLBs provide high latency and jitter and poor performance isolation [21].

**Programmable Data Plane and Rerouting Mechanisms.** The emergence of programmable switches paved the way for unprecedented innovation and experimentation in the data plane while sustaining the same packet-processing throughput [20], [22]. Programmable forwarding plane increases network reliability, offers greater visibility, provides efficient use of resources, and allows users to add new features. Programmable switches can provide fine-grained statistics at line-rate. This technology can achieve terabits per second (Tbps) throughputs and can offer a line-rate speed feedback exchange between different nodes [5], [23]. In this survey, routers, switches, network nodes, and forwarding elements are used interchangeably.

The research community leverages the provided characteristics and proposes a wide range of varying rerouting approaches. Each rerouting approach can be uniquely identified either by its rerouting feedback, its rerouting algorithm, or by both [19], [24]. New traffic rerouting mechanisms are deployed to deal with link/node failures, load imbalance, and congestion. These approaches benefit from data plane level fault detection and line-rate speed of fine-grained measurements collection while sustaining high packet-processing throughput. Furthermore, many legacy rerouting approaches can be implemented on the data plane, benefiting potentially from a significant increase in performance. For instance, SWIFT [13], GONGA [25], DRILL [26], Clove [27], and others [28] are state-of-the-art rerouting approaches that can be deployed over programmabledata plane switches.

### *1.1 Contribution*

The emergence of data plane programmability resulted in the recent adoption of different rerouting mechanisms. However, to the best of the authors' knowledge, no survey has addressed rerouting approaches in programmable switches. To this end, this paper discusses data plane programmability and stresses which of its characteristics rerouting approaches benefited from. The paper presents some limitations of legacy approaches while dealing with link/node failure, load imbalance, and congestion; explaining how traffic rerouting combined with programmable data plane technology might result in a promising solution for all the aforementioned issues. The paper continues by exploring different rerouting feedback approaches; discussing their common challenges and some used solutions; and analyzing the rerouting logic of multiple mechanisms. It provides a high-level illustration of how internal components of programmable switches communicate to provide a flexible yet line-rate performance while traffic is being rerouted, discusses challenges and considerations, and puts forward future perspectives and open research issues.

### *1.2 Organization*

The paper's organization is summarized in Fig. 1. Section 2 studies and compares existing surveys on programmable data plane rerouting related topics and demonstrates the added value of the offered work. Section 3 presents an overview of P4 programmable switches and highlights some of the P4 features which are leveraged by the rerouting approaches. Section 4 provides the methodology followed by this survey. Section 5 analyzes what special events can trigger the rerouting process by presenting different feedback approaches, performing internal comparison between these approaches, and concludes by comparing feedback mechanisms in programmable approaches with traditional mechanisms. Section 6 surveys, classifies, analyzes, and discusses the rerouting algorithms of various approaches. Section 6 concludes by comparing rerouting algorithms in programmable switches internally and externally with legacy algorithms. Section 7 compares the performance of the rerouting approaches with respect to legacy mechanisms. Section 8 outlines challenges and considerations extracted and induced from the literature and pinpoints directions that can be explored in the future to ameliorate the state-of-the-art solutions. Finally, Section 9 concludes the survey. The abbreviations used in this article are summarized in Table IX, at the end of the article.

## 2. Related Work

Rerouting approaches are comprised of two phases. The first phase is the feedback phase, where a network node detects a

TABLE I
COMPARISON WITH RELATED SURVEYS.

| Paper | Programmable Switches | | Feedback Phase* | | Traffic Rerouting Phase* | | Discussions | |
|---|---|---|---|---|---|---|---|---|
| | Features | Need | Link/Node Failures | Load Imbalance and Congestion | Link/Node Failures | Load Imbalance and Congestion | Challenges | Future Directions |
| [29] | ⊙ | ⊙ | ⊙ | ○ | ⊙ | ○ | ⊙ | ⊙ |
| [30] | ● | ⊙ | ○ | ⊙ | ○ | ⊙ | ⊙ | ⊙ |
| [31] | ● | ⊙ | ○ | ⊙ | ○ | ⊙ | ● | ● |
| [32] | ⊙ | ⊙ | ○ | ○ | ⊙ | ○ | ⊙ | ⊙ |
| [33] | ● | ⊙ | ⊙ | ⊙ | ⊙ | ⊙ | ⊙ | ⊙ |
| [34] | ⊙ | ⊙ | ○ | ○ | ○ | ○ | ⊙ | ● |
| [35] | ⊙ | ⊙ | ○ | ○ | ○ | ○ | ⊙ | ● |
| [36] | ● | ⊙ | ○ | ○ | ○ | ○ | ● | ● |
| [37] | ● | ● | ○ | ⊙ | ○ | ⊙ | ● | ● |
| This paper | ● | ● | ● | ● | ● | ● | ● | ● |

* Each traffic rerouting approach has two phases, a feedback phase and a traffic rerouting phase. The feedback phase describes the mechanism followed by programmable switches to detect and notify other switches about the occurrence of a problem (e.g., link failure, load imbalance, and congestion) in the network. The traffic rerouting phase describes the rerouting process programmable switches deploy to route the traffic.

● Covered in this survey ○ Not covered in this survey ⊙ Partially covered in this survey

problem in the network and notifies other nodes. The second phase is the traffic rerouting phase, where one or more nodes collaborate to reroute the traffic. Rerouting techniques are used to avoid failed link/node, to load balance, and to control congestion [18]–[20].

- *Link/node failures*: are common in large networks [1]. All traffic routed through a failed link/node will be stalled. Failed link/node are avoided by rerouting the traffic through an alternative path.
- *Load imbalance*: is a major problem in cloud computing [6]–[8]. In cloud computing, multiple resources can provide clients with the same set of services. A client can get a certain service by communicating with any resource providing the service. Load imbalance occurs when some resources are overloaded while others have low workloads [38].
- *Congestion*: is the primary source of packet loss in the network [39]. Congestion occurs when a network node receives more data than its outgoing link can forward. All packets that cannot be directly forwarded by a congested node are stored in the node's buffer. The node starts to drop packets when the buffer is full. Congestion can lead to a significant degradation in performance [40].

The implementation of each phase can vary significantly from one P4-based rerouting approach to another. To the best of the authors' knowledge, no previous work focuses on traffic rerouting approaches in programmable switches. The remaining of this section shows how related work falls short in addressing the two phases of P4-based rerouting approaches.

### *2.1 Feedback Phase*

Cordeiro *et al*. [32], Kaljic *et al*. [35], Satapathy *et al*. [34], and AlSabeh *et al*. [36] surveyed programmable data plane switches without discussing approaches that resolve link/node failures, load imbalance, and congestion problems. First, Cordeiro *et al*. discussed the opportunities and challenges for network and services operations for data plane programmability beyond OpenFlow. The authors presented the evolution of SDN from OpenFlow to data plane programmability and explained some of the domain-specific languages for the data plane. After that, the authors discussed the opportunities the programmable data plane offers and the challenges it proposes over security, dependability, accounting, performance, and configuration management. The survey implicitly mentioned rerouting by listing a few load balancing approaches. The survey did not discuss the feedback phase of any load balancing approach mentioned in it. Similar to Cordeiro *et al*., Kaljic *et al*. discussed data plane flexibility and programmability in SDN [41]. The survey started by overviewing the architecture of the data plane and the definitions of network flexibility and programmability in other domains. It then described the hardware and software-based technologies which support data plane implementations. The authors then surveyed several problems in seven categories; however, they did not discuss the rerouting mechanism.

Satapathy *et al*. presented a study report on the P4 programming language. The study explained the P4 language and provided some design considerations. It then described how to set up the environment of the P4 program on Ubuntu virtual machines. After that, it mentioned some use cases of P4 and concluded with future work. Finally, AlSabeh *et al*. provided a concise background on programmable switches and their main features that are relevant to security. The work surveyed, classified, and analyzed articles related to security applications developed with P4. The authors continued by employing a STRIDE analysis to examine vulnerabilities related to general P4 applications. The survey was concluded by examining general P4 challenges, presenting how they are related to the security domain, and discussing future endeavors and open research problems. This survey is limited to P4-based security applications in which the rerouting topic is barely discussed.

Chiesa *et al*. [29] surveyed a few P4-based approaches that focus on link/node failures. The authors discussed different

packet-based fast-recovery mechanisms in packet-switched networks. The work described different networking technologies, from traditional link-layer and IP-based mechanisms, over Border Gateway Protocol (BGP) and Multiprotocol Label Switching (MPLS) to emerging software-defined networks and programmable data planes. Most of the approaches that are discussed in this work run over legacy switches. A few P4-based approaches that resolve link failures were described in this paper. For the described approaches, the authors did not show what features of programmable switches are utilized to detect the problem and to notify other switches about the failure. On the other hand, Hauser *et al* [42]. focused on P4-based approaches. The survey discussed the approaches that resolve link failures, load imbalance, and congestion. It described the transition from traditional networks and SDN to data plane programming. It then described the two most common data plane programming models and introduced the P4 programming language. After that, it surveyed P4-based applied approaches in monitoring, traffic management and congestion control, routing and forwarding, advanced networking, network security, and miscellaneous domains. Although this survey included numerous different rerouting-based approaches, it only described the high-level details. The survey did not explain the feedback phase of the approaches that were described in it.

Kaur *et al*. [30] and Gomez *et al*. [37] focused on P4-based approaches that resolve load imbalance and congestion. In particular, Kaur *et al*. discussed the scalability and performance issues in Software-defined Network (SDN) systems and illustrated how data plane programmability could handle these problems. The work highlighted the need for programmable switches and presented the working flow model of P4 programmable data plane switches. However, the feedback phase of the rerouting approaches covered in this survey was briefly described. Gomez *et al*. also discussed P4-based approaches that resolve load imbalance and congestion. The survey focused on schemes that enhance TCP performance. It classified the aspects that impact TCP's behavior (e.g., congestion) and surveyed the P4-based solutions. In this survey, only a few rerouting approaches were discussed.

Similar to [30] and [37], Kfoury *et al*. [31] covered P4-based approaches that balance the load and control congestion. The work included the evolution, description, and features of programmable switches, compared the P4-based approaches internally and to legacy approaches, and discussed the challenges and future work. The paper divided the literature into seven categories and briefly illustrated different works under each group. The authors overviewed multiple rerouting approaches without explaining their feedback phase.

### *2.2 Traffic Rerouting Phase*

Chiesa *et al*. [29], Cordeiro *et al*. [32], and Hauser *et al*. [42] partially described the traffic rerouting phases of some P4-based rerouting approaches. In [29], the focus was on legacy rerouting approaches, where the authors briefly mentioned the traffic rerouting phase of some P4-based rerouting approaches while dealing with link failures. In [32], the authors mentioned some traffic rerouting techniques deployed over programmable switches. The focus of the survey was to show the added value of data plane programmability on SDN systems, where a few rerouting techniques were briefly described as use cases for programmable data planes. The authors did not show how the rerouting phase was implemented on the switch or what features of the data plane programmability have been utilized. Besides discussing rerouting approaches that deal with link failures, Hauser *et al*. [42] discussed P4-rerouting approaches that resolve load imbalance and congestion. The authors did an exhaustive survey that discusses data plane programmability and its applications on multiple domains, including the applications that resolve link/node failures, load imbalance, and congestion. However, the authors did not provide sufficient details about the surveyed work, as the rerouting phase was not explained in enough detail.

Similar to [42], Kfoury *et al*. [31] discussed the P4-based applications that handle load imbalance and congestion. However, the authors did not survey the approaches that resolve link/node failures. Besides, the rerouting phase of the surveyed approaches (approaches that resolve load imbalance and congestion) was briefly discussed without showing what features of data plane programmability each approach had utilized. Kaur *et al*. [30] and Gomez *et al*. [37] also did not focus on P4-related approaches that resolve link/node failures. The authors focused on the approaches that resolve load imbalance and congestion without explaining the different implementations of the traffic rerouting phase.

### *2.3 Survey's Scope and Novelty*

Table I summarizes the topics and the features described in the related surveys. It also highlights how this paper differs from the existing surveys. All previous surveys lack a critical analysis of the rising traffic rerouting mechanisms. This survey addresses the gap by explaining clearly the need for rerouting as a promising solution for the current challenging requirements. This work gathers programmable data plane mechanisms that are based on rerouting and analyzes them deeply. The analysis includes a comprehensive explanation of the different feedback mechanisms adopted by the rerouting approaches and a detailed illustration of the most up-to-date rerouting algorithms.

## 3. BACKGROUND INFORMATION

Table II compares P4 programmable switches and legacy forwarding devices. This section introduces the P4 language and the Portable Switch Architecture (PSA), and then highlights some of the programmable switches' features that are leveraged by rerouting approaches.

### *3.1 P4 Language and the Portable Switch Architecture (PSA)*

P4 language is used to define how the data plane of a programmable switch should process the packets. PSA defines a set of "packet paths." Different packet paths might have different processing logic. The programmer writes P4 programs to control the flow of packets in different packet paths [43].

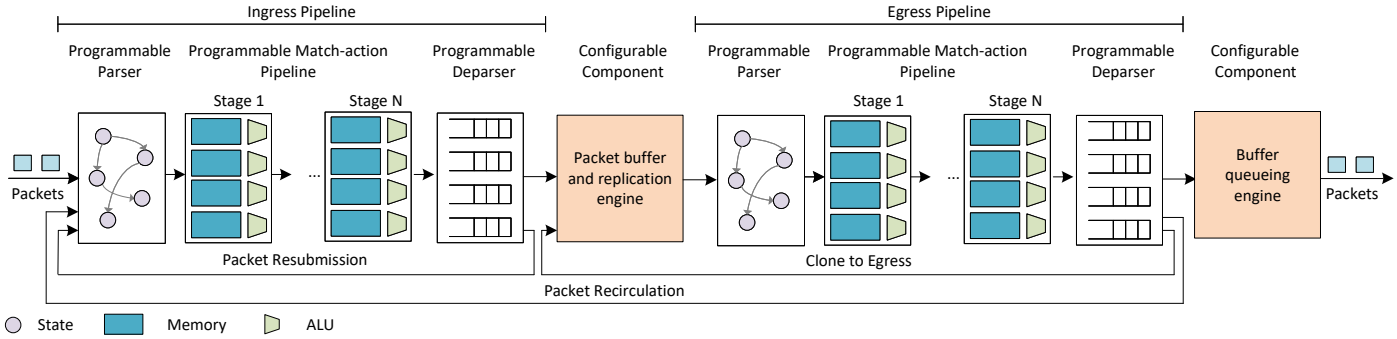


Fig. 2. Portable Switch Architecture (PSA) [43]. The PSA model consists of P4 programmable blocks and fixed-function blocks. The behavior of the programmable blocks is specified using P4. The Packet Buffer and Replication Engine (PRE) and the Buffer Queuing Engine (BQE) are target-dependent functional blocks that may be configured for a fixed set of operations.

Consider Fig. 2. The PSA model consists of P4 programmable blocks and fixed-function blocks. The first programmable block is the Parser. Incoming packets are parsed and validated according to the headers defined by custom or standard protocols. The packets then move to the ingress match-action pipeline. The match-action pipeline is used to define the way packets are processed inside the switch. The match-action pipeline contains multiple match-action tables. A table has one or more entries that are populated by the control plane. Each entry has a key, an action, and an action data. The key is used for the lookup operation. The switch builds a key for the incoming packets using their headers. The built key is used to search the match-action tables. Once a hit occurs, i.e., the built key matches a key for one of the entries inside the match-action tables, the action associated with the matched entry is performed using Arithmetic Logic Units (ALUs). Match-action tables have multiple memory blocks and ALUs. The memory blocks can be Static Random-Access Memory (SRAM) or Ternary Content Addressable Memory (TCAM). SRAM holds multiple stateful objects (e.g., registers, counters, meters) where further action logic can take place.

The next programmable block is the ingress deparser which has control over the packet contents and metadata to be sent to the Packet Buffer and Replication Engine (PRE). The programmer has the option to replicate (i.e., copies made to multiple egress ports) and store the packet in the PRE. The PRE is the first fixed-function block. The first three programmable blocks (i.e., the programmable ingress parser, the programmable ingress match-action pipeline, and the programmable ingress deparser) are usually denoted by the ingress pipeline. After the PRE, packets pass through the egress pipeline, which has a similar structure to the ingress pipeline. The two pipelines may share the same physical components, which reduces the implementation cost [22], [44].

### *3.2 Packet Generation*

Rerouting approaches can leverage the packet generators of the programmable switches to enhance the failure detection speed. Packet generators generate packets after being triggered by a specific event. The packet generator hardware can monitor the status of the ports. If one of the ports goes down, the packet generator generates a packet with a unique header that matches an entry in the match-action tables [45]. The programmer can define the triggering events and the actions to be taken after receiving the triggering signal. Rerouting approaches can detect, report, and avoid the down ports at line-rate by leveraging the packet generators.

The packet generators can also be used to implement the bidirectional forwarding detection (BFD) protocol. BFD is a network protocol used to detect link failures between different forwarding elements [46]. By leveraging the packet generators to deploy BFD, rerouting approaches can offload the link failure detection to the data plane, which decreases the dependency on the control plane and thus increases the overall performance. With packet generators, the probe generation gap between simultaneous probes can have a nanosecond granularity, a feature that might not be achievable using the control plane.

Besides, rerouting approaches can leverage the timers of the packet generators. There are two kinds of timers. The first one is a periodic timer. The other is a one-shot timer. Timers can be leveraged to generate probes. Such probes are always generated at the data plane level. The vendors provide APIs that allow the control plane to configure all different features in the packet generators.

Packet generation in programmable switches is not limited to the packet generator hardware. Previous work was able to leverage the packet recirculation and packet replication features of programmable switches to generate 1Tbps throughput [47]. Programmable switches are capable of generating packets while collecting measurements with nanosecond granularity [48]. Because the generated packets might have custom headers, P4TrafficTool [49] was introduced to generate plugins for the most common packet generators and analyzers tools (e.g., Wireshark [50], MoonGen [51], Scapy [52], and Pcap++ [53]), allowing them to process custom headers.

### *3.3 Header Modification*

Among the significant set of available actions provided by the match-action pipelines, line-rate header modification is one

TABLE II
COMPARISON BETWEEN PROGRAMMABLE AND LEGACY SWITCHES IN THE CONTEXT OF REROUTING.

| Feature | Programmable Switches | Legacy Switches | Notes |
|---|---|---|---|
| Packets Generation | Support failure detection; provide periodic and one-shot timers | Not supported | Switch can detect down ports and failed links at line-rate without the control plane intervention |
| Header Modification | Support custom header modifications | Restricted to the set of standardized header modification operations | Operator can implement custom headers to propagate rerouting signals or encapsulate rerouting paths |
| Multicasting | Support customizable algorithms and efficient dynamic tree updates; result in high packet overhead | Restricted by signaling protocols; has no packet overhead | Operator can implement a custom algorithm to propagate probes and rerouting signals |
| Telemetry | Provide microsecond granularity; support event-base monitoring; detect microbursts; bandwidth overhead* | Not supported; low bandwidth overhead | Switch can translate network events into rerouting signals at line-rate |
| Programmability | Support custom packet processing behaviors; simpler testing process | Fixed packet processing behaviors; complex testing processing | Operator can define and test new rerouting approaches on a real hardware in a relatively short time |

* The overhead is proportional to the percentage of packets being reported.

of the most adopted techniques by the rerouting approaches. The programmable parser allows the programmer to define the headers of the incoming packets according to custom or standard protocols. The programmer can either build his new header or modify the existing fields by programming the actions inside the match-action tables. This characteristic is leveraged by rerouting approaches for different reasons. Some approaches modify the headers to propagate the link utilization information [19], [20], [54]. Others use packet headers to propagate the list of failed links [55].

### *3.4 Multicast*

Multicast routing enables a source node to send a copy of a packet to a group of nodes. Some rerouting approaches deploy multicast techniques to control the propagation of probes. These techniques play a fundamental role in determining the efficiency and sometimes the feasibility of a proposed rerouting algorithm. Multicast uses in-network traffic replication to ensure that at most a single copy of a packet traverses each link of the multicast tree [31]. The multicast tree is the set of different paths a multicast packet should follow. In most standard multicast techniques, the multicast tree is saved on the switches. This adds a burden on the memory resources of the switches as the size of the multicast tree increases. Using P4-based multicast, the programmer can include the multicast tree inside the packet headers. No information about the multicast tree has to be included on the switch. Changes to the multicast tree can then be done by modifying the headers without the need to store any information related to the multicast tree at the switch [56]. This feature makes the addition of a new node straightforward.

Fig. 3 depicts a topology with two multicast trees. The identifications of all network nodes participating in a multicast tree are encapsulated inside a custom header. By processing the custom header of an incoming packet, the switch identifies the directly connected nodes to which the packet should be delivered. If the same packet should be forwarded to multiple next-hops, the current programmable switch utilizes the replication engine to create additional copies of the packet. Before forwarding the packets, the programmable switch removes its identification from the custom header.

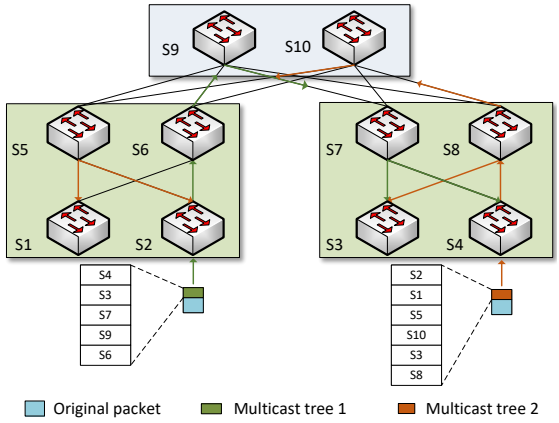


Fig. 3. P4-based multicast. The multicast trees are encapsulated inside a custom header. Programmable switches process the custom header to identify the next-hops.

### *3.5 Fine-grained Telemetry*

By using the programmable switches, operators can obtain per-packet visibility at line-rate. This visibility allows many rerouting approaches to leverage data plane signals to enhance their detection mechanism. These signals can be per-node signals (e.g., queuing and processing delays) or signals at the level of the network (e.g., TCP retransmissions). Some rerouting approaches leverage the per-node signals to detect and react to congestion. Many traditional congestion control algorithms need at least one round-trip time (RTT) to react to the perceived congestion [5]. In contrast, using programmable switches, the exact value of the queue occupancy can be measured, and immediate action might be taken at line-rate.

Other rerouting approaches leverage network-wide signals to resolve remote link failures. Traditional detection techniques need tens of seconds to detect remote failures [13]. In contrast, by monitoring the data plane signals, some rerouting approaches reduce the detection time of remote failures to one RTT [18].

Rerouting approaches can mainly utilize three monitoring techniques, flow monitoring, sketches, and In-band Network Telemetry (INT) [57].

- *Flow monitoring:* in the flow monitoring technique, traffic is monitored on a per-flow basis [33]. The collected records can be either used by the data plane to perform threshold-based actions [58], [59] or can be forwarded to the control plane, where further processing can be done [60]–[62].
- *Sketches:* as the number of flows increases, monitoring all flows becomes infeasible mainly because of memory limitations in programmable switches [31]. An alternative technique to flow monitoring is the use of sketches. Sketches are the output of streaming algorithms that approximate a measurement rather than outputting its exact value. Streaming algorithms are used when there is a memory limitation. The accuracy of a streaming algorithm depends on the number of items being monitored and on the memory resources available. A common use of sketches is detecting heavy flows [63], [64].
- *INT:* INT utilizes packet metadata to monitor the state of the network (e.g., congestion) or the internal state of a programmable switch (e.g., queue occupancy, link utilization, queuing delay, etc.) [31], where the metadata to be defined depends on the application [65], [66]. A key advantage of INT is that data monitoring and actions based on the collected data can happen at the data plane level without interacting with the control plane. Switches can have a network-wide view by exchanging their telemetry data. Data sharing can happen by appending the metadata to packet headers or using probes [67]–[69]. As described in Fig 4, an INT-enabled network has four parts: 1) INT source: is the node responsible for indicating what metadata should be collected from later nodes in the network. The node indicates the telemetry data to be collected by appending a custom header containing the instructions to the original packet; 2) INT transit hop: is the node that processes the instructions encapsulated by the INT source node and appends its metadata accordingly; 3) INT sink: is the node responsible for extracting the appended instruction custom header and metadata; and 4) INT sink: is the device responsible for collecting the accumulated metadata from the INT sink.

### 3.6 Full Programmability

For decades, the networking industry operated in a bottom-up approach. A fixed-function ASIC is at the bottom and enforces available protocols and features to the programmer at the top. This closed-design paradigm has limited the capability of the switches to proprietary implementations, which are hard-coded by vendors, inducing a lengthy, costly, and inflexible process. This limitation prevents many novel ideas from being implemented on real hardware. As mentioned before, the PSA architecture, which is programmed by the $P4_{16}$ programming language, has six fully programmable blocks. This open-design paradigm resulted in a much faster cycle

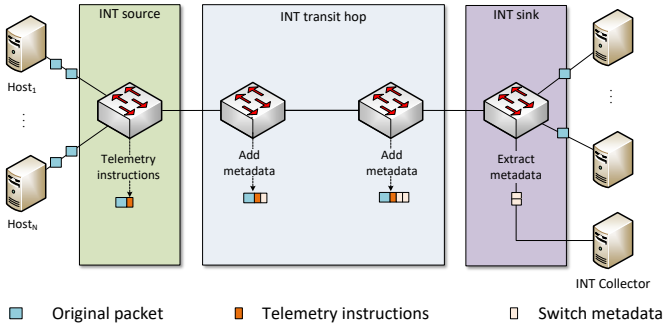


Fig. 4. In-band Network Telemetry (INT).

of innovation and experimentation, reduced complexity, and enhanced resource utilization of the programmable switches. This flexibility allows the operators and programmers to step out of the traditional restrictions and implement their own customized algorithms.

## 4. Methodology

Consider Fig. 5. The rerouting lifecycle consists of four steps. In the first step, a forwarding device detects the occurrence of a networking problem (e.g., link/node failure, load imbalance, congestion). In the second step, the device reports the existing problem to other devices in the network. These two steps constitute the feedback phase of a rerouting approach. In the last two steps, the device obtains the backup path and then modifies its forwarding rules. The last two steps constitute the traffic rerouting phase of the rerouting approach.

Although rerouting approaches that leverage programmable switches are not constrained by the aforementioned rerouting lifecycle, most of them can still be described as a combination of feedback and traffic rerouting phases.

Each phase of the rerouting process is discussed in a separate section. The implementation of each phase is further illustrated. Generally, there are three ways to implement the feedback phase: 1) data plane signals can be utilized to infer the occurrence of link/node failure or congestion; 2) probes can be used to collect information about the links utilization in a network or to detect failed links/nodes; and 3) packet headers can be customized to send network-assisted congestion feedback or to transmit the rerouting signals from one device to another. In the traffic rerouting phase, rerouting can happen either reactively or proactively. In reactive rerouting, the calculation of the alternative path happens after detecting the need for rerouting. In proactive rerouting, the alternative path is calculated and stored inside the switch before detecting the problem.

## 5. Feedback Phase

### 5.1 Traffic Behavior

Multiple approaches perform rerouting after monitoring and analyzing the traffic behavior. Some approaches leverage TCP reliability to detect failure. Others analyze processing and queuing delays to infer congestion. It is proven by

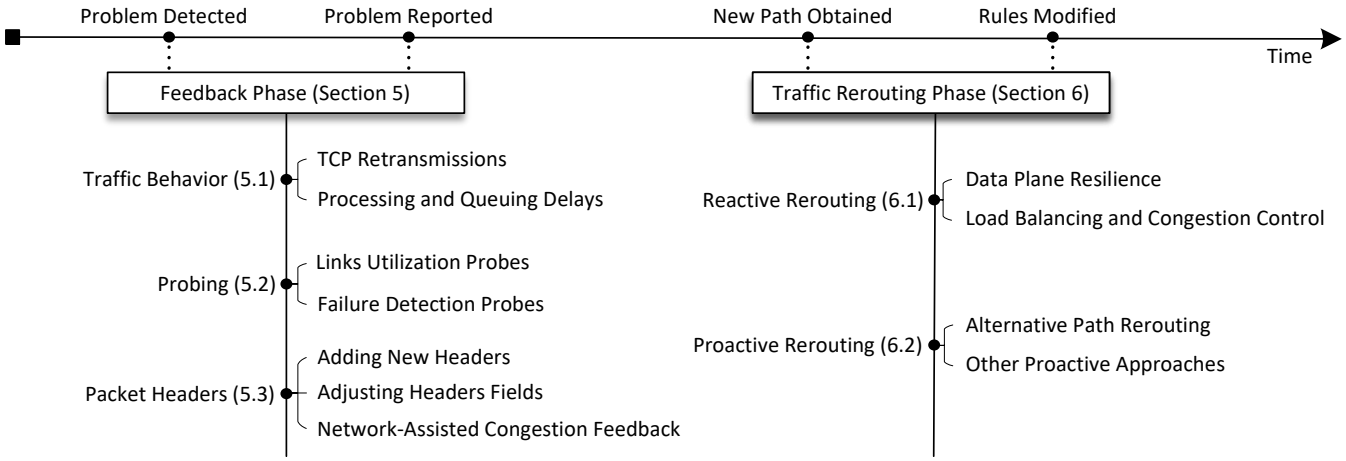


Fig. 5. Rerouting lifecycle and the methodology followed by the survey.

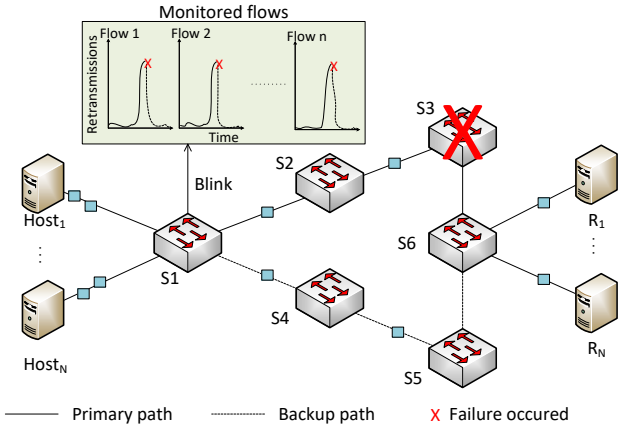


Fig. 6. Workflow of Blink. A programmable switch monitors the retransmissions of n flows. The switch reroutes the traffic when the flows start to retransmit packets simultaneously.

[5], [18] that traffic has predictable behavior upon specific changes in the Internet and that rerouting might be inferred through monitoring the traffic.

**TCP Retransmissions.** TCP retransmissions occur when the sender does not receive an acknowledgment of a sent packet within a period of time [70]. In Blink [18], the programmable switch monitors the incoming flows and uses TCP retransmissions as an indication of a remote link failure. In case multiple flows start to retransmit the same packets repeatedly, Blink considers that link failure occurred. The workflow of Blink is depicted in Fig. 6. Programmable switch S1 monitors $n$ flows out of the $N$ flows traversing the switch. When S3 fails, the monitored flows traversing S3 start to retransmit packets simultaneously. S1 detects the retransmissions and reroutes the traffic through the backup path.

**Processing and Queuing Delays.** Processing delay is the time taken by a switch to process the packet header. After the header is processed, the packet waits in a queue until it can be transmitted. The time the packet spent inside the queue represents the queuing delay. Turkovic *et al.* [5] propose fast network congestion detection and avoidance using P4, a mechanism that aims to detect and react to any increase in delays at the level of the switch. This approach leverages the programmable data plane to access network traffic measurements in real-time.

As shown in Fig. 7, fast detection deploys a local congestion module (denoted by Local Module in the figure) that is capable of obtaining processing and queuing delays besides monitoring the state of low-latency flows. Low-latency flows are characterized by a low end-to-end latency. Typically, the end-to-end latency should be less than 1ms [5]. A central controller can set the latency thresholds, the detection threshold for processing and queuing delays, and the number of consecutive packets with increased delay to detect congestion. The local congestion control module detects congestion based on the instructions received from the central controller and the collected measurements from the data plane.

### *5.2 Probing*

Probes are the action taken by network elements to share information about the utilization of the links and the occurrence of a link/node failure [71]. By considering the shared information, the switch might decide to reroute the traffic through an alternative path. The probing mechanism is leveraged by the programmable data plane switches to detect the need for rerouting. Probes can be used to gather global link utilization. Global link utilization is collected by measuring the utilization of all links in a network. Probes are also used to detect failed links.

**Link Utilization Probes.** In [19], [20], [72], special probes traverse the network periodically to gather global link utilization information. These approaches leverage programmable switches to update the load of the traversing

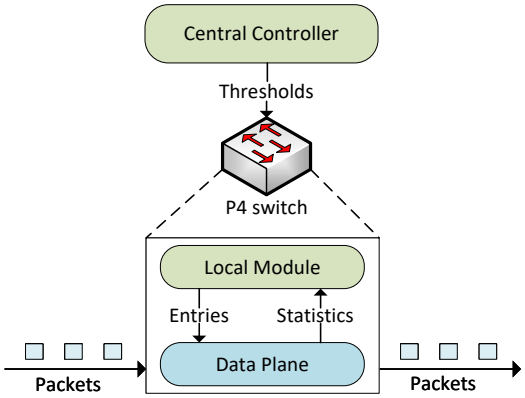


Fig. 7. Workflow of [5]. A central controller configures the latency thresholds and other parameters. The P4 switch gathers statistics (e.g., processing and queuing delays). The configured thresholds and gathered statistics are forwarded to the local module. Upon congestion detection, the local module modifies the entries of the data plane.

customized probes. For instance, HULA [19] probes cover all desired paths for load balancing. These probes are sent by all top-of-rack (ToR) switches through the uplinks that connect them to the data center network. In a data center architecture design, a ToR switch is used to connect servers within the same rack to aggregate switches [73]. Aggregate switches are used to connect ToR switches to a spine switch [74]. The spine switch is the backbone of the network, and it is responsible for interconnecting all ToR switches. In HULA, the probes are generated by the control plane. After that, the probes are propagated through the network. The receiving node adds its ports' utilization to the probe and then forwards it. As shown in Fig. 8, probes are sent by ToR *T1*, one on each of the uplinks connecting it to the aggregate switch *A1*. Once a probe reaches *A1*, it replicates it and forwards the copies to all other downstream ToRs (*T2*) and upstream spines (*S1, S2*). Spine *S1* replicates the received probe onto all the other downstream aggregate switches. When *A3* receives a probe from *S1*, it replicates it to all its downstream ToRs but not to any upstream spine. This makes sure that the probes cover all paths in the network. This also makes sure that no probe loops forever. Once a probe reaches another ToR, its journey ends.

**Failure Detection Probes.** SPIDER [23] provides a simple failure detection scheme based on the exchanged bidirectional "heartbeat" packets. In particular, if packets stop arriving at a certain port for a given interval, the failure detector is triggered, and the node asks its neighbor to send a heartbeat. The node can assure that its neighbor is reachable by receiving a heartbeat. However, if no heartbeat or a data packet is received for more than a predefined interval, the idle port will be declared down. As soon as new packets arrive, the port will be declared up again. The authors assume that failures are temporal. They use the packet duplication mechanism provided by the programmable switches to periodically check

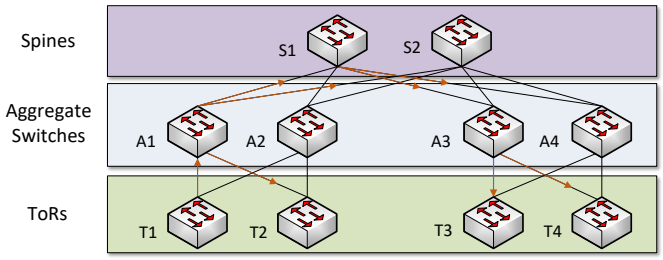


Fig. 8. HULA probe replication logic. The probe is generated at ToR T1 and follows the paths indicated by the orange arrows.

the availability of a failed node.

### *5.3 Packet Headers*

Besides probing, network elements can propagate information about the utilization of the links and about the occurrence of a link/node failure by encapsulating the gathered measurements inside packet headers. The encapsulation can be done either by adding new headers or by modifying the load of the existing ones. In either case, the packets reflect, dynamically, the representation of the network state, i.e., the links' utilization and the address of the failed link/node.

Furthermore, packet headers can be modified by the programmable switches to send network-assisted congestion feedback to the end hosts. The feedback is used to notify the end hosts about the congestion before it occurs. Programmable switches can notify end hosts by overriding header fields (e.g., ECN field), adjusting header fields (e.g., adjusting TCP advertised window), or trimming header fields and modifying packets (e.g., generating NACK packets). This section discusses how packet headers are used as a rerouting notification mechanism and network-assisted congestion feedback.

**Adding New Headers.** D2R [55] follows the detection mechanism proposed by the Failure Carrying Packets (FCP) protocol [75]. In FCP, each packet includes information about the failed links it encountered in its path. D2R assumes that network topology on the Internet does not undergo arbitrary changes and that each router can obtain the network map. Network topology is the physical and logical relationship of nodes in a network. It represents the schematic arrangement of the links and nodes [76]. A network map is a visualization of devices on a network and is used to determine the bottlenecks in network systems [77], [78]. D2R leverages programmable switches to update the customized packet headers. Packets following this approach have a dedicated header that stores the list of the failed links they encountered through their traversal from the sender to the receiver. The routers on the path update their network maps based on this list.

The workflow of D2R is as follows: the sender's router subtracts the set of failed links from its view of the topology and computes the path to the destination. If no valid path is found, the packet is dropped. If the router is able to configure a

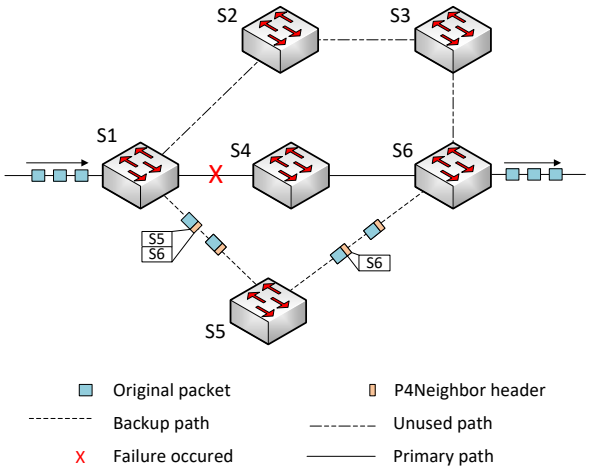


Fig. 9. Workflow of P4Neighbor. After detecting link failure, S1 appends a custom header containing the backup path to the original packets. S5 uses the custom header to forward the packets. S6 uses its routing table to forward the packets because the custom header is empty.

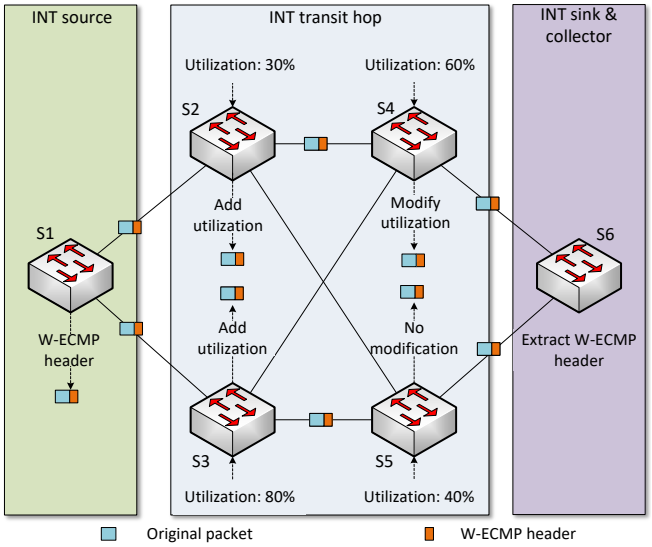


Fig. 10. Feedback phase of W-ECMP. Because the utilization of S4 is larger than S2, packets traversing S2-S4 route report the utilization of S4 to S6 as the utilization of path S1-S2-S4-S6. Packets traversing S3-S5 report the utilization of S3 as the utilization of path S1-S3-S5-S6.

valid path, the next-hop is checked to see if it is up. In case the next-hop is up, the router forwards the packet toward the next-hop. If the next-hop link is a failed one, the list of the failed links is updated to include the newly discovered failed node. After that, the process is repeated, i.e., the sender subtracts the updated list of failed links from its topology, computes a new path, and so on.

P4Neighbor [79] uses custom headers to store the backup path, aiming to reduce the storage overhead of the switch. When a packet faces link failure, this approach leverages the programmable switch to store the backup path inside a custom header. The switch then forwards the packet through the backup path. Subsequent switches are able to adapt to the new path by extracting the information stored inside the custom header of the arriving packet. Following this approach, only one switch entry for a backup path is needed.

The workflow of P4Neighbor is depicted in Fig. 9. The primary route from S1 to S6 is S1-S4-S6. When Link S1-S4 fails, S1 adds a P4Neighbor header to the packets that are routed through S1-S4. P4Neighbor header contains the ID of the switches on the backup path. The switch then forwards the packets to S5. S5 removes its ID from the header and uses the P4Neighbor header to forward the packets to S6. When the packets reach S6, the switch removes its ID from the header and checks the ID of the next switch. Because the header is empty, S6 removes the custom header and forwards the packets based on its routing table.

Weighted equal-cost multi-path (W-ECMP) [54] is a programmable approach that defines a new 32-bits long header and attaches it to the arriving packets. Leaf switches, i.e., switches used to aggregate traffic from servers and connect them to the core network, use the W-ECMP header to share the utilization of the links connecting them. Leaf switches translate utilization to weight and reroute the traffic, at flowlets granularity, according to the current set of weights. Flowlets are chunks of the same flow that follow different paths. A new flowlet is created when the switch starts to forward the packets of an ongoing flow through an alternative path. The switch reroutes the traffic through an alternative path if an increase in the inter-arrival time of packets is detected. In the W-ECMP header, the first 8 bits are used to identify the ID of the source leaf; the second 8 bits specify the route packets should follow in order to reach the destination leaf node; the third 8 bits hold the fingerprint of the switches along the path (Path_ID field); the last 8 bits hold the link utilization (max_utilization field).

W-ECMP is a special application of INT. The feedback phase of W-ECMP is depicted in Fig 10. S1 is the INT source and S6 is the INT sink and collector. There are four paths in the topology. Path 1 is S1-S2-S4-S6; path 2 is S1-S3-S5-S6; path 3 is S1-S3-S4-S6; and path 4 is S1-S2-S5-S6. Switch S1 is sending traffic to switch S6 through paths 1 and 2. For a packet traversing path 1, S2 first stores its ID in the Path_ID field and the port utilization in the max_utilization field of the W-ECMP header. S4 then overrides the max_utilization filed by its port utilization because it is higher. For a packet traversing path 2, S3 first stores its ID in the Switch_ID field and the port utilization in the max_utilization field of the W-ECMP header. S5 does not override the max_utilization filed by its port utilization because it is lower. Switch S6 detects the utilization of path 1 and path 2 by processing the W-ECMP headers of packets traversing the paths.

**Adjusting Headers Fields.** In SHELL [80], the load balancer depends on the client to route the traffic. The client should include the index of the application instance serving it. This index is defined at the time of connection establishment and is included in all packets subsequent to the connection request. The aforementioned special signal, or the application

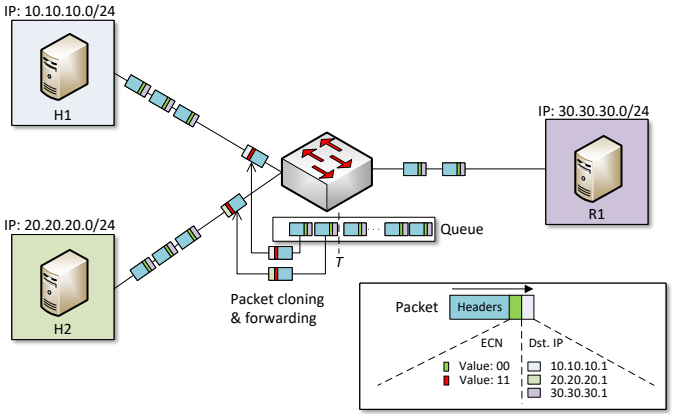


Fig. 11. Workflow of P4QCN. The programmable switch uses the ECN header to notify the senders about the congestion when the queue occupancy exceeds threshold *T*.

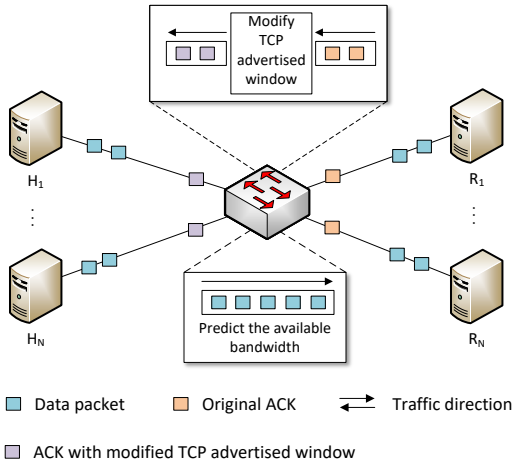


Fig. 12. Workflow of AWW. The programmable switch calculates the available bandwidth and then distributes the bandwidth on the flows by modifying the TCP advertised window in the ACK packets.

instance index, is stored in the low-order bits of the TCP timestamp. The P4 load balancer inspects the TCP timestamp sent by clients and directs the packets to their application instances.

**Network-Assisted Congestion Feedback.** P4QCN [39] utilizes programmable switches to extend the Quantized Congestion Notification (QCN) protocol to IP-routed networks [81]. QCN is a congestion control mechanism that operates at the Ethernet layer [82]. In QCN, a layer 2 switch monitors the queue capacity and sends congestion feedback to end hosts upon detecting congestion. QCN monitors the traffic on per-flow basis, where each flow is characterized by its source/destination MAC address. P4QCN deploys QCN on programmable switches, allowing QCN to operate on IP-routed networks. In P4QCN, programmable switches monitor the queue occupancy to predict congestion. The probability of congestion increases with queue occupancy. When the queue occupancy reaches a predefined threshold, a programmable switch clones the incoming packets, modifies their ECN [16] field to 11, sets their destination IP address to be the source IP address, and forwards them. This technique allows the early detection of congestion. In this approach, the sender interprets the packet with the ECN field equals to 11 as congestion feedback.

The workflow of P4QCN is depicted in Fig. 11. Hosts H1 and H2 are sending packets to the receiver R1. The three devices are in three different network domains. The queue occupancy at the programmable switch reaches the predefined threshold *T*. The switch then clones the incoming packet, modifies the ECN field of the cloned packets, forwards the original packets to R1, and forwards the cloned packet to their senders (i.e., H1 and H2).

AAW [17] is a congestion control mechanism that utilizes the programmable switches to explicitly notify end hosts about the available bandwidth. This approach is an application of INT, where programmable switches monitor the traffic and predict the available bandwidth in the network. Programmable switches estimate the available bandwidth and distribute it on the flows being forwarded through them. The switches distribute the available bandwidth by notifying each end host of the bandwidth available to him. To notify end hosts, the programmable switches modify the advertised window field of the ACK packets. End hosts use the advertised window of the ACK packet to adjust the sending rate. The workflow of AWW is depicted in Fig. 12. The programmable switch predicts the available bandwidth by monitoring the traffic. The switch then modifies the advertised window field of the ACK packets to distribute the predicted bandwidth on the flows.

Feldmann *et al*. [83] propose a P4-enabled network-assisted congestion feedback approach that uses NACK packets to notify end hosts about the congestion. In this approach, programmable switches notify the senders of elephant flows only [84]. There are two types of flow, mice, and elephants. Mice flows dominate by their numbers, while elephants dominate by their traffic volume [85], [86]. Switches are programmed to hold mice flows, elephant flows, and control flows in three different queues. When the queue occupancy (of the queue storing elephant flows' packets) reaches a specific limit, the switches notify only the elephant flows. To notify elephants flows, programmable switches keep track of the *top-k* elephant flows (i.e., elephant flows that are contributing the most to byte transferred). The switches then trim the payload of packets and forge NACKs. The NACKs are sent to the senders of the *top-k* flows.

### *5.4 Feedback Approaches: Comparison, Discussions, and Limitations*

Table III summarizes and compares the aforementioned feedback approaches implementations in programmable data planes. [5] infers congestion based on real-time measurements collected by the data plane. The larger the number of packets to detect congestion, the more accurate the mechanism is at the cost of a slower detection time. Blink [18] analyzes

TABLE III
INTERNAL COMPARISON BETWEEN FEEDBACK IMPLEMENTATIONS IN PROGRAMMABLE SWITCHES.

| Paper | Feedback Type | | | Network Overhead | Fault Detection Speed | Accuracy | Programmable Switch Contributions |
|---|---|---|---|---|---|---|---|
| | Traffic Analysis | Probing | Header Modification | | | | |
| [5] | ✓ | | | Low | Near line-rate* | Depends on the sampling frequency | Analyze queuing and processing delays from the data plane at line-rate |
| Blink [18] | ✓ | | | Low | Within the first RTT | Medium | Detect packets' retransmissions at line-rate |
| HULA [19], MP-HULA [87], DASH [20], Contra [72] | | ✓ | | Medium | Depends on the probing frequency | High | Process and update customized probes at line-rate |
| SPIDER [23] | | ✓ | | Low | Depends on the heartbeat interval | High | Implement stateful data plane packet processing behavior; packet duplication |
| SQR [24] | | ✓ | | Low | Arround 30 microseconds | High | Packet recirculation |
| D2R [55] | | | ✓ | Low | Near line-rate | High | Line-rate access to the customized packet meta-data; deploy search algorithm at the data plane level |
| P4Neighbor [79] | | | ✓ | Low | Near line-rate | High | Store forwarding rules on the packet headers |
| W-ECMP [54] | | | ✓ | Medium | Depends on the traffic | Medium | Import and manipulate custom header fields at line-rate |
| SHELL [80] | | | ✓ | Low | Near line-rate | High | Modify TCP timestamp to include load balancing parameters |
| AWW [17] | | | ✓ | Low | Near line-rate | Depends on the traffic | Per-packet monitoring; estimate bandwidth; header modification |
| [83] | | | ✓ | Low | Depends on the used threshold | Depends on the used threshold | Stateful data elements; header modification |
| P4QCN [39] | | | ✓ | Low | Depends on the used threshold | Depends on the used threshold | Per-packet monitoring; packet recirculation; header modification |

* In some approaches, fault detection happens at near line-rate because either multiple packets are considered by the switch to increase the accuracy of the implemented application or feedback signals upon remote failures have to pass by multiple nodes before informing the programmable switch.

the traffic and uses the data plane signals to detect link failures by leveraging the line-rate visibility provided by the programmable data planes. This approach can detect remote link failures within the first retransmission round; however, it might have some false positives. This problem can be mitigated by increasing the set of monitored flows, but this increases the memory overhead.

Some of the challenges that arise when dealing with data plane signals are due to P4 programmable language specifications [88], [89], and hardware constraints. One of the challenges is that it is impossible to monitor all the flows because of the memory constraints [18], and often no useful data can be gathered from tracking randomly sampled flows. The adopted solution to this problem is to monitor useful flows only. The usefulness of a flow depends on the approach in use. Blink [18] monitors flows that are sending traffic continuously, while the detection mechanism in [5] monitors low-latency flows. The former uses a flow selector that automatically replaces inactive flows with active ones. The latter uses a central controller to decide on the latency threshold of the flows.

Another challenge is to address the reason behind packet loss. In particular, approaches that use packet loss as a notification mechanism face some trouble indicating the evidence behind this loss. For instance, link failure and congestion are two different problems that result in losing packets. Blink [18] can distinguish packet loss caused by failure by focusing on timeout-induced retransmissions and leveraging the fact that failures affect many flows simultaneously. The work in [5] can distinguish packet loss caused by congestion through monitoring fine-grained measurements at the level of the processed data.

The third problem is that unplanned rerouting might lead to forwarding issues, especially in the case of Blink, where the root cause cannot be inferred from the signals provided by the data plane. Blink [18] partially solves this problem by making the backup selection data-driven, i.e., by tracking whether flows resume after rerouting them. [5] addresses this issue by proactive rerouting where an alternative route is preinstalled in the routing tables.

HULA [19], MP-HULA [87], DASH [20], and Contra [72] use probes to infer the state of the network. By leveraging programmable data planes, these approaches calculate the path utilization and modify the probes at each receiving node. Avoiding transient loops and assuring path exploration are common challenges among periodic probing mechanisms.

Some approaches overcome these challenges by constraining their range of operation on tree-based topology [19], [20]. Each switch in a tree-based topology has a defined set of downstream and upstream neighbors; packets are received from the downstream neighbors and sent to the upstream ones. In tree-based topologies, loops are prevented by construction, and path exploration is guaranteed. Contra [72] finds an alternative solution rather than bounding itself to a specific topology. Inspired by [90], [91], Contra assigns version numbers to probes and uses flowlet switching [92] to direct traffic to certain paths. Programmable switches discard probes with an old version number.

Another challenge arises when Multipath TCP is in use [87]. The importance of Multipath TCP or MPTCP has been illustrated in many projects [93]–[96]. MPTCP is a multiple connectivity mechanism that leverages multiple paths by aggregating their capacities and providing seamless failover [97]. According to MP-HULA [87], approaches that do not leverage MPTCP features, like HULA [19] and CONGA [25], yield low performance. To overcome this limitation, MP-HULA adjusts HULA to keep track of the best-k next hops and divides MPCTP subflows accordingly. DASH [20] assigns weight to the egress ports of the programmable switch. It then divides new flows across multiple paths by considering the weight assigned to each path. DASH dynamically adjusts to load changes by modifying the assigned weights.

Besides leveraging the header manipulation property provided by the programmable switches, SPIDER [23] uses the packet duplication feature to check if the failure problem is solved. SPIDER adds a low overhead on the network, and it has an accurate detection mechanism. However, compared to other approaches that use periodic probing mechanisms, SPIDER might take longer a time to detect failures.

D2R [55] includes the list of failed links in the packet headers. It uses the stored topology and the list of failed links received from the modified packet headers to start the traffic rerouting phase. This approach has a low overhead on the network, and the rerouting signal is relatively accurate. P4Neighbor leverages the programmable switches to install the reroute path inside a custom packet header. This approach has a low overhead over the network and in most cases it is reliable.

W-ECMP [54] defines a new header and, adds it to the ongoing traffic. This header keeps track of links' utilization, and it is used by the receiving nodes to alter the forwarding entries. W-ECMP might have a considerable overhead on the network. In W-ECMP, it is not guaranteed to collect the utilization of all the links in a network which might affect the accuracy of this approach. SHELL [80] leverages the programmable switch to replace some bits of the TCP timestamp with the index of the application instance. The index is required by the load balancer to perform load balancing. This approach almost has no overhead on the network as it adds no additional loads to the packets.

AWW [17] modifies the advertised window of the flows to avoid congestion. This approach is not scalable, as the programmable switches cannot store information about all flows in large networks. Moreover, AWW modifies the advertised window of mice flows also. However, mice flows are not subjective to congestion [83], and thus, they should not be part of the congestion avoidance mechanism. Feldmann *et al*. [83] resolved the scalability issue by monitoring elephant flows only. The authors propose using NACKs generated by programmable switches to notify senders about the congestion. This approach notifies the senders of the elephant flows only; however, the approach does not guarantee that the notification will be received by the senders.

Similar to AWW, P4QCN [39] includes mice flows in the deployed congestion avoidance mechanism. P4QCN predicts congestion by monitoring queue occupancy. In P4QCN, if the detection threshold is too low, frequent ECN packets might be sent by the switches to the end hosts. Thus, underutilizing the available bandwidth. On the other hand, a high threshold might not allow P4QCN to detect congestion in its early stage. A main limitation of P4QCN is that the approach requires modifying the end hosts to understand the congestion feedback.

### *5.5 Feedback Approaches: Comparison with Legacy*

Table IV shows a comparison between the feedback mechanisms in programmable and traditional approaches. Traditional mechanisms detect remote failures based on the propagation of BGP messages. The same mechanisms use either periodic probing or periodic heartbeats to detect local failures. The most deployed legacy load balancing approach does not consider the state of the network while splitting the traffic, and so it deploys no feedback mechanism.

Moreover, traditional congestion control mechanisms utilize packet loss as an indicator of congestion. To detect congestion before experiencing packet loss, congestion signaling mechanisms were proposed. Congestion signaling mechanisms (e.g., DCTCP [98], pFabric [99], PCC [100], QJUMP [101], NDP [102], Copa [103], etc.) are used to avoid congestion. These approaches usually depend on the receiver to notify the sender about the congestion, which might not be scalable if the RTT is significant. Finally, traditional layer-4 load balancing approaches either deploy an SLB to maintain PCC or use ECMP, which does not consider PCC.

Programmable data plane-based approaches use data plane signals (e.g., TCP retransmissions) as an indicator of a remote failure. The same approaches deploy both periodic probing and periodic heartbeats concurrently to detect local failures. Periodic probing is also used to propagate information about the utilization of the links and to report a failed link/node in a network. The same approaches detect congestion by monitoring packet meta-data at line-rate. Besides, programmable switches can provide senders with feedback before congestion occurs without involving the receivers. Thus, reducing the time needed for the feedback to reach the senders. Finally, programmable approaches leverage packet meta-data to reflect any updates in DIP pools.

TABLE IV
COMPARISON BETWEEN FEEDBACK MECHANISMS IN PROGRAMMABLE AND COMMON LEGACY IMPLEMENTATIONS.

| Problem | Programmable Approaches | Traditional Approaches |
|---|---|---|
| Remote Failures | Data plane signals | Routing messages |
| Local Failures | Combination of periodic and heartbeats | Periodic probing; periodic heartbeats |
| Load Imbalance | Periodic probing | Random splitting |
| Congestion | Line-rate monitoring of packets meta-data; periodic probing; network-assisted feedback | Packet loss; congestion signaling mechanisms |
| DIP Changes | Use programmable switches to monitor flows at line-rate | Use software to monitor flows; do not consider PCC |

## 6. TRAFFIC REROUTING PHASE

### *6.1 Reactive Rerouting*

In reactive approaches, and after receiving the rerouting signal, the programmable switch computes an alternative path either to avoid faulty links or to perform load balancing. After choosing the alternative path, the switch updates its forwarding strategy accordingly.

**Data Plane Resilience.** In D2R [55], even if a link failure occurs, the packets will be transmitted between the source-destination pair as long as a path exists. This approach assures that policies always hold, even under failures. To achieve that, D2R implements graph traversal algorithms inside the data plane. The traversal algorithms take as input the switch view of the network topology and the list of the failed links.

Whenever a failure is detected, the deployed routing algorithm is triggered, and the alternative path is configured at near line-rate. D2R is able to achieve near line-rate performance while computing a policy-compliant valid path by leveraging the computation power, parallelism, and packet recirculation features provided by the programmable switches. D2R architecture is divided into three planes: the policy plane, the control plane, and the data plane.

The policy plane is used to specify the policy requirements for each flow. This plane operates as a centralized controller. The control plane is waferthin [104], which programs the data plane according to the rules provided by the policy plane. The data plane can compute, given the FCP header, an active path by running a graph traversal algorithm without the interference of the control plane. The traversal graph algorithms used by D2R are modified versions of Breadth-First Search and Depth-First Search algorithms.

Breadth-First Search is modified because P4 supports only stack structure. The traditional Breadth-First Search is implemented using a first-in-first-out queue. Further, Depth-First Search might yield a very long path compared to Breadth-First Search; thus, D2R implements a new variant called Iterative Deepening Depth-First Search, where the length of the discovered paths can be bound.

Chiesa *et al*. propose PURR [105], a new Fast Rerouting (FRR) primitive which is used as a building block implementation for any FRR mechanism regardless of whether it acts over single or multiple link failures [1], [106]. PURR's primary goal is to support the fast transition to the alternative active port by avoiding packet recirculation. In FRR approaches, each packet is assigned to multiple output ports and forwarded through the first active one.

For example, assume that any packet can be forwarded through any egress port and that a switch failed to send a packet from port *J*. The switch tries ports *J+1, J+2,..., J+n* until it finds an active port or it tries all the indices. If the number of ports is five and the primary output port is 3, then the order of ports that are tried starting from port 3 is <3,4,5,1,2>. If the primary output port is 2, then the order of the ports that are tried starting from port 2 is <2,3,4,5,1>. The resulting orderings are called circular FRR sequences. Each circular FRR sequence can be obtained from any other sequence by performing *L* shifts knowing that *L* is less than the number of ports.

PURR shows that recirculating the packet while searching for the first active port degrades the flow completion time and wastes hardware resources. It proposes a parallel search to overcome these problems. In particular, PURR leverages the programmable switch to include a sequence of if-else statements inside the TCAM memory that searches for the active port in parallel. This allows for the identification of the first active port in one shot.

Ye *et al*. propose W-ECMP [54], a weighted ECMP load balancing scheme for data centers using a P4 switch. This approach makes ECMP congestion aware by increasing the probability of choosing low-utilized paths. Path utilization is calculated by comparing the amount of traffic a path is forwarding with its maximum forwarding capacity. Because a single path between two communicating devices might incorporate multiple links, the path utilization is set to be the utilization of the most congested link. W-ECMP defines a new header to track the link with maximum utilization in a given path. The W-ECMP header contains a Switch_ID field and max_utilization field. When a programmable switch receives a packet containing the W-ECMP header, it first stores its ID on the header and then compares the utilization of its port to the max_utilization value stored in the W-ECMP header. The switch overrides the max_utilization value stored in the header if the utilization of its port is larger. When this packet is received by the destination switch (i.e., the last programmable switch the packet will pass through before reaching the final destination), the switch extracts the IDs of the switches the packet passed through to identify the path used by the packet. The switch then uses the utilization value from the W-ECMP header to assign a weight to the path. The probability to choose a path to forward packets is equal to the weight of the path divided by the accumulation of the weights of all the paths.

The traffic rerouting phase of W-ECMP is depicted in Fig 13. S1 is the source switch (INT source), and S6 is the destination switch (INT sink and INT collector). There

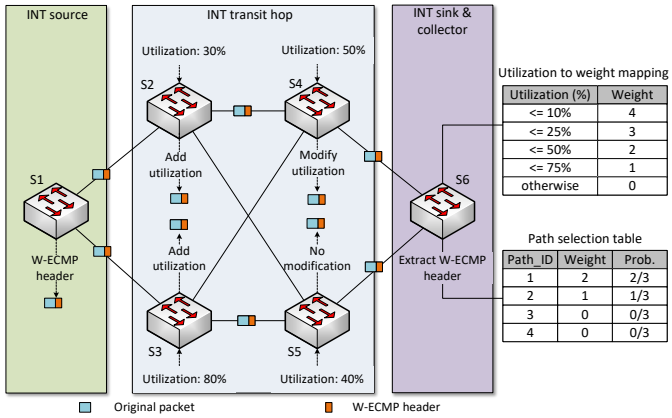


Fig. 13. Traffic rerouting phase of W-ECMP. The probability of choosing a path is equivalent to its weight divided by the accumulated weights of all paths. Path S1-S2-S4-S6 has ID 1; path S1-S3-S5-S6 has ID 2; path S1-S2-S5-S6 has ID 3; and path S1-S3-S4-S6 has ID 4.

are four paths in the topology. Path 1 is S1-S2-S4-S6; path 2 is S1-S3-S5-S6; path 3 is S1-S2-S5-S6; and path 4 is S1-S3-S4-S6. The W-ECMP headers of packets traversing path 1 notify S6 that the utilization of the path is 50%. S6 is also notified about the utilization of path 2 (80%) by the packets traversing that path. S6 then uses the *utilization to weight* table to assign weights to paths. Note that paths 3 and 4 have no weights because S6 did not receive any packets that traversed those paths. To calculate the probability of using each path, S6 divides the weight of a path by the summation of all the weights. The summation of all the weights in the *path selection* table is 3. The probability of choosing path 1 is 2/3. The probability of using path 2 is 1/3. As the utilization of a path decreases, its weight increases, and consequently, the probability of choosing the path for future transmissions increases.

**Load Balancing and Congestion Control.** Katta *et al*. propose a hop-by-hop utilization-aware load balancing architecture (HULA). In HULA [19], new paths are configured according to the load of the probes. Each programmable switch has dedicated tables populated and updated to keep track of the least utilized path. Usually, there are multiple paths that connect any two nodes in a network. The least utilized path represents the path with the maximum available bandwidth. In HULA, the receiving switch leverages the data encapsulated inside the probes to infer the best next-hop toward any destination and updates its tables accordingly. The best next-hope is the one that minimizes the maximum utilization of all links along the path.

After updating its tables, the receiving switch forwards the probes to other switches. HULA probe is a minimum-sized packet of 64 bytes. This probe contains a customized HULA header. The header consists of the ID of the ToR that generated the probe and the utilization of the best path. Each programmable switch estimates the link utilization per port according to an exponential moving average generator.

When a new probe enters the switch, new path utilization is computed at line-rate, and the result is compared with the stored path utilization. The tables are updated to reflect the new value if the newly calculated value is smaller than the stored one. After that, the probe's header is updated, and the probe is forwarded to other switches. Probes propagate as discussed in the previous section.

By leveraging the programmable switch, HULA can store the arriving time of the last arrived packet of each single flow. HULA divides flows into flowlets [107], where a flowlet is detected by the switch when the time interval between two consecutive packets is greater than a predefined threshold. This threshold should be big enough to avoid packet reordering. If a new flowlet is detected, the switch forwards it along the best-next hop, which might or might not be the same next-hop the previous flowlet has followed.

Benet *et al*. propose MP-HULA [87], an extension for HULA that supports MPTCP. The key difference here is that the tables added by MP-HULA hold the congestion state of the best-k next-hops instead of only holding the state of the best next-hop. Multipath HULA or MP-HULA switch parses the MPTCP headers to be able to associate MPTCP subflows to an MPTCP connection.

After obtaining the subflow information, the switch uses the congestion state of the best-k next-hops to reroute the traffic. MP-HULA balances the load at the granularity of flowlets, where it assures that no two MPTCP sub-flows are transmitting data through the same path. It leverages the programmable switch to keep track of the flowlet of each subflow. In particular, when a new flowlet is established, the switch checks if any other flowlet that belongs to the same MPTCP connection is assigned to the best available path. If this is the case, the switch directs the newly established flowlet to the first unused best path. In this way, the switch avoids sending multiple sub-flows of the same MPTCP connection to the bottleneck link while performing load balancing.

The switch manipulates the forwarding action by defining new tables. Flowlet table stores the needed information about each flowlet, i.e., the flowlet ID, its destination, its timestamping, the ID of the MPTCP connection, the number of subflows, and the best-next hop. Best k-hops table stores the utilization of the best-k next-hops. MPTCP subflow mapping table stores which subflow is forwarding throw which next-hop.

Hsu *et al*. propose DASH [20], a data plane adaptive splitting with hash threshold (DASH). DASH partitions the hash space into unique regions and assigns them to different routing paths. The hash space is the set that contains all the hash values of a hash function [108]. A hash function is a mathematical formula that takes data as an input and produces an output known as a hash [109]. DASH balances the load by assigning weights to paths according to their utilization and divides the hashing space among paths according to the assigned weights. If the links' utilization in the network change, only a small number of region boundaries changes are required.

DASH can adapt to the traffic conditions and yield efficient

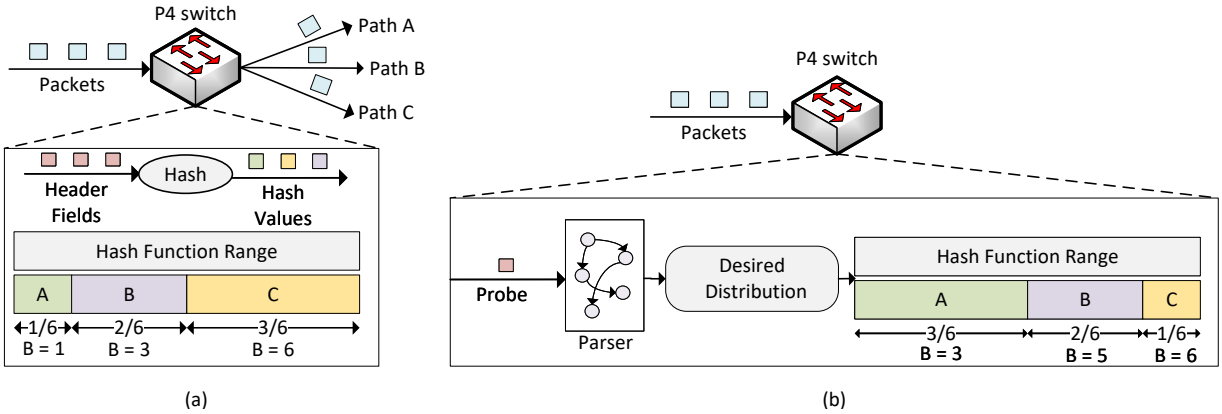


Fig. 14. Workflow of DASH. (a) The hash value of the arriving packets headers is checked against the boundaries. If the calculated value is less than the boundary of path A, the corresponding packet is forwarded along this path. Otherwise, the hash value is compared against the other boundaries, in an ascending order, and will be forwarded accordingly. (b) The distribution is adjusted based on the weights carried by the arriving probe.

utilization of the network capacity by leveraging a multi-stage pipeline, per-stage stateful ALUs, and adjusting the distribution at line-rate. When a packet arrives, the receiving switch calculates the hash value of its headers and infers the associated path by comparing the obtained hash value to the region boundaries.

Consider Fig. 14(a). There are three paths, *A, B,* and *C*. The current distribution is 1:2:3, which means that path *A* has 1/6 of the total hashing space, path *B* has 1/3 of the total hashing space, and path *C* has 1/2 of the total hashing space. When a packet arrives, its hash value is compared against region A's boundary. It is forwarded through the path associated with A if its value is smaller than the boundary. Otherwise, the hash value of the packet is compared sequentially against the remaining paths regions' boundaries and forwarded accordingly.

The path's boundary is stored inside register memory. The boundary comparison and packet hash value are made using a stateful ALU (SALU). Consider Fig. 14(b). The new distribution of weights is calculated based on the last received probe within a certain period. The boundaries are updated sequentially by adding the cumulative sum of previous path boundaries with the current path region size. For instance, path A boundary is equal to the path region size only; path B boundary is equal to the sum of path A boundary and its region size.

Pit-Claudel *et al*. propose SHELL [80], a stateless load-aware balancing in P4. In the data center and cloud architectures, applications are replicated over multiple application instances. All the application instances that share the same services have the same VIP. This VIP is advertised to the Internet. Each instance is capable of serving some incoming queries independently [110]–[112].

SHELL aims at forwarding new connections to a set of application instances that can decide locally to accept the connection. Each connection should be accepted by one instance only. The subsequent packets of the accepted connection should be marked such that the load balancer can know to which application instance to forward them without maintaining a per-connection-state. In particular, the control plane of SHELL holds a consistent table and a choice history table. The first table is used to direct the new connection request packets identified by their 5-tuple (IP source, IP destination, source port, destination port, transport protocol) hashing to a set of candidate application instances proposed by the control plane. These packets are directed through the set of application instances until one instance accepts the connection. Adopting the strategy followed by [113], the last instance in the set always accepts new connections.

The second table is used to direct subsequent packets to the accepted instance. This is done by keeping track of the index *ci* of the application instance that accepted the connection and communicating it back to the client. The client includes the application instance's index in all the subsequent packets. More details about the instance's index are presented in the previous section.

Consistent hashing buckets are used to store hash keys [114]. A hash key is the input of a hash function [115]. In SHELL, the application indices are the hash keys. In case any modification is done to the consistent hashing buckets due to changes in the set of application instances, the P4 load balancer directs the subsequent packets to two different application instances. The first application instance is the one that is currently associated with index *ci*, and the second instance is the one that previously carried *ci*.

### 6.2 *Proactive Rerouting*

In proactive approaches, an alternative path is precomputed and stored inside the routing tables. In most cases, proactive approaches are used to protect the connection against failures by switching the traffic to another path after receiving the rerouting signal. In some cases, the concept of proactive rerouting still exists where the alternative path is precomputed; however, the usage of the alternative path is approach-dependent.

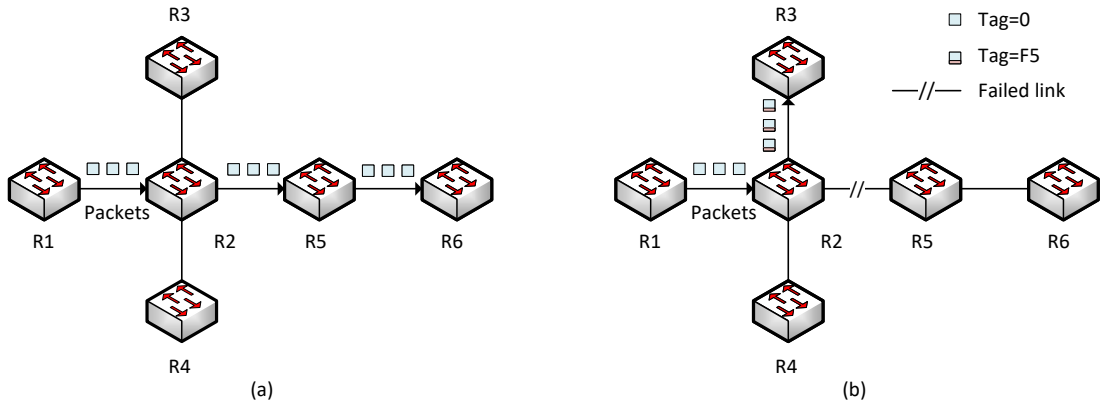


Fig. 15. SPIDER local link failure with respect to R2. (a) Packets are moving from R1 to R6 when there is no link failure. (b) Tag F5 is added to the rerouted packets at R2 after link R2-R5 has failed.

**Alternative Path Rerouting.** Qu *et al*. assume that the network is responsible for recovering the lost packets and propose Shared Queue Ring (SQR) [24]. This mechanism recovers packets that are lost during network convergence after a link failure occurrence. In SQR, the switch saves a copy of the recently sent packets, by leveraging the recirculation feature of the programmable switches, and retransmits them through an alternative path when a link fails; thus, packet loss can be avoided.

A predefined interval decides how recent the packet should be to be cached. This interval is typically an upper bound estimation of the time needed to detect and recover from the link failure. SQR is mainly used to protect the packets that belong to a latency-sensitive flow, i.e., a flow that requires low end-to-end latency. The arriving packets that are not protected by SQR are forwarded to specific egress ports based on the forwarding rules. However, if a packet that needs protection arrives, SQR makes a copy of it and caches this copy. After that, SQR checks if the packet's forwarding egress port is up. If the port is up, SQR infers that the packet is delivered safely, and the cached copy is removed. Otherwise, the packet is copied and cached again. This assures that the packet is safely sent out of the switch.

Holterbach *et al*. propose Blink [18], a fast connectivity recovery system that happens entirely in the data plane. Blink uses data plane signals (TCP retransmissions) to infer link failure. It routes the traffic through a precomputed alternative path. Holterbach *et al*. define four major requirements for fast rerouting using data plane signals: i) rerouting should only be triggered upon major disruption events, and it should be immune to noise and ordinary protocol behavior, ii) failure should be detected within the first retransmission round, iii) detection should be based on the most active flows, iv) backup next-hop should be chosen by the data plane under the condition that connection resumes.

Blink fulfills the first two requirements based on the detection mechanism discussed in the previous section. To achieve requirement three, Blink deploys two levels of filtering. It does not monitor all the forwarding keys (IP prefixes) of the routing table because that adds additional overhead and might affect its efficiency. Instead, Blink focuses on the most popular destination prefixes. Choosing the previously mentioned set of prefixes is a feature of Blink. Most of the Internet traffic is handled by a small set of prefixes due to the skewness of the Internet traffic [116].

The second stage of filtering is at the level of the flows. Blink deploys a Flow Selector that tracks, at line-rate, a subset of active flows for each monitored prefix. The default number of tracked flows is 64. The Selector defines active flows as the ones that send a packet within an interval of time. This time has a default value of 2 seconds. The Selector replaces the tracked flows as soon as they become inactive or after an interval of time which is 8.5 minutes by default. Blink achieves the last requirement by precomputing a set of alternative paths and dividing the flows among them to check their validity. Blink reroutes the traffic through the primary next-hop if all the next-hops are assessed as not working. It then waits for the convergence of the control plane to update the forwarding table.

Xu *et al*. propose P4Neighbor [79], an efficient link failure recovery mechanism based on programmable switches. Each switch in this approach is required to calculate an alternative path to all its neighbors. It then encapsulates the calculated path inside a packet's custom header. Once a link failure is detected, the switch uses the custom header to forward the packet through the precomputed alternative path. After recovering from the failure, the custom header is removed.

Each packet has a one-bit field that represents the current state of the packet. If this field holds value 0, then the packet is considered in the normal state and should be forwarded according to the routing table configuration. However, if the one-bit field holds value 1, then the packet is considered in the link failure recovery state, and the backup path should be installed at the custom header.

Once a new packet arrives, the programmable switch checks the one-bit field to identify the state of the packet. If the state of the packet is normal, the packet is forwarded according to its destination address. The port associated with the destination

path is checked for validity; if valid, the packet is forwarded normally. Otherwise, the state of the packet is changed to recovery, and the backup path is computed and installed inside the custom header. The custom header operates as a stack data structure. In the stack data structure, the values are stored sequentially where the first added value resides at the top of the stack. The second added value is stored directly after the first added one, and so on. After that, the packet is forwarded again to the ingress port, but this time the one-bit field indicates that this packet is in the recovery state.

The custom header is used to choose the alternative path the packet should follow when it is in the recovery state. The switch parses the set of headers and checks the head of the stack to forward the packet. When this packet reaches a new switch, the receiving switch parses it and detects that the packet is in the recovery state. After that, the switch checks the head of the stack to infer the forwarding path. However, if the stack is empty, the switch indicates that the packet has recovered from the failure. The switch then changes the state of the packet to the normal state.

Cascone *et al*. propose SPIDER [23], a fast failure detection and recovery mechanism in SDN with a stateful data plane. SPIDER differentiates between local and remote failures according to the number of nodes between the current node and the failed node. It defines the failure state as *Fi*, such that *i* is the index of the failed element. If the failed node is directly connected to the current node, the situation is defined as a local failover. Otherwise, the situation is defined as a remote failover.

For every possible *Fi*, there should be a backup path that can contain any elements from the primary path other than node *i*. The backup path should have a detour around node *i*. The reason behind this requirement is that SPIDER does not differentiate between link and node failures, and thus the worst-case scenario is always considered (any path to node *i* from its neighbors is a failed one, too). This consideration results in a very short failover delay (< 1ms). When a local failover is detected, the current node tags the incoming packets by the index of the failed node. Assume the index is *i* and that the tag is *Fi*. The node either sends the packet to a backup port if it has a detour around node *i*, or sends them back through their input port. Each receiving switch on the backward primary path back propagates the tagged packets till they reach a node capable of forwarding them through an alternative detour.

Consider Fig. 15. Node *R2* detected a local failure at node *R5*. It tagged the packets with tag *F5* and forwarded the packets through node *R3*. On the other hand, consider Fig. 16, node *R5* detected the failure. It tagged the packets with tag *F6* and checked if it had an alternative path that detours over the failed link. It had no alternative route, so it back propagated the packets to *R2*, which was able to find an alternative path through node *R4*. After that, node *R2* forwarded incoming packets that should go through node *R6* to the alternative path through node *R4*.

Miao *et al*. propose SilkRoad [11], an approach that makes

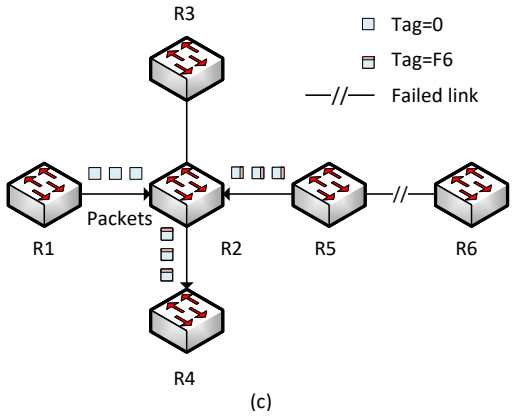


Fig. 16. SPIDER remote link failure with respect to R2.

stateful layer 4 load balancing fast and cheap using programmable switches. SilkRoad assures per-connection consistency and provides a direct path between the sender and the serving application instances by maintaining the connection state at the switch. In particular, the load balancer holds four tables, the VIPTable, the ConnTable, the DIPPoolTable, and the Transmittable.

If no update is occurring, VIPTable is used to match the newly arriving connection against the version of the DIP pool serving it. The results are then pushed to the ConnTable using the LearningTable such that all subsequent packets are forwarded to the right version. The DIPPoolTable maintains the version to DIP pool mapping. In case a DIP pool is under update, the Transmittable keeps track of the newly arriving connections that should be mapped to an old DIP pool version. After the update, all the packets that miss the ConnTable obtain the two DIP pool versions from the VIPTable. For each packet, the two obtained DIP pool versions are checked against the TransitTable. Entries inside the TransitTable use the old DIP pool version, whereas the ones that miss the table use the new DIP pool version.

SilkRoad manages to store millions of connections by minimizing the size of the match field and reducing the size of the action data part of each entry. In particular, this approach does not use the 5-tuple hashing (which is 37 bytes for IPv6 connections) as the match keys. Instead, it uses a compact hash digest (which is 2 bytes) as proposed in [117]. Regarding the action data part, SilkRoad maps each connection to a version of a DIP pool rather than mapping it to the actual pool. This approach updates certain DIP pools by applying the changes to a copy of the original DIP pool, assigning it a new version number, and updating the VIPTable to map new connections to the newly formed DIP pool version.

When a new packet arrives, its hash is checked against the hash digests in the ConnTable. If it is a hit, the packet is forwarded to the DIPPoolTable. Otherwise, the packet is pushed to the VIPTable. If the VIPTable is not under update, it returns the DIP pool version, which is the newest one. The VIPTable then pushes the results to the LearningTable.

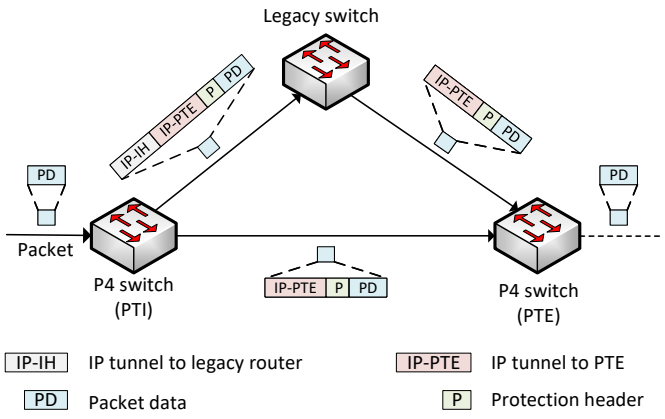


Fig. 17. Workflow of P4-Protect.

After that, the LearningTable updates the ConnTable and forwards the packet to the DIPPoolTable. However, if the VIPTable is under update, the arriving packets are stored inside the TransitTable. After the update, the packets that miss the ConnTable are checked against the TransitTable. The old DIP pool version is used in case these packets hit the Transmittable, and the new version is used if it was a miss. The DIPPoolTable matches the VIP and DIP versions and forwards the traffic to the associated DIP.

**Other Proactive Approaches.** Lindner *et al*. propose P4-Protect [118], a 1+1 path protection for programmable switches. In the 1+1 protection approaches, the traffic is transmitted safely between two nodes by leveraging two different paths. The sending node forwards the same packets through two paths simultaneously. The receiving node drops the duplicates and forwards the traffic. The key insight here is that the two paths are disjoint so that the packets are guaranteed to be delivered even if one of the routes fails.

A protection tunnel is built between two P4 switches. The sending node is denoted by the protection tunnel ingress (PTI) node, and the receiving node is denoted by the protection tunnel egress (PTE) node. The programmer can choose the characteristics of the flows to be protected. When a new flow arrives, the switch checks if the flow should be transmitted in the safe mode or in the normal mode. If this flow is to be transmitted in the normal mode, the switch forwards its subsequent packets according to the forwarding table's rules. However, if the flow is to be transmitted in the safe mode, the packets are equipped with a protection header and a new destination IP address. The protection header is 64 bytes long and contains the connection identifier (CID), the sequence number (SN), and the next protocol in use. At the PTE node, the incoming packets are parsed, and the protected packets are identified. The PTE node uses the CID to identify the protected packets, and then it uses the SN field to filter the duplicates. Finally, the receiving node infers the next protocol to parse based on the next protocol field.

Fig. 17 illustrates the workflow in the P4-Protect approach. One packet arrives at the PTI; it hits the match-action table where two duplicates are forwarded through two different protection tunnels. A protection header and the IP address of the receiving node are encapsulated in the packet headers. The first copy of the packet holds only the IP address of the PTE node, where the packet is forwarded directly to the node (there are no intermediary nodes). However, the second copy has to pass through an intermediary node. Two layers of IP addresses are added to its headers. The intermediary node IP address wraps the PTE node's IP address. The PTE decapsulates the copy that arrives first and drops the late one.

Giesen *et al*. propose Wharf [119], an in-network distributed approach that mitigates faulty links. When a failure is detected, Wharf-enabled switches activate a forward error-correction (FEC) scheme to recover lost frames. In FEC, the transmitter partitions the packets into multiple parts. The transmitter sends each part multiple times. The receiver recognizes only the portion of the data that contains no apparent errors [120]. Wharf has three building blocks: the link monitoring agent, the sending proxy, and the receiving proxy.

The monitoring agent tries to detect link malfunction and inform the connected pair of switches. The sending proxy partitions the arriving packets according to a set of traffic classes. Parity bits are added to the packets that should be sent across a faulty link. Parity bits store the checksum of the packet. By leveraging these bits, the receiving node can check if the received packet is transmitted without being modified by the network. The receiving node (receiving proxy) calculates the checksum of the arriving packet and compares it to the parity bit. The packet is accepted if the calculated checksum and the received parity bits are equal. Otherwise, the received packet is dropped.

Xie *et al*. propose GRED [121], a short-latency and low-overhead data placement and retrieval service for edge computing. This approach shortens the routing path and minimizes the forwarding table size by following a greedy forwarding algorithm. GRED utilizes an SDN controller to maintain a virtual space. The controller associates each switch to a point in the virtual space by computing the Delaunay Triangulation (DT) graph [124]. In DT graphs, data is guaranteed to be received by a destination node if a greedy forwarding approach is applied [125]. After computing the DT graph, the controller installs the forwarding entries on the programmable switches. In GRED, the forwarding entries are assigned based on the distance in the virtual space rather than per-flow information.

Kawaguchi *et al*. [122] design unsplittable flow edge load factor balancing (UELB) in SDN. This approach models the topology as a graph and formulates the problem of determining a balanced load distribution as a UELB problem. The approach then relaxes UELB (which is NP-hard [126]) using linear programming (LP) and uses an LP solver to determine the optimal routes. A controller continuously monitors link utilization. If congestion is detected on one of the links, the controller runs the traffic load balancing algorithm and updates the routing tables of programmable switches.

TABLE V
INTRA COMPARISON BETWEEN REROUTING ALGORITHMS IN PROGRAMMABLE SWITCHES.

| Paper | Rerouting Algorithm | | Problem | | | Forwarding Behavior | Packet Loss | Programmable Switch Contributions |
|---|---|---|---|---|---|---|---|---|
| | Reactive | Proactive | Link/Node Failure | Load Imbalance | Congestion | | | |
| HULA [19] | ✓ | | ✓ | ✓ | ✓ | Modify path utilization table | Medium | Read and write to registers at line-rate |
| MP-HULA [87] | ✓ | | ✓ | ✓ | ✓ | Modify the best k hop tables | Medium | Increment the counters at line-rate |
| DASH [20] | ✓ | | | ✓ | ✓ | Adjust the region boundaries | Medium | Packet recirculation; define new data structure |
| Contra [72] | ✓ | | ✓ | ✓ | ✓ | Modify the entries of the forwarding tables | Medium | Dynamic code that adapts to the implemented policies |
| [5] | | ✓ | | | ✓ | Switching to a preinstalled forwarding entry | Low | Provide a congestion control program deployed within the switch |
| Blink [18] | | ✓ | ✓ | | | Use a precomputed alternative path | Low | Choose the alternative path at the data plane level |
| SPIDER [23] | | ✓ | ✓ | | | Use the data stored inside the register arrays | Low | Modify packet meta-data at line-rate; maintain per-flow state at line-rate |
| D2R [55] | ✓ | | ✓ | | | Modify the iddfs table contents | Almost none | Path recomputation at the data plane level; provide complex packet processing pipelines at line-rate; packet recirculation |
| P4-Neighbor [79] | | ✓ | ✓ | | | Include the alternative path in the packet headers | Almost none | Packet header modification at line-rate; provide complex packet processing pipelines at line-rate |
| SQR [24] | ✓ | | ✓ | | | Alternative port decided by the control plane | Almost none | Packet recirculation; modify packet meta-data and have fine-grained time measuring capability |
| PURR [105] | ✓ | | ✓ | | | Modify the egress port | Low | Active-port search in parallel; packet recirculation |
| Wharf [119] | ✓ | | ✓ | | | Encapsulate protection headers | Medium | Customized header processing and manipulation |
| P4-Protect [118] | | ✓ | ✓ | | | Follow the IP addresses encapsulated in the packet headers | Almost none | Per-flow visibility at line-rate; header modification at line-rate |
| SilkRoad [11] | | ✓ | | ✓ | | Define packet meta-data fields to carry changes | Almost none | Modify packet meta-data at line-rate; maintain per-flow state at line-rate |
| W-ECMP [54] | ✓ | | | ✓ | | Update the weighted utilization table | Medium | Line-rate access to the arriving packets' timestamps |
| GRED [121] | | ✓ | | ✓ | | Modify the entries of the forwarding tables | Almost None | Match on next hop with minimum distance |
| UELB [122] | | ✓ | | ✓ | | Modify the entries of the forwarding tables | Almost None | Use *P4-runtime to update forwarding table |

* P4-runtime is an API used by the control plane to interact with the data plane at run time [123].

### *6.3 Rerouting Algorithms: Comparison, Discussions, and Limitations*

Table V summarizes and compares the aforementioned programmable data plane-based algorithms. HULA [19], MP-HULA [87], and Contra [72] can deal with link failures, load imbalance, and congestion simultaneously. DASH [20] is limited to load imbalance and congestion. MP-HULA, Contra, and DASH avoid congesting the primary link by spreading the traffic among multiple paths, while HULA directs the traffic along only one path at any given time. The operator in HULA, MP-HULA, and DASH can adjust the probing system such that these probes can propagate different network metrics (e.g., queue occupancy, queuing delay, processing delay). In Contra, the operator has the flexibility to change the whole routing policy. A routing policy is a set of instructions provided by the operator and used by the routers to populate the routing tables.

Blink [18], SPIDER [23], and D2R [55] can deal with both remote and local link failures. SPIDER and D2R consider the root cause of the failure, i.e., the address of failed node/link.

Blink can not identify the root cause of the failure. Blink and SPIDER are proactive rerouting algorithms that highly benefit from the line-rate visibility provided by the programmable switches. D2R implements a reactive rerouting algorithm and makes use of the complex packet processing pipelines feature in the programmable switches.

SPIDER [23], D2R [55], and P4Neighbor [79] use the header modification capability of the programmable switches to reroute the packets. SPIDER and D2R include information about the failed links in the packet header. SPIDER encapsulates the tag of the failed node, while D2R encapsulates the list of the failed links. On the other hand, P4Neighbor includes the alternative path inside the packet headers.

SQR [24] and PURR [105] are two primitives used to enhance the performance of FRR approaches. SQR's main goal is to prevent packet loss after link failure occurrences. It does so by caching the packets of the protected flows using the packet recirculation feature of the programmable switches. PURR's main goal is to find the first active port in a programmable switch. It does so by leveraging the TCAM memory to perform a parallel search. The same algorithm makes use of the recirculation feature of the programmable switches.

Wharf [119] deploys a reactive mechanism to mitigate faulty links by sending packets encapsulated with parity bits. P4-Protect [118] is a proactive rerouting mechanism that protects specific flows by forwarding their traffic through two independent paths simultaneously. Wharf makes use of the flexible header processing and manipulation feature of the programmable switches. P4-Protect makes use of the per-flow visibility and pack header customization features provided by the programmable switches.

Silkroad [11] is a proactive stateful layer 4 load balancing mechanism. This approach leverages the programmable switches to maintain a per-flow connection state. W-ECMP [54] is a reactive load balancing mechanism. It uses programmable switches to include its custom header in the arriving packets. W-ECMP leverages the switches to update the load of the probes.

GRED [121] and UELB [122] are proactive load balancing approaches that rely on the control plane to calculate and maintain the forwarding rules. In GRED, the control plane is required to re-calculate the DT graph when new nodes are added or removed from the topology. Similar to GRED, the control plane of UELB models the topology as a graph and calculated the routes that minimize the most utilized paths. The two approaches do not adapt to changes in the topology without the intervention of the control plane.

### *6.4 Rerouting Algorithms: Comparison with Legacy*

Table VI compares rerouting algorithms in programmable and legacy approaches. Traditional approaches wait for the convergence of the control plane upon remote link failures. For local link failures, traditional approaches use a preinstalled alternative path computed offline by the control plane. The most deployed traditional rerouting algorithm splits the traffic randomly based on a hash function without considering the links' utilization. For congestion control, traditional approaches do not reroute the traffic through alternative routes. They adjust the sending rate at the sender. Finally, Upon DIP pool update, traditional approaches either deploy thousands of SLBs to keep the PCC or deploy ECMP, which does not consider the PCC while balancing the traffic.

TABLE VI
COMPARISON BETWEEN REROUTING ALGORITHMS IN PROGRAMMABLE AND COMMON LEGACY IMPLEMENTATIONS.

| Problem | Programmable Approaches | Traditional Approaches |
|---|---|---|
| Remote Failures | Use preinstalled routes; compute new path at near line-rate | Wait for the convergence of the control plane |
| Local Failures | Use packet metadata to reroute the traffic at line-rate | Use preinstalled paths |
| Load Imbalance | Modify registers values at line-rate | Random forwarding |
| Congestion | Reroute the traffic | Modify sending rate |
| DIP Changes | Modify registers values at line-rate | Use software to modify table entries; random forwarding |

Programmable data plane-based rerouting algorithms either use a preinstalled path or compute a new path at near line-rate to avoid remote failed links. For local link failures, programmable algorithms use packet meta-data and preinstalled alternative paths to reroute the traffic. Load balancers in programmable switches can reroute the traffic at line-rate by modifying the values stored inside the register arrays. To control congestion, programmable approaches divide the nodes in a network into multiple groups. In each group, the alternative paths are installed in one node. Other nodes within the same group may not be capable of rerouting the traffic. Their sole role is to inform their predecessors when congestion is detected. Finally, programmable algorithms maintain the PCC while balancing the traffic.

## 7. PERFORMANCE COMPARISON

Many approaches were proposed to solve link failures, congestion, and load imbalance problems long before the emergence of the programmable data planes. This section discusses how different programmable data plane-based approaches perform compared to the most popular legacy ones.

### *7.1 Link Failure*

According to [127], [128], link failure is a fundamental task, and it is the most frequent failure in the network. The literature proposed many approaches to mitigate this issue. This section discusses the performance of the programmable approaches compared with legacy ones against link failure.

Table VII compares the effectiveness of programmable data plane-based approaches with traditional ones regarding the link

failure problem. In legacy approaches, where the control plane is distributed, routers exchange protocol messages to advertise the changes in the network topology. According to [18], [129], it might take a very long time to inform all the routers in the network about the failed links where severe packet losses might occur. Subramanian *et al*. [55] claim that networks might take an unreasonable amount of time to converge from failures, even in state-of-the-art mechanisms. Besides, they claim that no state-of-the-art approach can ensure that policies hold [130]–[132]. On the other hand, they claim that D2R is able to compute alternative paths to any destination at a near line-rate while achieving policy compliance.

Obtaining sub-second convergence from link failure becomes achievable [133], [134] after the deployment of fast-convergence frameworks like IPFFR [135], Loop-Free Alternate [136], and MPLS Fast Reroute [15]. Nevertheless, these approaches work well only upon internal failures, where the average convergence time upon remote link failure is above 30 seconds [14], [15], [137]. On the other hand, Blink [18] can detect the remote failure within the first retransmission round and can reroute the traffic at line-rate by storing a per-prefix next-hops list. Moreover, Xu *et al*. [79] claim that P4Neighbor can achieve, in terms of forwarding entries, a reduction rate ranging from 57.9% to 84.5% compared with the traditional destination-based recovery mechanisms. The same authors also claim that the only drawback of using P4Neighbor is the increase in the number of hops by 1.08 to 1.98 hops.

The literature describes many centralized approaches which mitigate faulty links in WAN [138] and data centers [139]. Giesen *et al*. [119] claim that Wharf is the first approach of its kind that mitigates faulty links by running as an in-network function in a distributed manner. Tests show that when Wharf is deployed, the congestion window (cwnd) does not decrease, even if the loss rate is increasing. Wharf is able to sustain 5 Gigabits per second throughput under 10% packet loss condition. If Wharf is not deployed, the throughput is estimated to be less than 25 Megabits per second.

1+1 protection is usually deployed over the physical layer, the link layer, or MPLS [79]. In Multiprotocol Label Switching, or MPLS, each packet is assigned a label. The forwarding elements use the labels to forward the traffic [140]. P4-Protect [118] extends this approach and deploys 1+1 protection over the network layer. The drawback of P4-Protect is the increase in the processing delay. Compared to the processing time when P4-Protect is not deployed, the processing time at the ingress port is 127% for the unprotected forwarding and 166% for the protected one. At the egress port, the processing time at both the unprotected and the protected forwarding is 127% compared to the processing time when P4-Protect is not deployed.

Chiesa *et al*. [105] claim that FRR primitives from other contexts do not support arbitrary FRR sequences. In contrast to these primitives, the same authors claim that PURR can be a building block for any arbitrary FRR approach. Compared with F10 [141], the state-of-the-art FRR mechanism, PURR is able to reduce the flow completion time (FCT). For small flows and under one link failure, PURR can reduce the FCT from 653 microseconds to 384 microseconds under low network loads conditions. Under higher loads, PURR is able to reduce the FCT by a factor of 2x. Under two link failures, the reduction in FCT under 10% and 70% loads reach 5.5x and 2.8x simultaneously. Finally, the same authors claim that PURR achieves near-optimal throughput at low network loads.

TABLE VII
COMPARISON BETWEEN THE EFFECTIVENESS OF PROGRAMMABLE AND COMMON LEGACY APPROACHES AGAINST LINK FAILURES.

| Feature | Programmable Approaches | Traditional Approaches |
|---|---|---|
| Performance | High for both remote and local failures | Low for remote failures; medium for local failures |
| Policy Compliant | Yes | No |
| Memory | Medium (limited fast memory) | High (abundant slower memory) |
| Failure Detection | Near line-rate | Long time for remote failures; order of microseconds for local failures |
| Packet Loss | None to low | Medium to high |
| Flexibility | High | Low |
| Performance of FRR Primitives | High | Medium |

Cascone *et al*. [23] claim that SPIDER can detect and recover from failures in less than 1ms, whereas traditional OpenFlow-based solutions need 10s of milliseconds. For them, SPIDER's main advantage over other legacy approaches is that it is implemented totally on the data plane. SPIDER is inspired by legacy approaches like BFD [142] and MPLS FRR [143]. SPIDER outperforms BFD by eliminating the use of a slow control channel and reusing the data packets to piggyback heartbeats. SPIDER outperforms MPLS FRR by eliminating the need for a reservation protocol. FRR uses Resource Reservation Protocol (RSVP) to calculate the backup path. RSVP is used to reserve resources in a network. The forwarding element reserves network resources (e.g., bandwidth) before start transmitting data. RSVP is a complex protocol. SPIDER leverages a remote controller to provide both the primary and the backup paths instead of using the RSVP protocol.

Qu *et al*. [24] claim that SQR is the first approach of its kind that tries to recover packet losses at the in-network level without the need for any modification at the end hosts. SQR enhances the existing route recovery mechanisms by avoiding packet loss during failure convergence. To evaluate its performance, this approach is integrated with a link failure detection and network reconfiguration scheme. The scheme is ShareBackup [144]. ShareBackup installs shared backup switches in a data center network. These switches are used to replace the failed switches. After the failure has been solved, the backup switches are set free to be used again whenever a failure occurs. When ShareBackup is not enhanced by SQR, the TCP cwnd size is reduced rapidly due to link failure. However, when SQR is integrated with ShareBackup, the cwnd is not affected by the failure. The tests show that SQR

is able to isolate the failure from the sender where the cwnd sustains its value.

### *7.2 Load Balancing and Congestion Control*

Load balancing that adapts to traffic fluctuation on a small timescale and efficient congestion control is necessary to guarantee good performance [20]. This section compares the performance of programmable and legacy approaches when it comes to congestion and load imbalance.

Table VIII compares the effectiveness of programmable data plane-based approaches with traditional ones regarding load imbalance and congestion problems. According to Katta *et al*. [19], ECMP is the most deployed load balancing scheme. ECMP does not consider congestion, at which it separates the flows randomly based on a hash or round-robin scheduling [54]. Besides, ECMP might congest the network by assigning more than one large flow, i.e., flow with a high packet rate, to the same path due to hash collision [145]–[148]. Hash collisions occur when the hash function outputs the same hash to two different flows. ECMP performs poorly in asymmetric topologies [25], [149]. In asymmetric topologies, the traffic has multiple routes to enter and exit the topology [150]. The literature proposes many approaches to address ECMP limitations. In general, the approaches can be classified either as centralized schemes or as in-network distributed ones. Centralized approaches monitor the network and avoid forwarding heavy flows over the same path by detecting hash collisions [145], [151]. These approaches either require additional network infrastructure [151], or they are reactive to congestion [145]. In-network approaches might be effective; however, they need specialized networking hardware [152].

Compared to legacy in-network approaches, HULA [19] can achieve a lower average FCT. Under 50% and 90% load, HULA is able to reduce average completion time by 1.6x and 3x, respectively. HULA is a congestion-aware load balancing mechanism that reacts to link failures too. Contra [72] is restricted by neither a network topology nor a specific policy. Hsu *et al*. [72] claim that Contra has a competitive performance, in terms of flow completion time, with state-of-the-art systems that are based on deterministic topology and routing policy.

Many programmable approaches like DASH [20], MP-HULA [87], and W-ECMP [54] split the traffic over multiple distinct paths and show enhancement in the performance. Hsu *et al*. [20] claim that DASH outperforms both ECMP and HULA. DASH is 12-25% better than ECMP in symmetric topology, and it outperforms HULA by 22.1% and ECMP by 16% at 80% workload. Benet *et al*. [87] claim that MP-HULA with MPTCP outperforms both HULA (2.1x at 50% load, 1.7x at 80% load) and ECMP. When combined with MPTCP, ECMP performs the worst compared to HULA and MP-HULA. Ye *et al*. [54] show that W-ECMP performs better than HULA in the average FCT. W-ECMP enhances the FCT by 10% for large flows when compared with ECMP and outperforms HULA for small flows.

TABLE VIII
COMPARISON BETWEEN THE EFFECTIVENESS OF PROGRAMMABLE AND COMMON LEGACY APPROACHES AGAINST LOAD IMBALANCE AND CONGESTION.

| Feature | Programmable Approaches | Traditional Approaches |
|---|---|---|
| Performance | High | Medium |
| Cost | Low (a few nodes might suffice) | High (thousands of SLBs are used) |
| Memory | Medium (limited to on-switch memory) | High (external storage units can be used) |
| Visibility | Per-packet at line-rate | Limited |
| Accuracy | High (can reflect the exact state of the network) | Low (usually follow probabilistic algorithms to reflect the state of the network |
| Flexibility | High (operator can change the functionality of the algorithm) | Low (limited to the set of functionalities defined at the deployment |
| Performance Isolation | High | Low |
| Jitter | Low | High |
| Computation Overhead | Low | High |

Another limitation of ECMP is that it is not resilient against the changes to the application instance set [80]. Many legacy approaches tried to overcome this limitation by following the consistent hashing mechanism [12], [21], [153]–[155]. In the consistent hashing mechanism, the hash table is stored on multiple servers. Each part of the hash table has a copy on multiple servers. According to [21], consistent hashing does not consider the load state of the application while assigning the queries. This randomness might degrade the performance of the applications that depend on the control plane to perform multiple tasks [80]. Besides, Layer 4 load balancing, most often, is implemented in software servers [12], [21]. According to [11], [21], an SLB can support DIP pool changes and ensure the PCC. However, almost 3.75% of the data center size should be working as SLBs. Processing the packets by an SLB can add computation overhead, high latency and jitter [156], and poor performance isolation [11]. An SLB might be the bottleneck of the most delay-sensitive applications [157], [158].

On the other hand, Miao *et al*. [11] claim that SilkRoad is able to replace up to hundreds of SLBs, and it can reduce 45%-95% SRAM usage. 10 million connections can fit into Silkroad programmable switching ASIC. It ensures the PCC under the most frequent DIP pool updates recorded from the network [11]. Moreover, this approach is able to perform full line-rate load balancing with sub-microsecond processing latency and achieves tighter performance isolation than SLBs. Pit-Claudel *et al*. [80] claim that SHELL significantly increases the PCC when compared to other stateless load balancing implementations. SHELL can sustain sixty million packets per second (Mpps), which is 22x the value recorded from a single-core software implementation of an SLB used by Google [21].

Furthermore, Chord [159] is a widely deployed legacy load balancing approach. Chord is a DHT solution for data storage

and retrieval in peer-to-peer (P2P) networks. In a network with *N* peers, each peer holds routing information about *O(log n)* other peers [160]. Chord adds virtual nodes to the network by appending rules to the routing tables [121]. A main drawback of Chord is that the routing paths might be significantly longer than the shortest path. Another drawback of Chord is the storage overhead, as multiple routing rules are added to the routing table in order to create the virtual network [121].

Xie *et al*. [121] claim that GRED requires less memory and maintains shorter paths than Chord. Their evaluation shows that GRED uses <30% routing path lengths and performs better load balancing compared to Chord. On average, GRED requires 1 overlay hop to reach the destination, while Chord requires 5. While the number of rules added by Chord increases with the network size (i.e., additional *O(log n)* routing rules), the number of rules added by CRED is independent of the network size.

Besides, legacy networks use congestion signaling mechanisms to avoid congestion. These mechanisms notify senders to reduce the sending rate in order to avoid packet loss [83]. ECN [16], Data Center TCP (DCTCP) [161], and BECN [162] are common congestion signaling mechanisms. ECN works on the transport layer and allows the receiver to notify the sender about the congestion before it occurs. A router on the path between the sender and the receiver marks the ECN header when its queue occupancy reaches a predefined threshold. When the marked packet arrives at the receiver, the receiver sets the ECN echo bit in the TCP header to inform the sender that this packet has encountered congestion during its traversal [163]. One limitation of ECN is that it operates at the transport layer. Salim *et al*. [162] and Akujobi *et al*. [164] discussed the need for decoupling ECN from the transport layer. Another limitation of ECN is that some packet loss might occur before the sender is notified. If a flow has a high RTT, some packets might be lost before the notification from the receiver reaches the sender.

DCTCP extends ECN and returns the number of bytes that have encountered congestion in the network. However, this mechanism is only recommended for data center networks [163]. In BECN, ICMP Source Quench (ISQ) messages are used to notify the sender about the congestion. In this approach, the router directly notifies the sender without involving the receiver. One drawback of this approach is the additional bandwidth used by the generated messages. Another drawback is that the approach does not deploy any mechanism to verify that the notification message has been received by the sender.

AWW [17], P4QCN [39], and [83] deploy network-assisted congestion feedback, an approach that enhances congestion signals by using the programmable switches to notify the senders. AWW uses TCP advertised window to notify the senders; P4QCN uses the ECN header to notify the senders; and [83] uses NACKs to notify the senders. While AWW and P4QCN utilize packets coming from the receiver to generate the notification, [83] generates NACKs using the data plane of the programmable switches. Consequently, [83] has the fastest feedback approach compared to AWW and P4QCN. on the Other hand, [83] requires bandwidth to notify the sender, while AWW and P4QCN use no additional bandwidth to send the notification.

When compared with ECN-enabled flows, AWW is able to achieve a 5-10% increase in the throughput [17]. P4SQN is able to achieve around a 20% decrease in latency and packet rate loss compared to QCN [39].

Compared to BECN, the sender in AWW and P4QCN is guaranteed to receive network-assisted notifications. AWW and P4CN use the ongoing traffic between the two communicating entities to notify the sender, assuring that the notifications will reach the sender if there are packets being exchanged between the sender and the receiver. Another advantage of AWW and P4QCN over BECN is that no additional messages are generated, and thus, they use less bandwidth than BECN.

Similar to BECN, [83] generates messages instead of relying on ongoing traffic to notify the sender. In [83], programmable switches continuously generate messages. Switches stop generating packets only when the sender lowers his sending rate. Thus, the approach in [83] is more reliable than BECN at the cost of additional bandwidth usage.

## 8. Challenges and Future work

This section presents challenges pertaining to programmable switches and their effects on rerouting implementations in P4. Furthermore, a number of current initiatives and future directions addressing the presented challenges are discussed. The challenges and the future trends are illustrated in Fig. 18.

### *8.1 Security*

For the approaches that depend on data plane signals, defending themselves against cyberattacks might be extremely challenging. For instance, in Blink, if the hacker is able to hold half of the monitored flows, he/she can easily abuse the rerouting strategy. The attacker can trigger rerouting by retransmitting the same packets. In this way, the illusion of link failure is created, and the node reroutes the traffic unnecessarily. The unnecessary rerouting results in wasting bandwidth and congesting the network.

Approaches that use probes to get links' utilization might be targets for cyberattacks. One possible cyberattack can be performed by pushing fabricated probes that contain wrong links' utilization information. In this scenario, the deployed load balancing algorithm will reroute the traffic based on a false indication leading to load imbalance. This kind of attack can become even more severe if the attacker is able to direct the traffic through one path only. The targeted path will be over-utilized, and many connections will be lost due to congestion. Consequently, the overall performance of the data center will be severely affected.

Cyberattacks might target load balancing approaches that use hashing. An attacker can flood the network with new connection requests. The load balancer will reserve a hash value for each incoming request. Because the hash space of a hash function is limited, the load balancer might not be able to serve new connections.

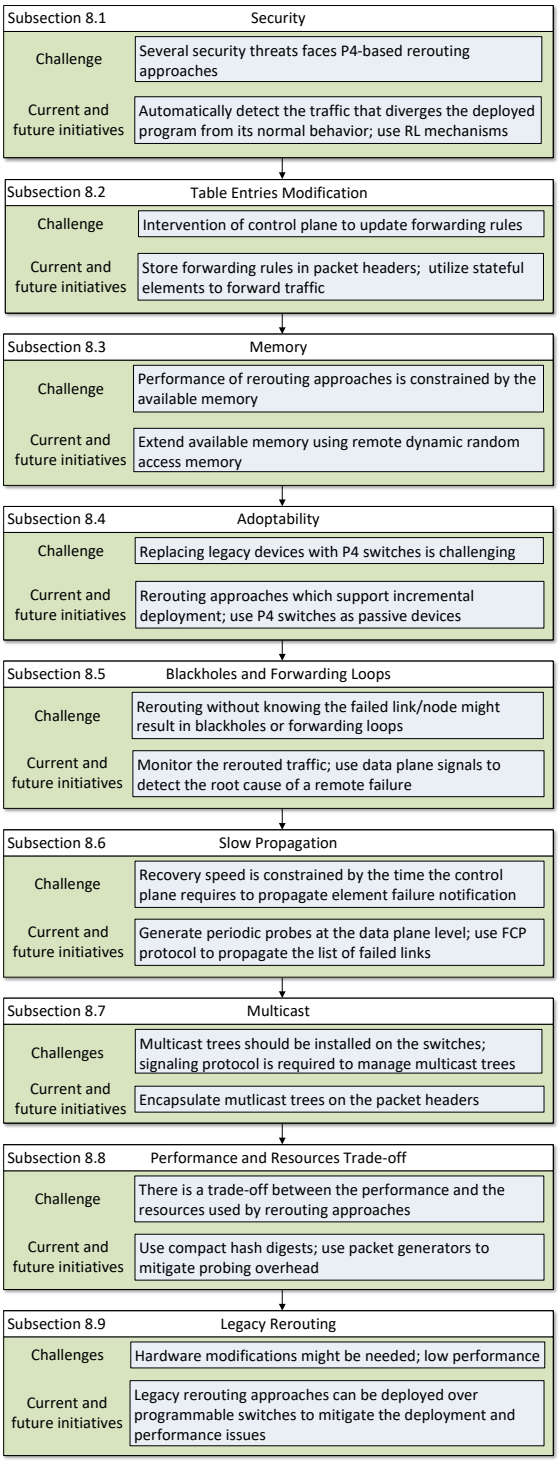


Fig. 18. Challenges and future trends.

**Current and Future Initiatives.** To address security issues, researchers have tried to detect attacks by monitoring unpredicted traffic patterns. However, this approach has had a highly unreliable performance [31]. An alternative approach was proposed by [165], which automatically detects the traffic that diverges the deployed program from its normal behavior. This approach might be unscalable as it might be challenging to identify when the program diverges from its normal behavior. As future work, researchers can consider using complex algorithms that leverage the advances in Machine Learning (ML) (e.g., Reinforcement Learning (RL)). Additionally, future work should consider defining an efficient data plane level collection mechanism to enhance the data analysis process and thus the attack detection speed and accuracy.

### *8.2 Table Entries Modification*

The objective of traffic rerouting is to mitigate a networking problem by changing the forwarding behavior of the switch such that the traffic is rerouted from the main path to an alternative path. Rerouting speed can be considered one of the main concerns of any rerouting algorithm. Many rerouting approaches calculate the alternative path proactively in order to speed up the traffic rerouting phase. However, a traditional unavoidable delay is the intervention of the control plane to change the forwarding rules. The data plane cannot directly modify the table entries. Delegating tasks to the control plane incurs latency and affects the application's performance.

**Current and Future Initiatives.** A common way to limit the intervention of the control plane is to forward the traffic based on a header value instead of modifying the forwarding table entries. P4Neighbor [79] encapsulates the alternative path rules inside the packet headers. The receiving P4Neighbor node chooses the egress port based on the information inside the packet header. The traffic rerouting phase in P4Neighbor happens at line-rate at which no control plane intervention is needed. SHELL [80] follows a similar approach as in P4Neighbot. The programmable switch in SHELL forwards the traffic based on the packet headers instead of using the forwarding tables. In future work, rerouting based on the packet headers should be used in stateful load balancing algorithms. Future work should also consider using stateful registers to control the forwarding behavior in link failure problems.

### *8.3 Memory*

In some stateful load balancing rerouting-based approaches, a switch has to maintain the state of each flow. This might be challenging as the memory inside the programmable switch is limited. The available memory space in the switch is decided based on the on-ship memory. Only a few hundred megabytes of SRAM [11] and a hundred kilobytes of TCAM [80] are available on the current switches. The efficiency of stateful load balancing mechanisms highly depends on the available memory. For instance, there is a trade-off between the size of the hash function, the number of the flows that can be monitored, and the performance of load balancing mechanisms that are based on hashing. Additionally, memory constraints might make some approaches more vulnerable to cyberattacks.

**Current and Future Initiatives.** Kim *et al*. [166], [167] proposed exploiting remote Dynamic Random Access Memory

(DRAM) deployed over some data centers' servers to extend the available memory on programmable switches. This approach is able to achieve near line-rate throughput, where the remote accessing overhead is estimated by 1-2 microseconds [31]. However, because the switch's data plane and the memory are not directly connected, packets might be lost. This loss should be eliminated. Otherwise, this approach might not be reliable enough to be deployed. Another limitation is that complex matching in remote memories is not addressed where only address-based memory access is supported. As future work, protective rerouting approaches (e.g., P:1+1) might be deployed between a programmable switch and the remote DRAM to guarantee packet delivery even under link failures. Because operators have no control over the incoming traffic, which makes optimal load balancing nearly impossible, future work could consider allowing programmable switches to share resources. In this case, the load will be divided equally among the switches even if the incoming traffic is unfairly distributed. In other words, SRAM and TCAM capacities of all the available programmable switches in a certain data center are considered one big block of shareable memory. The memory constraints might not hold unless all the deployed switches are overly-saturated with traffic. This approach, going in parallel with an efficient load balancing approach, might result in an optimal bisection of the available bandwidth.

### *8.4 Adoptability*

It is not easy for operators to adjust the networking infrastructure inside data centers. Programmable switches are no exception. Despite the significant performance improvements they bring, most operators will probably not exchange their legacy devices with programmable ones in one shot [31].

**Current and Future Initiatives.** Multiple rerouting approaches have already been proposed, which can be incrementally deployed [24]. Other approaches have suggested delegating data analysis operation to programmable switches, while legacy switches preserve the forwarding functionality [59], [168], [169]. Future work should consider deploying a hybrid system between legacy and programmable switches. The role of the programmable ones is to detect failures, gather legacy switches statistics, and reroute the traffic.

### *8.5 Blackholes and Forwarding Loops*

Data plane signals are successfully leveraged to infer remote link failure within one RTT. However, these signals cannot have a-priori information about the root cause of a disruption. The chosen backup path might direct the rerouted traffic toward the failed node resulting in blackholes and/or forwarding loops.

**Current and Future Initiatives.** Blink [18] monitors the backup paths to check their validity. In this way, any blackhole or forwarding loops might be detected. The drawback of this approach is that it is relatively slow as Blink sends the traffic and waits for any retransmission. If retransmission is detected, Blink infers that this path is not working and reroutes the traffic again; otherwise, Blink assumes that this path is functioning and no more actions are taken. As future work, more research should focus on the feasibility of using data plane signals to detect the root cause of the remote failures.

### *8.6 Slow Propagation*

After detecting link/node failure, rerouting approaches propagate the address of the failed element to other nodes in the network. The control plane is used to either generate the probes to propagate the address of the failed element or to route the probes. The intervention of the control plane slows the time needed to propagate the failure occurrence to other nodes. The control plane cannot generate probes at line-rate, and it might take a non-negligible amount of time to calculate the route of the probes. Usually, the control plane is used to guarantee that all the nodes in the network are informed about the address of the failed node.

**Current and Future Initiatives.** One approach to assure that all nodes are informed about the failure is to flood the network with probes. This approach does not scale well as the number of nodes in a data center increases. HULA [19] deploys a periodic probing mechanism that infers remote failures. Probes are guaranteed to reach all the nodes in the network even if a failure occurs. This approach might be effective, but it is restricted to tree-based typologies. D2R [55] deploys FCP protocol to propagate the list of failed links; however, not all remote nodes are guaranteed to be notified. Future work should consider a practical way that assures all remote nodes are notified without flooding the network with feedback probes.

### *8.7 Multicast*

Traditional multicast can be considered unscalable as the multicast tree is required to be installed on the switches. Besides, a signaling protocol is required to manage the multicast tree. Traditional multicast might fail to provide service for hundreds of thousands of tenants due to the data plane and control plane limitations [56]. As a consequence, tenants tend to rely on unicast-based approaches to perform multicast functionalities. This results in CPU overhead at the end-host and imposes high and unpredictable latency.

**Current and Future Initiatives.** P4-based multicast can be considered highly scalable and flexible. In P4, no status information is required to be stored inside the switches, and no signaling protocol is needed to manage the multicast tree [56]. Moreover, P4-based multicast supports dynamic tree updates and does not constrain the switch to multicast packets based on the type of the IP address. To the best of the writers' knowledge, no probe-based rerouting approach has deployed P4-based multicast. P4-based multicast can be used by the programmable approaches to propagate information about links' utilization in a network. For instance, the probing

system deployed by HULA (discussed in Section V. B.) uses traditional multicast. If HULA deployed the P4-based multicast, the overall performance of this load balancing approach would increase for at least two reasons. First, HULA would save memory by using packet headers to manage the multicast tree. Second, HULA would become more flexible as it can support the dynamic tree updates, and it is no more constrained by the fixed functionalities provided by the signaling protocol. In future work, load balancing approaches should leverage the potential of P4-multicast. Future work should also consider using P4-multicast to propagate failure events.

### *8.8 Performance and Resources Trade-off*

There is a trade between performance and resources in most programmable-based rerouting approaches. For example, in the hashing algorithms used to identify flows, there is a trade-off between the number of bits used to identify a flow and the memory resources used to store its ID. In link failure detection mechanisms, the higher the frequency of probes, the more bandwidth is wasted on these probes. On the other hand, the lower the frequency is, the slower the detection happens. In the approaches that monitor the data plane signals, the higher the number of monitored flows, the higher the accuracy of the approach at the cost of an increase in memory usage. In load balancing approaches, the performance depends on the speed of the generated probes, which directly increases the bandwidth usage.

**Current and Future Initiatives.** Miao *et al*. [11] mitigate the performance-resources trade-off by using a compact hash digest instead of the 5-tuple (source IP address, destination IP address, source port, destination port, transport protocol) hashing. This approach manages to reduce the storage needed to store each connection from 37 bytes to 2 bytes only. Pit *et al*. [80] mitigate the performance-resources trade-off by storing the index of the serving instance instead of maintaining a per-flow state while performing load balancing. The approach reduces memory usage while sustaining high accuracy. The main drawback of this approach is that it requires the end host to modify the packet structure. Regarding the probing overhead, the packet generators of the programmable switches can be used to implement the BDF protocol at the data plane level. Using the packet generators to implement the BDF protocol reduces the resource usage of the control plane and enhances the overall detection speed. The speed of detection depends on the frequency of generating probes. In future work, rerouting approaches should leverage the packet generation capability of the programmable switches to maximize both performance and resource utilization.

### *8.9 Legacy Rerouting*

Many legacy approaches suffer from severe limitations. For example, CONGA [25], a distributed congestion-aware load-balancing approach, was implemented in custom silicon on a switching chip [19]. For CONGA to be adopted, months of hardware design and verification efforts are needed. Even if the required modification to the hardware was applied, the CONGA algorithm could not be enhanced.

Another example is SWIFT [13]. SWIFT is a link failure mitigation technique that utilizes BGP withdrawal messages to predict failed links. The main drawback of SWIFT is that it might take minutes to detect remote link failures, as BGP updates can take minutes to propagate after the corresponding failure [19]. This technique uses the 48 bits of the MAC address to store a list of links a packet will traverse to reach its destination. It then stores a list of pre-computed backup next-hops for the links. Thus, the number of remote failures SWIFT can recover from and the number of backup next-hops it can use are constrained by the size of the MAC address field.

**Current and Future Initiatives.** Some legacy approaches are P4-friendly, i.e., they can be deployed over programmable switches. The features data plane programmability provides can mitigate different limitations of legacy approaches. For example, CONGA can be directly deployed on a P4 programmable switch, eliminating the need for months of hardware design and verification efforts. Further, programmable data planes allow the operator to update the algorithm of CONGA to accommodate new traffic patterns.

In the case of SWIFT, it can be improved by utilizing different features of programmable switches. After deploying SWIFT on a programmable switch, the approach will not be constrained by 48 bits to store the two lists, i.e., the list of links and the list of backup next-hops. The operator can define the length of the custom header which stores the two lists based on the number of links to be monitored. As a result, the trade-off between the number of links to be monitored and the number of backup next-hops is eliminated. Besides, SWIFT can utilize the packet generator hardware to accurately detect failed links. The packet generator will notify other switches of the location of the failed link. Thus, the time required by SWIFT to predict failed links will be eliminated, and consequently, the overall time required by SWIFT to reroute traffic will be significantly reduced.

Similar to SWIFT and CONGA, the performance of many P4-friendly legacy approaches (e.g., DRILL [170], Clove [27], and others [28]) can be significantly improved if they are implemented on programmable switches.

## 9. CONCLUSION

This article presents a detailed survey on different rerouting mechanisms based on programmable data planes. The survey describes the crucial need and wide deployability of traffic rerouting in fulfilling the current requirements. Afterward, the survey discusses the feedback mechanisms implemented by the recent rerouting approaches. It then presents the pros and cons of using one approach over the other. The survey explains how packets' headers are modified, how forwarding tables are altered, and how control signals are exchanged to guarantee near line-rate traffic rerouting and assure connectivity continuation. After that, rerouting approaches are classified based on networking issues they tackled and compared against the

state-of-the-art legacy approaches. The survey concludes by discussing challenges and considerations as well as various future trends and initiatives and explaining the potential of deploying rerouting in new research fields.

## ACKNOWLEDGMENT

This work was supported by the U.S. National Science Foundation under grant number 2118311.

TABLE IX
ABBREVIATIONS USED IN THIS ARTICLE.

| Abbreviation | Term |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| ALU | Arithmetic Logic Unit |
| API | Application Programming Interface |
| BFD | Bidirectional Forwarding Detection |
| BGP | Border Gateway Protocol |
| BMv2 | Behavioral Model Version 2 |
| BQE | Buffer Queuing Engine |
| CID | Connection Identifier |
| CPU | Central Processing Unit |
| CWND | Congestion Window |
| DASH | Data plane Adaptive Splitting with Hash threshold |
| DCTCP | Data Center TCP |
| DRAM | Dynamic Random Access Memory |
| DT | Delaunay Triangulation |
| ECMP | Equal-Cost Multi-Path Routing |
| FCT | Flow Completion Time |
| FEC | Forward Error Correction |
| FIFO | First-in-First-out |
| FRR | Fast Rerouting |
| HULA | Hop-by-hop Utilization-aware Load balancing Architecture |
| IP | Internet Protocol |
| ISQ | ICMP Source Quench |
| ITU | International Telecommunication Union |
| ML | Machine Learning |
| MPLS | Multiprotocol Label Switching |
| Mpps | Million Packet Per Second |
| LP | Linear Programming |
| P4 | Programming Protocol-Independent Packet Processors |
| PCC | Per Connection Consistency |
| PISA | Protocol Independent Switch Architecture |
| PRE | Packet buffer and Replication Engine |
| PSA | Portable Switch Architecture |
| PTE | Protection Tunnel Egress |
| PTI | Protection Tunnel Ingress |
| QCN | Quantized Congestion Notification |
| RL | Reinforcement Learning |
| RTT | Round Trip Time |
| SALU | Statefull Arithmetic Logic Unit |
| SDN | Software-Defined Networking |
| SLB | Software Load Balancer |
| SN | Sequence Number |
| SQR | Shared Queue Ring |
| SRAM | Static Random-Access Memory |
| Tbps | Terabits per Second |
| TCAM | Ternary Content Addressable Memory |
| TCP | Transport Control Protocol |
| ToR | Top-of-Rack |
| VIP | Virtual Internet Protocol |
| W-ECMP | Weighted Equal-Cost Multi-Path |

## REFERENCES

[1] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, pp. 350–361, 2011.

[2] N. Promwongsa, A. Ebrahimzadeh, D. Naboulsi, S. Kianpisheh, F. Belqasmi, R. Glitho, N. Crespi, and O. Alfandi, "A comprehensive survey of the tactile Internet: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, 2020.

[3] K. Sacha and M. Jon, "What is the tactile Internet." [online]. Available: https://tinyurl.com/hynmp8bu.

[4] D. Van Den Berg, R. Glans, D. De Koning, F. A. Kuipers, J. Lugtenburg, K. Polachan, P. T. Venkata, C. Singh, B. Turkovic, and B. Van Wijk, "Challenges in haptic communications over the tactile Internet," *IEEE Access*, vol. 5, pp. 23502–23518, 2017.

[5] B. Turkovic, F. Kuipers, N. van Adrichem, and K. Langendoen, "Fast network congestion detection and avoidance using P4," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, pp. 45–51, 2018.

[6] R. Kaur and P. Luthra, "Load balancing in cloud computing," in *Proceedings of international conference on recent trends in information, telecommunication and computing, ITC*, Citeseer, 2012.

[7] R. R. Malladi, "An approach to load balancing in cloud computing," *Int. J. Innov. Res. Sci., Eng. Technol*, vol. 4, no. 5, pp. 3769–3777, 2015.

[8] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in *2012 international conference on computing, electronics and electrical technologies (ICCEET)*, pp. 877–880, IEEE, 2012.

[9] J. Crichigno, N. Ghani, J. Khoury, W. Shu, and M. Wu, "Dynamic routing optimization in wdm networks," in *2010 IEEE Global Telecommunications Conference (Globecom)*, 2010.

[10] J. Crichigno, W. Shu, and M. Wu, "Throughput optimization and traffic engineering in WDM networks considering multiple metrics," in *2010 IEEE International Conference on Communications (ICC)*, 2010.

[11] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICS," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 15–28, 2017.

[12] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, *et al.*, "Ananta: Cloud scale load balancing," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 207–218, 2013.

[13] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever, "Swift: Predictive fast reroute," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 460–473, 2017.

[14] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 175–187, 2000.

[15] R. Oliveira, B. Zhang, D. Pei, and L. Zhang, "Quantifying path exploration in the Internet," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 445–458, 2009.

[16] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," tech. rep., 2001.

[17] J. Jiang and Y. Zhang, "An accurate congestion control mechanism in programmable network," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0673–0677, IEEE, 2019.

[18] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 161–176, 2019.

[19] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, pp. 1–12, 2016.

[20] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Adaptive weighted traffic splitting in programmable data planes," in *Proceedings of the Symposium on SDN Research*, pp. 103–109, 2020.

[21] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 523–535, 2016.

[22] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[23] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sanso, "Fast failure detection and recovery in SDN with stateful data plane,"

*International Journal of Network Management*, vol. 27, no. 2, p. e1957, 2017.

[24] T. Qu, R. Joshi, M. C. Chan, B. Leong, D. Guo, and Z. Liu, "Sqr: In-network packet loss recovery from link failures for highly reliable datacenter networks," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–12, IEEE, 2019.

[25] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, pp. 503–514, 2014.

[26] S. Ghorbani, "Micro load balancing for low-latency data center networks,"

[27] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pp. 323–335, 2017.

[28] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *NSDI*, pp. 113–126, 2013.

[29] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.

[30] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[31] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, 2021.

[32] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 784–818, 2017.

[33] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *Journal of Network and Computer Applications*, p. 103561, 2022.

[34] A. Satapathy, *Comprehensive study of P4 programming language and software-defined networks*. PhD thesis, Institute for Development and Research in Banking Technology, 2018.

[35] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, vol. 7, pp. 47804–47840, 2019.

[36] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment," *Computer Networks*, vol. 207, p. 108800, 2022.

[37] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "A survey on TCP enhancements using P4-programmable devices," *Computer Networks*, vol. 212, p. 109030, 2022.

[38] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017.

[39] J. Geng, J. Yan, and Y. Zhang, "P4QCN: Congestion control using P4-capable device in data center networks," *Electronics*, vol. 8, no. 3, p. 280, 2019.

[40] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 219–230, 2013.

[41] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[42] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, 2021.

[43] T. P. A. W. Group, "P4$_{16}$ portable switch architecture (PSA)." [online]. Available: https://tinyurl.com/yanh5wz3.

[44] V. Gurevich, "P4 mapping to barefoot tofino(tm)." P4 Language Consortium, Oct. 2019, [online]. Available: https://tinyurl.com/4j97r2t5.

[45] V. Gurevich and A. Fingerhut, "P4$_{16}$ programming for intel tofino using intel P4 studio." P4 Language Consortium, May 2021, [online]. Available: https://tinyurl.com/37vbxeyy.

[46] Barefoot Networks, "Packet generation in the data plane of a forwarding element." [online]. Available: https://tinyurl.com/j6e6zya9.

[47] Z. Xi, Y. Zhou, D. Zhang, J. Wang, S. Chen, Y. Wang, X. Li, H. Wang, and J. Wu, "Hypergen: High-performance flexible packet generator using programmable switching asic," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pp. 42–44, 2019.

[48] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, "P4sta: High performance packet timestamping with programmable packet processors," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[49] D. Jindal, R. Joshi, and B. Leong, "P4traffictool: Automated code generation for p4 traffic generators and analyzers," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 152–153, 2019.

[50] Wireshark. [online]. Available: https://www.wireshark.org/.

[51] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, pp. 275–287, 2015.

[52] Scapy, "Packet crafting for Python2 and Python3." [online]. Available: https://scapy.net/.

[53] Clementperon, "PcapPlusPlus." [online]. Available: https://github.com/seladb/PcapPlusPlus.

[54] J.-L. Ye, C. Chen, and Y. H. Chu, "A weighted ECMP load balancing scheme for data centers using P4 switches," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp. 1–4, IEEE, 2018.

[55] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "D2R: Dataplane-only policy-compliant routing under failures," *arXiv preprint arXiv:1912.02402*, 2019.

[56] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 458–471, 2019.

[57] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[58] W. Wang, P. Tammana, A. Chen, and T. E. Ng, "Grasp the root causes in the data plane: Diagnosing latency problems with spidermon," in *Proceedings of the Symposium on SDN Research*, pp. 55–61, 2020.

[59] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang, "Fine-grained queue measurement in the data plane," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 15–29, 2019.

[60] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–16, 2018.

[61] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with* flow," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 823–835, 2018.

[62] GitHub: FlowLens, "." [online]. Available: https://github.com/dmbb/FlowLens.

[63] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 164–176, 2017.

[64] L. Tang, Q. Huang, and P. P. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2026–2034, IEEE, 2019.

[65] B. Niu, J. Kong, S. Tang, Y. Li, and Z. Zhu, "Visualize your ip-over-optical network in realtime: A p4-based flexible multilayer in-band network telemetry (ml-int) system," *IEEE Access*, vol. 7, pp. 82413–82423, 2019.

[66] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, and G. Pongracz, "Programmable event detection for in-band network telemetry," in *2019 IEEE 8th international conference on cloud networking (CloudNet)*, pp. 1–6, IEEE, 2019.

[67] Z. Liu, J. Bi, Y. Zhou, Y. Wang, and Y. Lin, "Netvision: Towards network telemetry as a service," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pp. 247–248, IEEE, 2018.

[68] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*, pp. 487–495, IEEE, 2019.

[69] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, vol. 180, p. 107386, 2020.

[70] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[71] TechTarget, "Probe." [online]. Available: https://tinyurl.com/35a36dbu.

[72] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 701–721, 2020.

[73] E. Earls, "Top-of-rack switching." [online]. Available: https://tinyurl.com/ypamrrye.

[74] G. Lee, "Cloud data center networking topologies." [online]. Available: https://tinyurl.com/yf3mz99r.

[75] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 241–252, 2007.

[76] Gartner, "Network topology." [online]. Available: https://tinyurl.com/8uuznpj5.

[77] Techopedia, "Network map." [online]. Available: https://tinyurl.com/2p87fvtj.

[78] M. McGee, "What is a network map?." [online]. Available: https://tinyurl.com/26t8yhpt.

[79] J. Xu, S. Xie, and J. Zhao, "P4Neighbor: Efficient link failure recovery with programmable switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 388–401, 2021.

[80] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, and T. Clausen, "Stateless load-aware load balancing in P4," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pp. 418–423, IEEE, 2018.

[81] R. Pan, B. Prabhakar, and A. Laxmikantha, "QCN: Quantized congestion notification," *IEEE802*, vol. 1, pp. 52–83, 2007.

[82] M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar, "Stability analysis of QCN: the averaging principle," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 49–60, 2011.

[83] A. Feldmann, B. Chandrasekaran, S. Fathalli, and E. N. Weyulu, "P4-enabled network-assisted congestion feedback: A case for nacks," in *Proceedings of the 2019 Workshop on Buffer Sizing*, pp. 1–7, 2019.

[84] A. Feldmann, J. Rexford, and R. Caceres, "Efficient policies for carrying web traffic over flow-switched networks," *IEEE/ACM transactions on Networking*, vol. 6, no. 6, pp. 673–685, 1998.

[85] A. M. Abdelmoniem and B. Bensaou, "Reconciling mice and elephants in data center networks," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 119–124, IEEE, 2015.

[86] L. Guo and I. Matta, "The war between mice and elephants," in *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pp. 180–188, IEEE, 2001.

[87] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, "Mp-hula: Multipath transport aware load balancing using programmable data planes," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 7–13, 2018.

[88] "The P4 language specification." The P4 Language Consortium, May 2017, [online]. Available: https://tinyurl.com/5n8evdey.

[89] "$P4_{16}$ language specification." The P4 Language Consortium, May 2017, [online]. Available: https://tinyurl.com/2dkpj7ck.

[90] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 234–244, 1994.

[91] J. Chroboczek, "The babel routing protocol, rfc 6126," *Quagga Routing Software Suite, GPL licensed*, 2011.

[92] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's burstiness with flowlet switching," in *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, Citeseer, 2004.

[93] J. He, M. Suchara, M. Bresler, J. Rexford, and M. Chiang, "Rethinking Internet traffic management: From multiple decompositions to a practical protocol," in *Proceedings of the 2007 ACM CoNEXT Conference*, pp. 1–12, 2007.

[94] N. Michael and A. Tang, "HALO: Hop-by-hop adaptive link-state optimal routing," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1862–1875, 2014.

[95] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 253–264, 2005.

[96] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 3, pp. 1300–1309, IEEE, 2001.

[97] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 431–439, IEEE, 2016.

[98] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, pp. 63–74, 2010.

[99] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[100] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "{PCC} vivace: Online-learning congestion control," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 343–356, 2018.

[101] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can {JUMP} them!," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 1–14, 2015.

[102] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 29–42, 2017.

[103] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 329–342, 2018.

[104] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *Proc. HotNets*, pp. 59–64, 2004.

[105] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, "PURR: A primitive for reconfigurable fast reroute: Hope for the best and program for the worst," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 1–14, 2019.

[106] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *IEEE INFOCOM 2004*, vol. 4, pp. 2307–2317, IEEE, 2004.

[107] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[108] IBM, "Db2 hash spaces." [online]. Available: https://tinyurl.com/5c44c43e.

[109] Ben, "What does it mean to hash data and do I really care?." [online]. Available: https://tinyurl.com/46xttw3e.

[110] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[111] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and Ulterior Software Engineering*, pp. 195–216, 2017.

[112] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, vol. 11, pp. 22–22, 2011.

[113] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "6lb: Scalable and application-aware load balancing with segment routing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 819–834, 2018.

[114] N. Salunke, "Extendible hashing (dynamic approach to dbms)." [online]. Available: https://tinyurl.com/2p9waret.

[115] A. Zola, "Hashing." [online]. Available: https://tinyurl.com/bdhbcje3.

[116] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging zipf's law for traffic offloading," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 16–22, 2012.

[117] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75–88, 2014.

[118] S. Lindner, D. Merling, M. Häberle, and M. Menth, "P4-protect: 1+1 path protection for P4," in *Proceedings of the 3rd P4 Workshop in Europe*, pp. 21–27, 2020.

[119] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, *et al.*, "In-network computing to the rescue of faulty links," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 1–6, 2018.

[120] TechTarget, "Forward error correction (FEC)." [online]. Available: https://tinyurl.com/44vdcfnh.

[121] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1029–1039, IEEE, 2019.

[122] E. Kawaguchi, H. Kasuga, and N. Shinomiya, "Unsplittable flow edge load factor balancing in sdn using p4 runtime," in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, IEEE, 2019.

[123] N. McKeown, "P4 Runtime – Putting the Control Plane in Charge of the Forwarding Plane." https://tinyurl.com/2hpxr848, 2017. [Online;].

[124] C. D. Toth, J. O'Rourke, and J. E. Goodman, *Handbook of discrete and computational geometry*. CRC press, 2017.

[125] P. Bose and P. Morin, "Online routing in triangulations," *SIAM journal on computing*, vol. 33, no. 4, pp. 937–951, 2004.

[126] H. Kasuga, M. Yamada, and N. Shinomiya, "A traffic load leveling method for communication links with paired tiesets in a distributed management network," in *Abstracts of IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, pp. 126–136, 2018.

[127] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–528, 1998.

[128] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, vol. 1, pp. 406–416, IEEE, 2003.

[129] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 293–306, 2001.

[130] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," *ACM Sigplan Notices*, vol. 49, no. 1, pp. 113–126, 2014.

[131] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 328–341, 2016.

[132] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp. 543–546, 2014.

[133] A. K. Atlas, G. Choudhury, and D. Ward, "IP fast reroute overview and things we are struggling to solve," *North American Network Operators Group (NANOG)*, vol. 33, 2005.

[134] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 35–44, 2005.

[135] M. Shand and S. Bryant, "IP fast reroute framework," 2010.

[136] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," 2008.

[137] P. Huang, A. Feldmann, and W. Willinger, "A non-instrusive, wavelet-based approach to detecting network performance problems," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 213–227, 2001.

[138] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, pp. 527–538, 2014.

[139] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 362–375, 2017.

[140] Paloalto Networks, "MPLS | what is multiprotocol label switching." [online]. Available: https://tinyurl.com/y7wjuanw.

[141] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 399–412, 2013.

[142] D. Katz and D. Ward, "Bidirectional forwarding detection (BFD)(RFC 5880)," *IETF*, 2010.

[143] P. Pan *et al.*, "Fast reroute extensions to RSVP-TE for LSP tunnels", rfc 4090," 2005.

[144] D. Wu, Y. Xia, X. S. Sun, X. S. Huang, S. Dzinamarira, and T. E. Ng, "Masking failures from application performance in data center networks with shareable backup," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 176–190, 2018.

[145] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, *et al.*, "Hedera: dynamic flow scheduling for data center networks.," in *Nsdi*, vol. 10, pp. 89–92, San Jose, USA, 2010.

[146] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proceedings of the Ninth ACM conference on Emerging Networking Experiments and Technologies*, pp. 49–60, 2013.

[147] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*, pp. 2130–2138, IEEE, 2013.

[148] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: Reducing the flow completion time tail in datacenter networks," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 139–150, 2012.

[149] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*, pp. 1–14, 2014.

[150] TREND MICRO, "What is symmetric vs. asymmetric mode?." [online]. Available: https://tinyurl.com/36dfmu2v.

[151] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 407–418, 2014.

[152] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[153] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pp. 654–663, 1997.

[154] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," *Computer Networks*, vol. 31, no. 11-16, pp. 1203–1213, 1999.

[155] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Transactions on Networking*, vol. 6, no. 1, pp. 1–14, 1998.

[156] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 27–38, 2014.

[157] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 249–264, 2016.

[158] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[159] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM computer communication review*, vol. 31, no. 4, pp. 149–160, 2001.
[160] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.
[161] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, and G. Judd, "Data center TCP (DCTCP): TCP congestion control for data centers," tech. rep., 2017.
[162] J. H. Salim, B. Nandy, and N. Seddigh, "A proposal for backward ECN for the internet protocol (IPv4/IPv6)," *Internet Draft, draft-salim-jhsbnns-ecn-00. txt*, 1998.
[163] S. Shahzad, E.-S. Jung, J. Chung, and R. Kettimuthu, "Enhanced explicit congestion notification (EECN) in TCP with P4 programming," in *2020 International Conference on Green and Human Information Technology (ICGHIT)*, pp. 35–40, IEEE, 2020.
[164] F. Akujobi, I. Lambadaris, R. Makkar, N. Seddigh, and B. Nandy, "BECN for congestion control in TCP/IP networks: study and comparative evaluation," in *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, vol. 3, pp. 2588–2593, IEEE, 2002.
[165] Q. Kang, J. Xing, and A. Chen, "Automated attack discovery in data plane systems," in *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*, 2019.
[166] D. Kim, Y. Zhu, C. Kim, J. Lee, and S. Seshan, "Generic external memory for switch data planes," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pp. 1–7, 2018.
[167] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 90–106, 2020.
[168] E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "Dynamic router's buffer sizing using passive measurements and P4 programmable switches," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 01–06, IEEE, 2021.
[169] E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "Enabling TCP pacing using programmable data plane switches," in *2019 42nd IEEE International Conference on Telecommunications and Signal Processing (TSP)*, 2019.
[170] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 225–238, 2017.