A Survey on TCP Enhancements using P4-programmable Devices

Jose Gomez[a], Elie F. Kfoury[a], Jorge Crichigno[a], Gautam Srivastava[b]
[a]College of Engineering and Computing, University of South Carolina, Columbia, U.S.A
[b]Department of Mathematics and Computer Science, Brandon University, Canada

Abstract

The increasing performance requirements of today's Internet applications demand a reliable mechanism to transfer data. Many applications rely on the Transmission Control Protocol (TCP) as the transport protocol, due to its ability to adapt to properties of the network and to be robust in the face of many kinds of failures. However, improving the performance of applications that rely on TCP has been limited by the closed nature of legacy switches, which do not provide accurate visibility of network events. With the emergence of P4-programmable devices, developers can rapidly implement and test customized solutions that use fine-grained telemetry, provide sub round-trip time feedback to end devices to enhance congestion control, precisely isolate traffic to offer better Quality of Service (QoS), quickly detect congestion and re-route traffic via alternate paths, and optimize server resources by offloading protocols.

This paper first surveys recent works on P4-programmable devices, focusing on schemes aimed at enhancing TCP performance. It provides a taxonomy classifying the aspects that impact TCP's behavior, such as congestion control, Active Queue Management (AQM) algorithms, TCP offloading, and network measurement schemes. Then, it compares the P4-based solutions, and contrasts those solutions with legacy implementations. Lastly, the paper presents challenges and future trends.

Keywords: TCP, P4, Programmable Data Plane, Congestion Control, AQM, SmartNICs, Network Diagnosis.

1. Introduction

The Transmission Control Protocol (TCP) [1] has been adopted as the standard transport protocol used by most Internet applications to transfer data. Efforts to enhance the performance and efficiency of TCP and other transport protocols [2–5] focus on improving congestion control algorithms, performing fine-grained network measurements, and offloading transport protocol functions to specialized network hardware.

With the advent of programmable data planes, developing custom protocols is performed in a top-down approach, facilitating the experimentation of novel ideas, diverging from the development process that the networking industry traditionally follows. The Programming Protocol-independent Packet Processor (P4) [6] language is widely adopted to describe the forwarding behavior of programmable data planes. P4 is a domain-specific language, which provides a standardized way to describe how packets are processed independently of the target's architecture. TCP congestion control and Active Queue Management (AQM) algorithms can benefit from the P4's language packet manipulation features to have more control and a better view of network events. Moreover, P4 is supported by various network components such as programmable Network Interface Cards (NICs). They can offload network stack operations and server workloads by executing them locally and reducing the communication overhead via bypassing the Operating System (OS) kernel.

Emerging technologies such as 5G, cloud computing services, big data analysis, and the Internet of Things (IoT) demand high utilization, low latency, and dynamic management of computer networks. Nevertheless, employing legacy networking devices to implement such technologies increases the complexity of the underlying network and provides less flexibility to apply changes or add new features. The management and maintenance of the legacy network devices depend on the features that the vendor included in the device, which constrains operators to define the forwarding behavior and customize traffic monitoring. Furthermore, legacy devices suffer from the network ossification problem [7], which is a significant barrier to implementing new protocols.

This paper surveys and discusses schemes that enhance TCP performance using P4-programmable data planes, focusing on four main categories: congestion control, AQM algorithms, TCP offloading using programmable NICs, and network diagnosis schemes that leverage P4-programmable data planes to detect, report, and troubleshoot TCP performance degradation.

Traditionally, congestion control is achieved using end-to-end mechanisms with little support from the network, relying on packet losses to share resources fairly. AQM algorithms complement congestion control schemes by taking actions in the intermediate nodes (i.e., switches) rather than waiting for the end host to react. In such a way, AQMs prevent network congestion from escalating. This survey covers RFC-standardized and custom

Email addresses: gomezgaj@email.sc.edu (Jose Gomez), ekfoury@email.sc.edu (Elie F. Kfoury), jcrichigno@cec.sc.edu (Jorge Crichigno*), srivastavag@brandonu.ca (Gautam Srivastava).
* Corresponding author

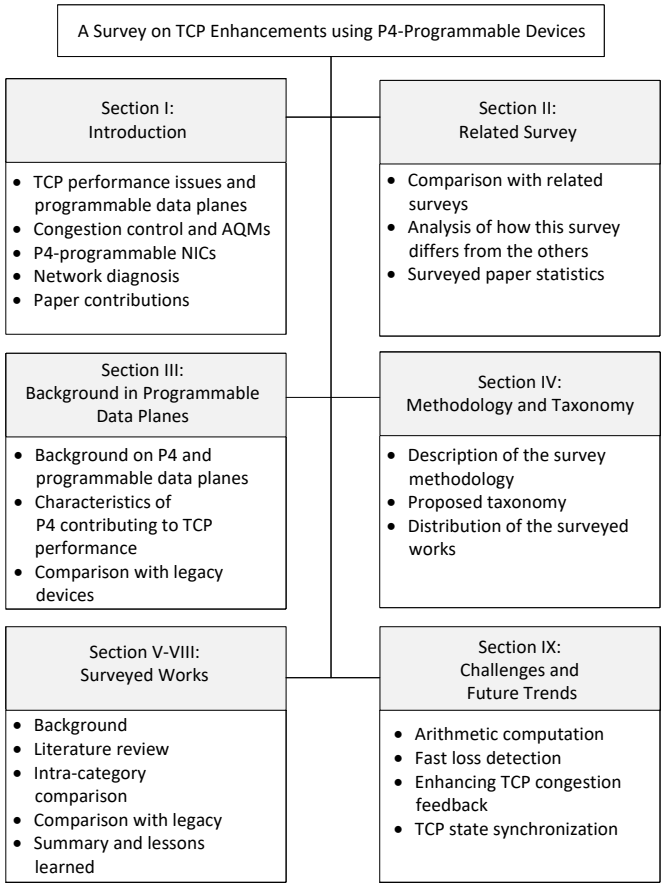

Figure 1: Paper roadmap.

AQM algorithms implemented in P4, showing how P4-programmable data planes are employed to realize protocols such as Random Early Detection (RED) [8], Proportional-Integral controller Enhanced (PIE) [9], CoDel [10], Common Application Kept Enhanced (CAKE) [11] and Fair Queueing (FQ) [12, 13]. Additionally, this paper covers custom AQMs designed to solve performance issues by implementing a solution based on theoretical approaches.

Another critical limitation that affects TCP performance is produced by the processing time that packets spend in the end host. TCP and other transport protocols are implemented in the end hosts, which use the general-purpose CPU to encapsulate, verify and fragment TCP segments. P4-programmable NICs can offload TCP and part of the kernel networking processes, making packet processing closer to the network, reducing the processing overhead, and releasing the server's resources. This survey covers works related to P4-programmable NICs that extend, offload and optimize the resources used by protocols such as TCP and applications entirely implemented in the NIC.

This paper also presents relevant works on network monitoring and diagnosis. Programmable data planes have more granularity to detect, report, and react to network issues at a line rate. Compared to legacy flow export protocols such as NetFlow [23] and IPFIX [24], programmable data planes can be employed to detect network flaws and mitigate them in an early stage. Moreover, either the control or data plane can take actions (e.g., marking, dropping, prioritizing packets, precomputing arithmetic operations, and rerouting traffic) to mitigate a failure and provide an accurate report of network events.

1.1. Contributions

To the best of the authors' knowledge, the literature has been missing a survey on TCP enhancement leveraged on P4-programmable data planes. Programmable data planes reduce the complexity of forwarding devices, facilitate adding new features, and supports fine-grained telemetry. This paper also discusses the usage of P4-programmable NICs to offload the server's resources. Performing part of the computation in the NIC, offloads servers from handling network stack processing. Furthermore, it also describes P4-based network diagnosis schemes used to report and mitigate performance issues. The granularity offered by programmable data planes is essential to identify and minimize performance degradation issues in an early stage.

This paper also provides a comparison between P4 schemes in the same category and with legacy approaches. Additionally, the challenges and future trends are also discussed. The main contributions of this survey can be framed as follows:

- Discussing the key aspects that make programmable data planes good candidates for improving network performance.
- Surveying the most relevant works that leverage data plane programmability to improve TCP performance and efficiency.
- Proposing a taxonomy that categorizes the works that enhance network and TCP performance.
- Describing the common challenges and considerations and listing current and future initiatives that can advance the state-of-the-art systems.
- Discussing challenges and limitations of the surveyed works.

1.2. Paper Organization

Figure 1 illustrates the paper roadmap. Section 2 describes the related surveys. Section 3 provides a background congestion control algorithms, AQMs and P4. Section 4 described the proposed taxonomy, section 5 surveys the P4-assisted congestion control, section 6 describes the P4-based AQMs, section 7 covers the improvements that P4 programmable NICs provide to offload network operations, section 8 explains how P4 devices are employed to detect, prevent and mitigate network performance issues. Section 9 discusses the challenges and future work and, section 10 concludes the paper.

1.3. Routers and Switches

Traditionally, a switch refers to a layer-2 device, and a router refers to a layer-3 device. However, with programmable data planes, the programmer decides what layers and protocols to parse and process. Thus, in this article, the terms switch and router will be used interchangeably, noting that a switch can process not only layer-2 protocols but also upper-layer protocols.

Table 1: Comparison with Related Surveys.

Survey	Programmable data planes and P4 language			Taxonomy				Discussions		Focus of the survey
	Evolution	Description	Features	Background	Literature	Intra-category comparison	Comparison with legacy	Challenges	Future directions	
[14]	●	⊙	⊙	⊙	⊙	○	○	○	⊙	SDN
[15]	●	○	○	⊙	⊙	○	○	⊙	⊙	General
[16]	⊙	⊙	⊙	○	○	○	○	○	○	SDN
[17]	●	○	⊙	○	⊙	○	○	○	○	SDN
[18]	⊙	⊙	○	⊙	⊙	⊙	○	⊙	⊙	Measurements
[19]	⊙	●	⊙	●	⊙	○	○	⊙	⊙	General
[20]	●	●	⊙	●	⊙	○	⊙	●	●	SDN
[21]	●	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	General
[22]	○	⊙	⊙	⊙	⊙	○	⊙	⊙	●	5G/SDN
This survey	⊙	●	●	●	●	●	●	●	●	TCP

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

2. Related Surveys

Dargahi *et al.* [14] surveyed the security applications enabled by Software-Defined Networking (SDN). They provide a background on stateful SDN data planes, dissect the relevant security issues and present an analysis of concrete use case applications such as the port knocking application, the UDP flooding mitigation application, and the failure recovery. They conclude the paper by discussing significant vulnerabilities and possible attacks to the stateful SDN data plane. This survey does not cover any topic related to TCP and how P4-programmable data planes can improve aspects of such protocol.

Bifulco and Retvari, [15] presented a survey on programmable data planes that focuses on the recent trends and issues in the design and implementation of programmable network devices. They cover the abstractions and architectures proposed, debated, realized, and deployed during the last decade. The authors discuss future research direction, providing a list of open problems based on abstractions, simplified operations, data plane optimization, and verification frameworks. The survey does not describe current works that aim at solving the issues covered in the paper.

Satapathy [16] presented a survey on SDN and P4. The author analyses the issues of traditional networks and how programmable data planes can be employed to support and improve cloud services and big data applications. The author presents two use-cases consisting of advanced network telemetry and DDoS attack mitigation using programmable data planes. The survey also collects and compares P4 related work. The study concludes by focusing on future works related to the evolution of programmable data planes, P4 language, and defining high-level policies that can be implemented using P4.

Kaljic *et al.* [17] provides a comprehensive survey on programmable data planes architectures with a particular emphasis on the problem of programmability and flexibility. The authors provide an overview of software and hardware technologies that enable SDN data plane implementations, identifying key factors that deviate from the original data plane architectures related to Forwarding and Control Element Separation-based (ForCES) and OpenFlow specifications. The paper categorizes and describes works that include performance improvements, energy consumption, quality of service, measurement and monitoring, security, and integration with other network technologies. Finally, they discuss and propose future research directions that should be considered to develop data plane architectures. This survey does not provide any comparison between related schemes and legacy devices.

Tan *et al.* [18] presented a survey about the latest applications leveraged on In-band Network Telemetry (INT). The authors analyze the development history, research situation, and application results of INT technology. The work covers INT applications such as congestion control algorithms, network troubleshooting, microburst detection, and traffic engineering. Although the survey discusses key technologies in the field, it does not compare the works within the same category and with legacy devices.

Hauser *et al.* [19] present a survey covering P4 and its applied research domains. The paper starts by introducing the concept of data plane programming and the relationship with adjacent concepts. They describe the P4 programming language, architecture, compilers, targets, and data plane APIs. They also include works in areas such as optimization of development and deployment, research on P4 targets, and P4-specific approaches for control plane operations. This survey contains brief descriptions of P4-related works but lacks an in-depth analysis and comparison between each category's results.

Kaur *et al.* [20] present a survey in programmable data planes in the context of SDN. The authors provide a comparison of the related works, background on P4 language, and the most relevant data plane architectures. They summarize and discuss works related to network monitoring, security, load balancing, and packet aggregation. The paper lacks inter and intra-category comparison and does not explore aspects that involve TCP performance. The paper concludes by providing a discussion about research gaps in programmable data planes.

Kfoury *et al.* [21] presented an exhaustive survey related to P4 programmable data planes. It includes a general overview of the P4 language and its applications. The authors thoroughly reviewed the literature to provide a taxonomy of applications developed with P4 language, surveying, classifying, and discussing challenges collected from more than 200 articles. It also presents future perspectives and open research issues. However, the survey provides a general description of each topic and does not cover TCP-related aspects.

Paolucci *et al.* [22] surveyed P4-programmable data

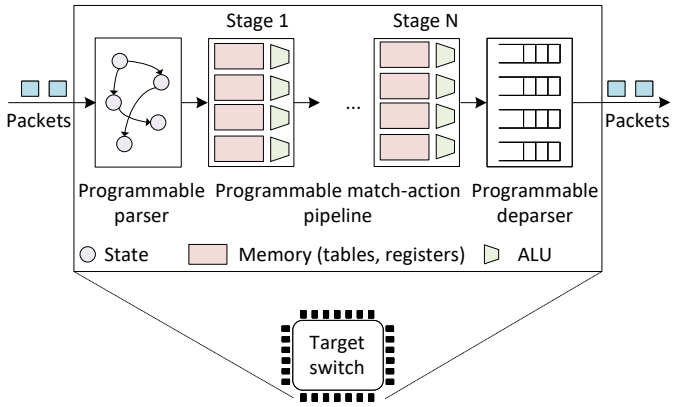

Figure 2: The PISA architecture.

planes use cases in the context of 5G SDN and Network Function Virtualization (NFV) edge. The authors present the benefits of P4-programmable devices for edge networks. Then, they present use cases and discuss research directions related to traffic engineering, in-band telemetry, network slicing and multi-tenancy, cybersecurity, and 5G VNF offloading. The authors conclude by highlighting the benefits of P4 in edge/fog and industrial applications and sustain that the flexibility of programmable data planes paves the way for massive adoption.

2.1. Scope of this Survey

To the best of the authors' knowledge, this is the first survey that focuses on the TCP enhancements using P4-programmable data planes. This survey compels works that employ P4-programmable data planes to improve or solve current issues that involve congestion control algorithms, AQMs, and P4-programmable NICs. Additionally, it presents schemes that aim at diagnosing and mitigating performance issues in the network. Each section of this survey provides an overview of P4-based schemes, highlighting key features and comparing them with legacy approaches (i.e., inter-category comparison). Moreover, this paper also discusses the limitations that result from comparing the works in the same category (i.e., intra-category comparison). Finally, this survey analyzes the most relevant aspects of current P4-based schemes and presents the challenges and future trends.

3. Background in Programmable Data Planes

The emergence of programmable data planes results from the technology evolution of SDN, which provides flexibility, enables programmability, and facilitates network automation by centralizing the network intelligence into a controller. An SDN controller keeps a global view of network topology, making network management more efficient than legacy approaches. However, SDN devices do not allow data plane programmability. They only recognize a fixed set of header fields defined by a communication protocol such as OpenFlow [26].

With P4, the programmer defines the packet headers and the forwarding behavior. Furthermore, P4 programs can run on different platforms without modifying the runtime application (i.e., the control plane and the interface between control and data planes are target agnostic). Although the technology maturity and support for P4 devices can still be considered low compared to legacy and SDN devices, programmable data planes enable operators to gain more control over the network.

3.1. P4 Architectures

Data plane architectures have been proposed to map P4 programs onto high-speed network ASICs. Examples of P4 architectures include the Protocol Independent Switching Architecture (PISA), the Portable Switch Architecture (PSA), and the Tofino Native Architecture (TNA). These architectures represent a forwarding model that can be implemented in various target devices such as software switches, hardware switches, and Network Interface Cards (NICs). The goal of a P4 architecture is to define a programming model to describe how packets are being processed. Although data plane algorithms can be characterized using programming languages such as Python or C++, they do not map well into high-speed ASICs. Thus, P4 has been proposed as a tool to describe the packet processing behavior.

3.1.1. The PISA Architecture

Figure 2 illustrates the PISA architecture. PISA is an abstraction of a packet processing model comprising the following elements:

- Programmable parser: extracts the packet header information for further processing and can be represented as a state machine.
- Programmable match-action pipeline: performs actions over the packet headers and intermediate results. A match-action stage includes multiple memory (e.g., tables, registers) and Arithmetic Logic Units (ALUs) to enable multiple lookups and actions simultaneously.
- Programmable deparser: recomposes the modified packet headers to emit them with the corresponding payload.

The relevance of using a multi-stage pipeline to process packets rather than a general-purpose single stage CPU is that processing network packets involves looking up into multiple header fields. Although a multi-stage pipeline could add processing latency, it can process multiple packets simultaneously. For instance, if a packet is being modified in stage 2 of the pipeline, a lookup is taking place in stage 1, thus enabling high-speed packet processing.

3.1.2. The PSA Architecture

P4-programmable data plane devices solve many of the SDN limitations. Over the last few years, a group of researchers developed a machine model for networking. The model is known as the Portable Switch Architecture (PSA) [27, 28] and describes the standard capabilities of a network switch. The PSA was designed with instruction sets optimized for network operations, and P4 is the high-level language for programming PSA devices.

The PSA is a P4 architecture created and developed by the P4 consortium, more specifically the P4 working group [25]. Figure 3 depicts the processing pipeline.

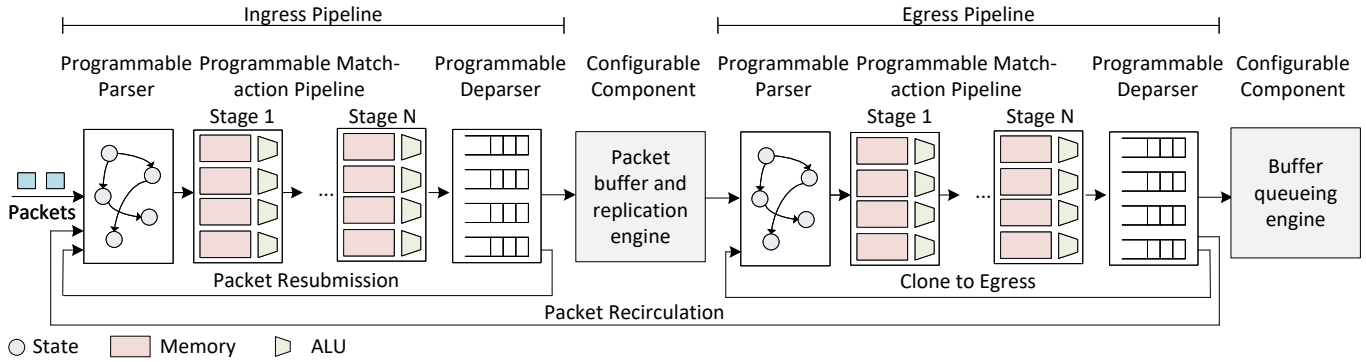

Figure 3: Scheme of the Portable Switch Architecture (PSA) [25].

The PSA also provides P4 libraries known as externs, which extend data plane features by supporting functional constructs, namely counters, meters, registers, and a set of "packet paths" that enable P4 programs to control the flow of packets in a forwarding plane. The PSA processing model comprises the following elements: parser, control blocks, and deparser. The architecture also defines configurable fixed-function components and packet primitives such as:

- Sending a packet to a unicast port: It is involved in regular forwarding operations.
- Dropping a packet: It is used to respond to events that demand regulating congestion or queue length. P4 programs might induce packet drops to regulate TCP's sending rate, avoid TCP synchronization, and improve fairness, link utilization and other metrics.
- Sending a packet to a multicast group: It can be used to efficiently send packets to a group of hosts, which can facilitate the implementation of streaming applications, multicast VPN, Content-Delivery Networks (CDN), and other data broadcasting operations.
- Cloning: It creates a copy of the packet. The original packet is independent of the cloning operation. For example, packet cloning can ensure that all packets arrive at their destination over lossy links [29].
- Resubmission: It permits reprocessing the current packet by moving it from the end of the ingress pipeline to its beginning. Resubmission is typically applied for packet re-parsing purposes. A scheme might need to parse a header field multiple times before moving it to the packet buffer and replication engine.
- Recirculation: It restarts the ingress processing of a packet after finishing the egress pipeline. It is used when packets headers contain a large number of fields that cannot be parsed at once. This primitive can also be used to reprocess queuing statistics (e.g., in approaches that implement AQMs).

Currently, vendors do not provide the compiler backend that maps the PSA architecture onto their respective ASICs. Therefore, the PSA architecture remains as a theoretical model. However, other packet processing architecture presents variations of the arrangement of the blocks present in the PSA architecture. Examples of such variation are the V1model, the Simple SUME architecture, and the Tofino Native Architecture (TNA). The latter was developed to describe Intel Tofino packet processing pipelines and presents similarities to the PSA architecture.

3.1.3. The Tofino Native Architecture (TNA)

TNA is an architecture model [30] defined by Intel for their family of Tofino switching ASICs. TNA consists of an ingress parser, an ingress match-action control block, an ingress deparser, an egress parser, an egress match-action control block, and an egress deparser. The TNA architecture model is implemented in the Intel Tofino ASIC, which processes packets at a line rate independently of the complexity of the P4 program. This increased performance is enabled by a high degree of pipelining and parallelization. Recently, Intel released the second generation of Tofino ASICs that offers more flexibility, more performance (up to 12.9Tbps), more P4 resources, and less energy consumption [31]. The performance and programmability of Tofino 2 aim at fulfilling the needs of large data centers and cloud service provider networks.

3.2. P4-programmable Devices

P4-programmable devices (i.e., P4 targets) are packet processing devices in which their behavior is described by the P4-language [6]. That means that P4-programmable devices execute an externally defined processing algorithm that differentiates them from configurable or flexible approaches [32]. P4-programmable devices allow programmers to determine the processing algorithm implemented in a network and individual packet processing nodes (e.g., hardware and software switches, NICs, and network appliances).

Depending on the purpose of these devices, they usually present variations of the PSA, which includes switches, NICs, NetFPGAs, and virtualized data planes such as P4-OVS [33] and P4rt-OVS [34] compatible hardware and virtual targets.

3.3. P4-language

P4 is a language for programming protocol-indepenent packet processors. It provides an abstract model and constructs for programming the data plane

Table 2: Comparison between P4-programmable and Non-programmable Targets [37].

Feature	**P4-programmable**	**Legacy/ Non-programmable**
Microburst Detection	Possible	Not possible
Network Feedback	Standard/Custom	Standard (e.g., ECN)
Reaction Time to Congestion	Faster	Slower
Flexibility	Higher	Lower
Telemetry and Analytics	Fine-grained	Sampled
TCP Offloading	Fully supported	Limited
Traffic Isolation	Custom	Predefined (e.g., ACL/CoS)

optimized to describe the forwarding behavior. A P4-enabled device is protocol-independent, which means that it is not tied to any specific network protocol. P4 programs allow the definition of packet headers and specifying packet parsing and processing. The P4 language is developed and standardized by the P4 language consortium [36], it is supported by various software-based and hardware-based target platforms, and it is widely applied in academia and industry.

3.4. P4-programmable Devices and TCP performance

P4 allows the programmer to define custom packet processing schemes to improve TCP and other protocols' performance. Such features allow more granular control over the network by monitoring and mitigating issues that impact TCP performance, such as congestion, low link utilization, and unfairness. P4 is also used to describe the behavior of network elements, such as hardware and software switches, NICs, and FPGA-based prototypes. P4-programmable devices have several unique features that differentiate them from legacy devices (i.e., fixed-function switches, non-programmable NICs). Table 2 compares some features of P4-programmable devices and legacy/ non-programmable devices.

3.4.1. Fine-grained Telemetry

Network telemetry is necessary for troubleshooting TCP performance problems. Programmable data planes supports In-band Network Telemetry, which allows the development of novel schemes that benefit from fine-grained telemetry data to report and take actions in the presence of network flaws. On the other hand, in legacy networks, important information is often missed due to the coarse nature of traditional monitoring schemes (e.g., NetFlow [23], sFlow [38] and IPFIX [39]). Programmable data planes can also add telemetry data to packet headers and use the metadata to infer queue latency and detect microburst. P4-programmable devices enable custom packet processing. Each packet in the pipeline is accompanied by its metadata (e.g., queue length, timestamps from different processing stages) which can be used as inputs to determine the actions taken. Moreover, programmable data planes can calculate per-flow statistics at line rate and metrics such as packet loss, RTT, and queueing delay.

The P4 language consortium became part of the Open Networking Foundation [35] since 2019.

3.4.2. Traffic Isolation

Using P4 common constructs, programmers can establish traffic isolation by slicing the network according to policies (e.g., latency, drop, bandwidth allocation, etc.) and QoS requirements. P4-targets can handle this operation with little or no performance penalty. For instance, TCP congestion control algorithms presents different variants (e.g., CUBIC, Reno, BBR, DCTCP), leading to unfairness issues when several flows share a link. The flexibility provided by match-action tables permits the allocation of TCP flows in different queues. For instance, flows can be classified according to the type of congestion control, the duration of the flow (i.e., separating elephants and mice flows), or according to user-defined policies such as prioritizing flows of an application.

3.4.3. Fast Reaction Time upon Congestion

Both programmable data planes and legacy devices process packets at line rate. With programmable data planes, developers can specify the precise actions to mitigate issues such as congestion, high latency, or poor link utilization. For instance, data planes can avoid congestion by applying fast rerouting. This approach consists of having a primary and multiple backup routes if the primary link cannot fulfill the QoS requirements imposed by the application. Rerouting can quickly adapt if the network condition changes or the operator wants to apply policy changes. On the other hand, operators must adapt predefined functions with legacy devices to mitigate the previously mentioned issues.

3.4.4. Protocol and Application Offloading

P4 targets can handle protocol mechanisms previously managed by the server's CPU. For example, the three-way handshake can be handled by a P4-programmable NIC to offload the server's CPU. Additionally, cloud applications such as microservices, can be entirely implemented in a P4-programmable NIC. TCP is a transport protocol that is implemented in the end host. Therefore, offloading some operations to a Network Processor Unit (NPU) can distribute the workloads and improve performance.

3.4.5. Microburst Detection

Switch's buffers aim to absorb the surge of packets, and thus minimize packet losses. However, legacy network devices usually do not provide good visibility of the queue dynamics, making it harder to detect, diagnose and fix network issues. Furthermore, in high-speed network environments (e.g., data centers), short-lived spikes in network traffic can lead to periods of high queue utilization. These network spikes are known as microbursts. Microbursts produce a sudden increase in the queue length within a sub-millisecond time scale exhausting the egress buffer. Microbursts detection is a functionality that is not available in legacy network devices but can be implemented using P4-programmable devices. P4 enables the development of fine-grained measurements schemes that can help to monitor the queue, detect microbursts and perform actions to counter them. The correct handling of microbursts in the network reduces the packet loss rate and contains congestion at prudent levels.

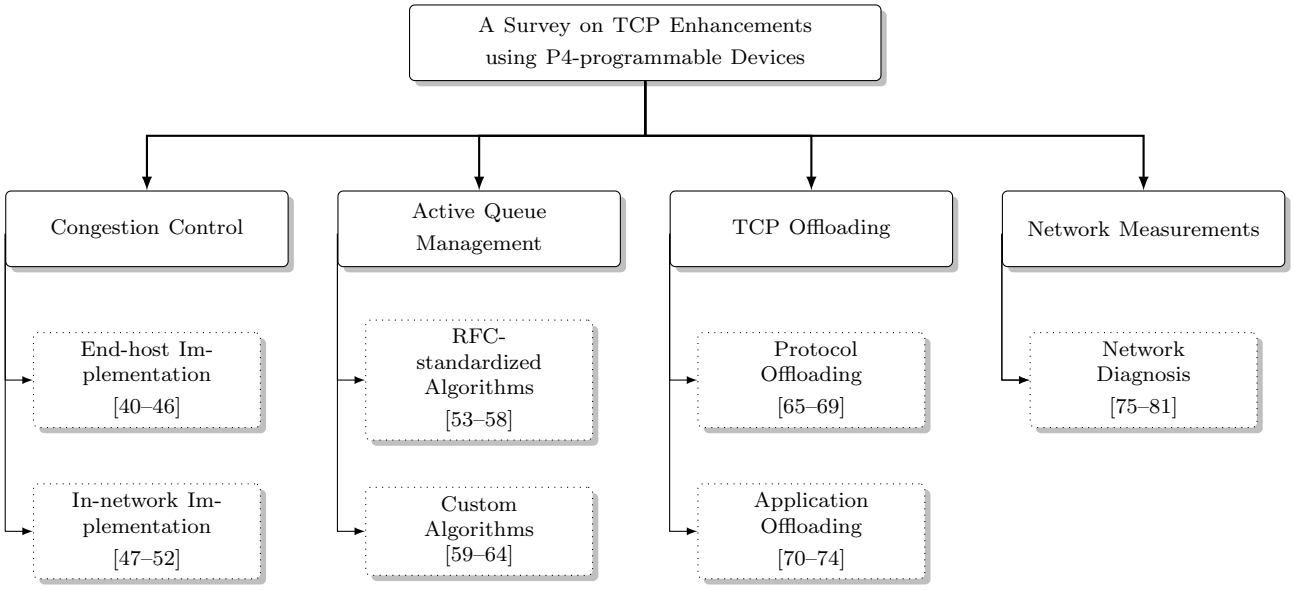

Figure 4: Taxonomy of TCP enhancements using P4-programmable devices.

4. Methodology and Taxonomy

Figure 4 depicts the proposed taxonomy. The taxonomy was designed to cover relevant works on P4-programmable data planes where the scheme aims to improve TCP performance. This work aims to categorize the surveyed works based on various high-level disciplines.

The taxonomy provides a separation of categories so that a reader interested in a specific topic can move to the corresponding section and find the necessary background to understand the section's content. Each high-level category is further divided into sub-categories. For instance, works related to "End host Implementation" correspond under the "Congestion Control" category.

Each section has a table that summarizes relevant characteristics of each work discussed in that section. Following the literature review, each section presents an inter and intra-category comparison. At the end of each section, relevant outcomes are discussed, highlighting the surveyed schemes' strengths and weaknesses. Figure 5 shows the distributions of the surveyed papers per year and category.

5. Congestion Control

Applications need a reliable rate control mechanism to send data with high performance and fully utilize the bandwidth while avoiding congestion. The TCP congestion control aims to fulfill those demands by maximizing the rate and managing how many packets to inject into the network. Congestion control algorithms aim at efficiently using network resources and minimizing congestion escalation, thus preventing the network from collapsing. P4-programmable data planes provide the means to enhance and create new congestion control algorithms. Methods such as isolating the flows' dynamic by allocating them in different queues, using telemetry information to have a better view of network events, and enhancing current TCP schemes that rely on congestion signals such as ECN, can be implemented and enhanced using P4-programmable data planes.

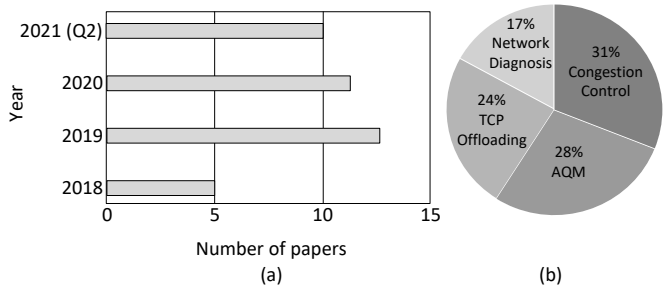

Figure 5: (a) Distribution of surveyed data plane research works per year. (b) Distribution of the surveyed paper based on the taxonomy subcategories.

5.1. Understanding TCP Issues

5.1.1. Unfair Resource Allocation

Allocating resources to match the demand of applications implies that the network takes an active role in controlling the congestion. The distributed nature of the network components (i.e., senders, middleboxes, and receivers) makes designing a resource allocation mechanism challenging. Thus, developers opted to implement the congestion control mechanism in the end host. This approach allows end hosts to decide the amount of data sent into the network and use a control mechanism to manage the congestion. However, some applications could experience an unfair resource allocation.

Congestion control mechanisms aim to ensure high throughput and low latency by controlling the sending rate. For instance, a congestion control mechanism that injects packets into the network to keep network links busy also increases the number of packets in routers' queues, resulting in an increased queuing delay. With programmable data planes, developers can implement schemes that accurately report to end hosts queueing metrics without incurring excessive overhead.

5.1.2. Unfairness among Competing TCP Flows

Congestion is an undesirable event for the network, and preventing it can be achieved by reducing some nodes

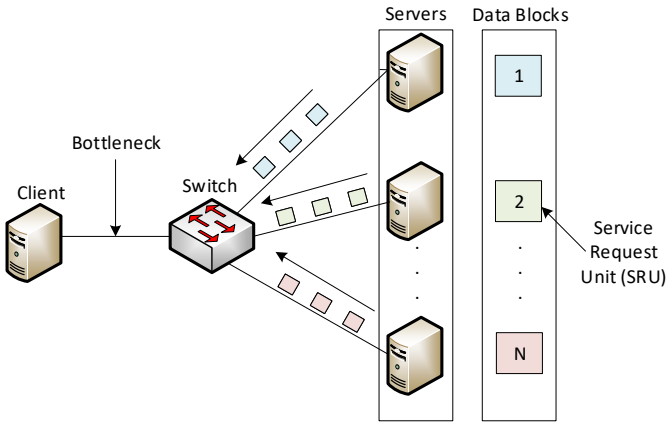

Figure 6: A cluster-based storage environment. A client request data from multiple servers through synchronized reads. Then, the incoming packets cannot be handled by the switch. Thus, TCP suffers performance degradation [82].

from sending data, thus releasing resources that other nodes can use. Another method to mitigate congestion consists of reducing the sending rate of end hosts to allow a fair resource allocation among competing participants. TCP congestion control algorithms attempt to ensure fairness. However, enforcing fairness among competing flows might require explicit information from the network, such as the number of flows using a congested link. A possible solution is implementing a reservation-based resource allocation scheme that can enforce rate policies based on the information obtained from the network. P4-programmable data planes can support schemes that collect and process network metrics in a custom manner.

5.1.3. Inaccurate Congestion Feedback

Receiving precise and timely feedback information is essential for TCP to make a good decision in controlling the sending rate. Schemes that focus on improving feedback information are usually modeled as an observation control loop (i.e., they react after observing events such as packet losses or failures). Legacy networks are limited to what they observe and what is needed to make the right control decision [83]. For example, delay-based congestion control algorithms consider delay as a congestion signal, which can be a poor indicator and produce an uncorrelated number of retransmissions. P4-programmable data planes can feed congestion control algorithms with the precise information to perform more specific actions such as reducing the sending rate based on metrics not implemented in standard protocols.

5.1.4. TCP Incast

TCP incast occurs when multiple servers concurrently send packets to a single client, blocking another server from transmitting data (see Figure 6) [84]. In such scenario, the bottleneck switch cannot handle all the incoming flows resulting in the client experiencing much lower throughput than the link capacity. TCP incast events increase flows' queuing delay and reduce application-level throughput below the link bandwidth. With P4-programmable data planes, data center operators have a granular resolution of the events occurring in the network, thus facilitating the tools to mitigate or reduce the impact of TCP incast events.

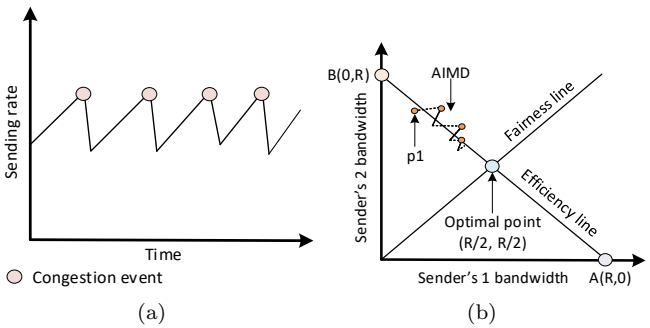

Figure 7: (a) AIMD function. The rate increases in a constant rate (i.e., additive increase) upon a loss event, which decreases the rate exponentially (i.e., multiplicative decrease). (b) Bandwidth allocation region realized by two competing TCP flows.

5.2. End host Implementations

This section describes TCP congestion control implementations that require the end hosts to participate in regulating the sending rate.

Traditional TCP congestion control algorithms are mainly end host implementations. The basic strategy of TCP congestion control consists of sending packets into the network and reacting to congestion events. This approach allows sending hosts to determine how many packets can safely traverse the network. TCP regulates the sending rate by changing the size of its congestion windows according to a function. This function dictates the variation of the congestion window size in the presence of congestion. The most popular mechanism that governs the congestion window size is called Additive Increase-Multiplicative Decrease (AIMD). Figure 7(a) shows the AIMD function used to regulate the sending rate.

Figure 7(b) explains how the AIMD mechanism helps competing flows to converge to fairness. The bandwidth allocation to flow one is on the x-axis, and flow two is on the y-axis. Suppose that TCP shares the bottleneck bandwidth equally between the two flows. In that case, the bandwidth will fall along the fairness line that starts from the origin. When the sum of the allocation is equal to 100% of the bottleneck capacity, the allocation is efficient. On point A, flow 1 receives 100% of the capacity, and on point B, flow two receives 100% of the capacity. These solutions are not desirable, as they lead to starvation and unfairness. Assume that point p1 depicts the sending rates of senders one and two at a given time. The dynamics governed by the AIMD rule will eventually guide the two bandwidths to converge at the optimal point at (R/2, R/2). Chiu *et al.* [85] describe the reasons why TCP converges to a fair and efficient allocation. This convergence occurs independently of the starting point [86].

Not all the congestion control algorithms are governed by the AIMD rule. For instance, BBRv1 [87] is a rate-based algorithm, meaning that at any given time, it sends data at a rate without considering packet losses. During the probe period (1 RTT duration), the sender probes for additional bandwidth, sending at a rate of 125% of the bottleneck bandwidth. During the subsequent period, drain (1 RTT duration), the sender reduces the rate to 75% of the bottleneck bandwidth, allowing any bottleneck queue to drain. Figure 8 shows RTT and delivery rate as a function of the inflight data. Loss-based con-

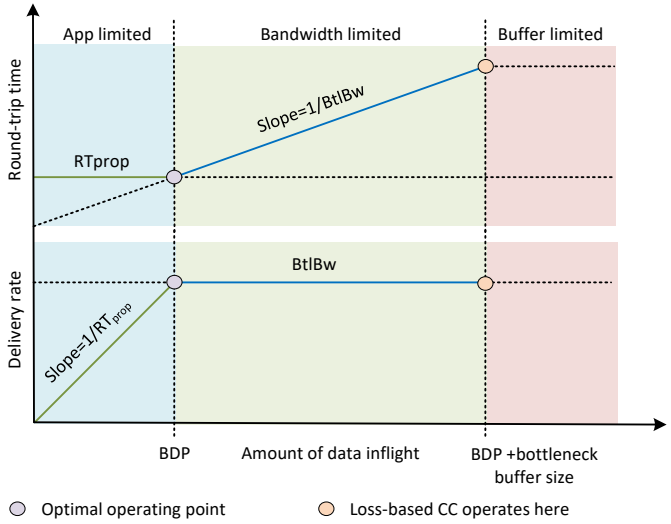

Figure 8: BBRv1 sending rate adjustment mechanism. During the probe period, the sender probes for additional bandwidth, sending at a rate of 125% of the BtlBw. In the drain stage, the sender reduces the rate to 75% of the bottleneck bandwidth, allowing any bottleneck queue to drain.

gestion control algorithms operate at the right edge of the bandwidth-limited region, using the full bottleneck bandwidth at the cost of high delay and frequent packet loss. On the left edge, it is observed Kleinrock's operating point [88]. At this point, the delivery rate is maximum, and the RTT is the optimal minimum. BBRv1 operates at this point. ECN is a TCP/IP mechanism to notify about the congestion by sending explicit feedback to the sender before congestion escalates. Dropping a packet certainly works as a signal of congestion for traditional congestion control algorithms (e.g., Reno, CUBIC), which improves the performance of long-lived bulk data transfers. However, real-time applications (e.g., videoconference, video streaming, industrial control) are affected due to delay and losses. For such low-bandwidth delay-sensitive TCP traffic, packet drops and packet retransmissions can cause noticeable delays for the user. These delays can escalate for some connections due to the coarse-granularity of the TCP timer that delays the source's retransmission of the packet [89]. Therefore, notifying the sender through an explicit notification signal is more appropriate to improve such applications' performance. Figure 9 describes the behavior of ECN. When the number of packets buffered in a switch exceeds a certain threshold, the Congestion Encountered (CE) IP header field is enabled. After the marked packets reach the receiver, the ECN field in the TCP header can notify the sender that the link is experiencing congestion. Consequently, the sender can slow down the sending rate before dropping packets.

5.2.1. Literature Review

Feldman *et al.* [40] proposed a system called Network-assisted Congestion Feedback (NCF). It controls the rate by sending an explicit congestion notification to the sender using NACKs. The system ensures a fair sharing of resources across both mice and elephant flows, requiring only short queues. The authors consider that elephant flows are responsible for congestion in the network, thus they must be separated from mice flows. NCF separates

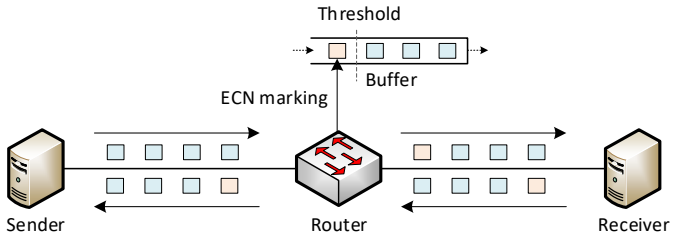

Figure 9: ECN mechanism. The switch enables the ECN field in the packet header after the congestion surpasses a predefined threshold. Then, it notifies the sender that the network is experiencing congestion before the switch starts dropping packets.

them, resulting in higher throughput and reducing the impact of TCP incast. NCF works in both data centers and Wide Area Networks (WAN). Figure 10 shows an overview of the NCF mechanism. The system classifies the traffic into two main categories, elephant and mice flow. The system applies congestion control only to elephant flows. Mice flows are not subject to any congestion control. NCF identifies elephant flows by constantly updating rolling sketches (i.e., counters). Rolling sketches are P4 data structures that, in this use case, help to dynamically identify elephants without keeping the state for all flows.

Handley *et al.* [41] presented Novel Datacenter Protocol (NDP), aiming to achieve low delay and high throughput simultaneously. NDP avoids the slow start phase and immediately sends data at the maximum rate. The method behind this scheme is called per-packet multipath load balancing, which avoids core network congestion at the expense of reordering. It also reduces the unfairness in incast situations. Incast increases the queuing delay of flows and decreases application-level throughput to far below the link bandwidth. To prevent congestion, NDP uses a per-packet multipath and trims the packet's payload in the event of queue filling when the switch's queue is increasing. Although the scheme produces some packet reordering, the receiver takes advantage of the metadata to achieve very low latency for short flows, with minimal interference between flows intended to different destinations. The authors claim that NDP improves the FCT of short flows in comparison to DCTCP and DCQCN. Under a heavily loaded network with a small switch buffer size, NDP utilizes over 95% of the maximum network ca-

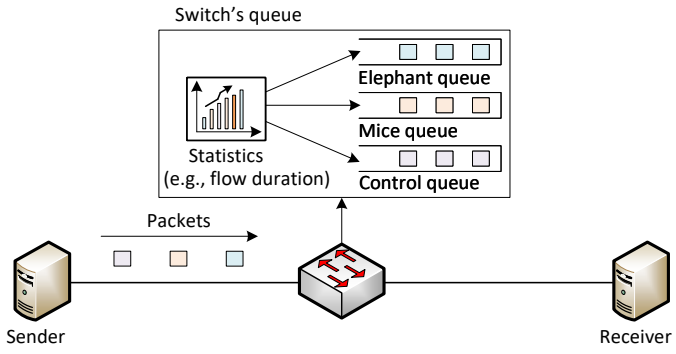

Figure 10: Overview of the NCF mechanism. The authors categorized Internet traffic into two types: 1) Elephant flows, which involve the transfer of a large volume of data and has a long duration, and 2) Mice flows, which constitute flows with shorter duration and smaller data transfer sizes.

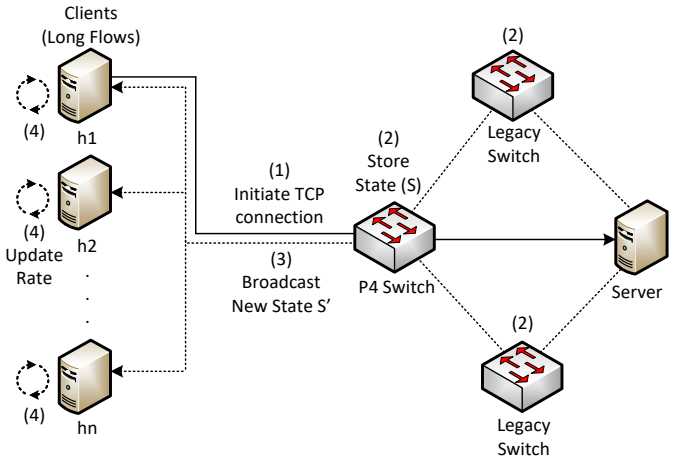

Figure 11: High-level architecture of P4-enabled pacing system. The P4 switch identifies the number of TCP flows traversing it to notify the senders how to adjust the sending rate.

pacity and lowers the delay during TCP incast.

Kfoury *et al.* [42] proposed a novel scheme based on programmable data plane switches. In this scheme, the programmable switch notifies the sender of the right TCP pacing rate, resulting in a fair share of the bandwidth. The programmable switch embeds the pacing rate in the IP options header, making it compatible with legacy devices. Figure 11 shows the high-level architecture of the system. 1) An end host inserts a custom header replacing the IP-Options header field to initiate a TCP connection. The custom header is inserted during the 3-way handshake. On the other hand, the switches parse the received packets' headers and check if the end host requests a new flow. 2) Each switch stores the latest state S of the network and inserts its bottleneck link capacity and the current total number of hosts in the SYN-ACK message. 3) Then, the switches broadcast S' to all previously connected hosts. Finally, when the hosts receive the SYN-ACK message, they adjust their transmission rates according to the IP options header information. 4) Upon receiving a TCP packet on the port used for initiating its TCP connection, the end host parses the custom header and adjusts its transmission rate by applying pacing with a rate derived from the custom header fields values. Additionally, the processing overhead is minimal, as custom packets are only generated when a flow joins or leaves the network. With input from switches, end devices are dynamically notified to adjust the pacing rate. The scheme increases throughput and enhances fairness.

Li *et al.* [43] presented a congestion control algorithm called High-Precision Congestion Control (HPCC). The congestion control is based on Multiplicative-Increase Multiplicative-Decrease (MIMD), which makes it fast, stable and responsive in the presence of packet losses. It achieves three principal goals: ultra-low latency, high bandwidth, and increased stability simultaneously in large-scale high-speed networks. It is easy to deploy and integrate with legacy networks, requiring only the activation of standard INT features. Results show a reduction of the FCT at least four times. The congestion is reduced compared to similar congestion control algorithms used in data centers such as DCTCP [92], TIMELY [93], and DCQCN [94]. Figure 12 illustrates how HPCC works. When a packet arrives at the switch's ingress interface,

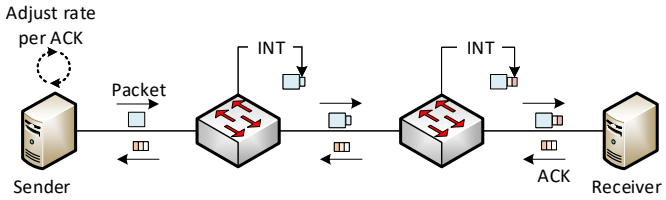

Figure 12: HPCC: INT-based High Precision Congestion Control. Each switch adds telemetry data to the packets and piggybacked to the sender within the ACKs.

the switch adds an INT header to the packet. When the TCP packet that contains data reaches the egress port of an edge switch, the INT information is piggybacked into the TCP packet that carries the ACK. The end hosts then use this information to adjust the sending rate in every ACK.

Shahzan *et al.* [44] optimized traditional ECN-based congestion notification systems, which rely on the receiver to indicate congestion. Usually, ECN-based systems wait for an RTT before the sender reacts to the congestion. The authors state that this could be an issue in networks with a high BDP. Therefore, they propose an enhanced ECN mechanism for the early detection of congestion using P4 devices. The system is called Enhanced Explicit Congestion Notification (EECN). In this approach, the sender does not have to wait for the receiver to indicate congestion. Instead, the switches notify the end hosts. Experimental results show that their system does not generate any additional traffic in the network, thus reducing the notification time compared to traditional ECN-based schemes.

Kang *et al.* [90] presented Proactive Congestion Notification (PCN), which uses Distributed Deep Learning (DDL) techniques to avoid congestion. The authors leverage DDL to collect training data from distributed workers (i.e., end hosts), thus reducing the total training time. However, experiments demonstrated that the DDL architecture induces bursty traffic. Therefore, the authors proposed PCN to prevent congestion. PCN proactively regulates the switch's queue length to prepare it when bursty traffic arrives. Results show that PCN improves the throughput of DDL traffic by 72% on average.

Laraba *et al.* [91] presented an approach for detecting and mitigating the impact of TCP misbehaviors. The scheme leverages programmable data planes to mitigate optimistic ACK attacks [95] and ECN abuses [96]. The authors propose a security monitoring system based on an Extended Finite State Machine (EFSM) to monitor stateful protocols in the data plane. Results show that the system can detect and discard optimistic ACKs at the switch without reaching the server. Consequently, a legit TCP connection does not experience performance degradation. With respect to ECN abuses, the authors reported that the scheme achieves a fair bandwidth share during an attack. They evaluated their approach in scenarios when ECN reaction is not enabled and when RED with ECN is configured in the switch.

5.2.2. End hosts Implementations Comparison, Discussions, and Limitations

Table 3 compares the schemes described in the previous section. In low latency environments, HPCC presents

Table 3: End-host Congestion Control Comparison.

	Ref	Name	Strategy	Traffic Isolation	Congestion Feedback	Custom Congestion Control	Convergence	End host Modification	Tuning Parameters	Target
Host-centric	[40]	NCF	Uses NACKs to control elephant flows	✓	NACKs	×	Fast	✓	Thresholds, queue sizes, data structures	N/A
	[41]	NDP	Trim packet headers and priority forward	✓	NACKs	✓	Fast	×	Switch buffers and Initial window	NetFPGA SUME
	[42]	N/A	Calculates the pacing rate as a function of the number of flows	×	Number of TCP flows	×	Medium	✓	Link capacity	BMv2
	Ref	**Name**	**Strategy**	**Rate Increment Mechanism**	**Congestion Feedback**	**Congestion Avoidance Mechanism**	**Integrates with Legacy**	**Reduces Packet Overhead**	**Tuning Parameters**	**Target**
Enhanced Feedback	[43]	HPCC	Computes INT metadata to set the sending rate	Multiplicative Increase (MI)	INT	Multiplicative Decrease (MD)	×	×	η, $maxStage$, $W_{AI} \cdot \eta$	Tofino
	[44]	EECN	The switch notifies the sender instead of waiting for the receiver	Def. by the End host CC	Custom ECN	Congestion Control + ECN	✓	✓	ECN threshold	N/A
	[90]	PCN	Reduces the impact of bursty DDL traffic	Def. by the End host CC	ECN	Congestion Control + ECN	✓	✓	ECN threshold	BMv2
	[91]	N/A	Mitigates TCP Optimisc ACK attack and ECN abuses	Def. by the End host CC	ECN + ACKs	Stateful processing EFSM	✓	✓	N/A	BMv2

rate dynamics dominated by the MIMD rule. It provides good link utilization and good throughput. HPCC and EECN use a feedback-based mechanism to regulate their sending rate. However, HPCC could incur overhead as each switch inserts its INT metadata. This overhead increases linearly with the path length. INT solves the overhead issue at the cost of losing accuracy but achieving better FCT. Basat *et al.* [45] presented Probabilistic In-band Network Telemetry (PINT), which bound the amount of telemetry information added to each packet. PINT handles concurrent queries while bounding the per-packet bit overhead using each packet for a query subset with cumulative overhead according to predefined policies. PINT mitigates the high bit overhead produced in schemes such as HPCC resulting in similar performance while getting good visibility. The works described in the previous section report the FCT as a metric that evaluates the performance of the schemes. The FCT is a key metric that describes how long it takes to complete a data transfer. Dukkipati and McKeown [97] demonstrated that high throughput does not lead to a better FCT. On the contrary, it could increase due to several factors (e.g., high RTT, bloated buffers, flows take different paths).

Schemes such as EECN do not require any change in the end host. Thus it can be easily integrated with current deployments. Although ECN does not provide better visibility than INT (i.e., it only notifies whether there is congestion), enabling explicit network feedback improves metrics such as the FCT, reduces packet losses, and ensures better fairness among competing flows.

NDP and NCF use NACKs as congestion feedback but with different purposes. NDP avoids congestion by applying per packet multihop load balancing. NCF dynamically tracks elephant flows and restricts them from using NACKs to notify the sender. Another difference is that NCF can be used either in data centers and Internet-wide deployments, although there are no implementation results to evaluate the solution's effectiveness. Moreover, NCF solves the incast problem by separating the elephants from mice flows. NDP's principal limitation is the high retransmission rate and the moderately expensive resources allocated to implement the protocol. NCF sends network congestion information through the NACKs. The latter might produce unnecessary overhead, thus making it less attractive to operators. NDP and NCF require operators to manually tune predefined parameters such as thresholds, queue size, maximum latency, etc. Also NCF and NDP perform traffic isolation. The difference is that NDP requires end host modification to interact with a custom header, while NCF does not. NDP presents hardware implementations on different targets and customization, such as in [74]. On the other hand, NCF is a proposal that requires further testing in P4 targets.

The method in [42] uses TCP pacing. Pacing smooths throughput variations and traffic burstiness, maximizing the link utilization and minimizing queuing delays. A drawback of this method is that it produces unnecessary overhead when the number of flows is high. Also, the pacing approach by itself often has significantly worse performance than regular TCP due to the synchronization with packet losses [98]. Therefore it can only be suitable for deployments that involve a small number of large flows (e.g., in Science Demilitarized Zone (Science DMZ) networks [99]).

Table 4: Comparison between P4-based End host Implementations and Traditional Congestion Control Implementations.

Characteristic	P4-based Implementation	Traditional Implementation
Accuracy	High; operating at sub-millisecond scale	Medium; control plane intervention adds a processing penalty
Convergence	Fast; MIMD and custom protocols	Slow; Traditionally AIMD
Queue monitoring	Fine-grained	Coarse-grained
Fairness	High; several scheme aim at improving fairness and other metrics	Low; limited options to ensure fairness
End host Modification	Required in many schemes	Not Required; Traditional protocols are usually present in the end host
Congestion information	Many; ECN, INT standard and, custom metadata	Few; ECN

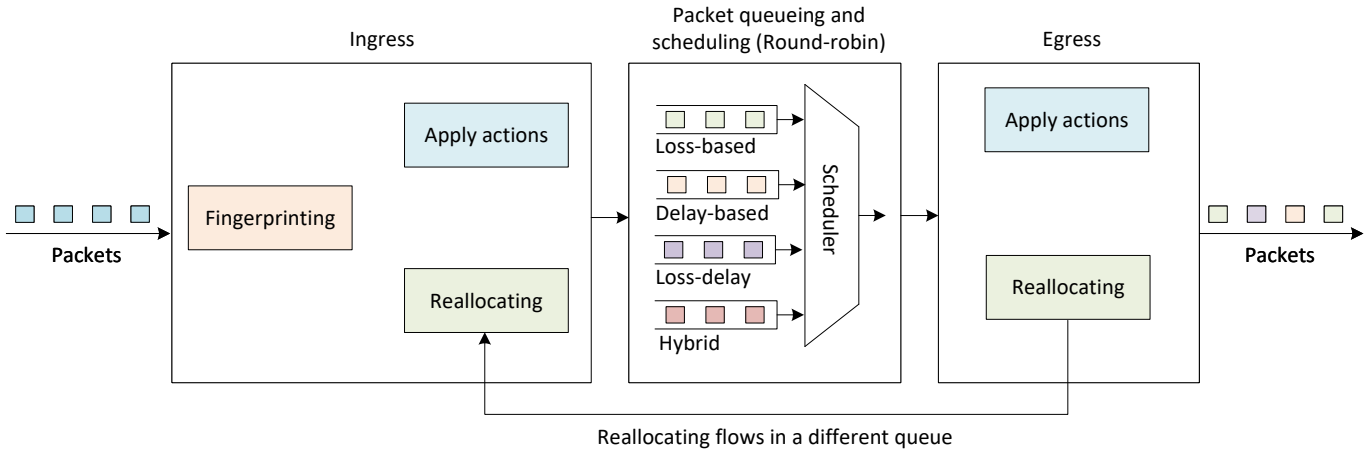

Figure 13: The system comprises three modules: The fingerprinting module groups the flows according to their congestion control algorithm. The reallocation module redistributes the queues between groups when it is required, and the apply actions module applies custom policies to ensure fairness among flows [47].

5.2.3. Comparison with Legacy Approaches

Table 4 compares the main characteristics between end hosts implementation congestion control schemes with legacy. P4 helps developers to improve TCP feedback by including standard (e.g., INT, ECN) and custom (e.g., pacing rate in the IP options) information into packets. On the other hand, legacy schemes only work with standard headers, which makes them less flexible.

ECN approaches experience moderate convergence times as they rely on a single bit to notify congestion without providing any information about the level of congestion or the congested link location. On the other hand, programmable switches can give better feedback to indicate congestion (e.g., INT metadata in HPCC), which results in higher detection accuracy, near-zero queueing delays, and faster convergence time. However, traditional congestion control algorithms' distributed nature allows them to operate without modifying the network infrastructure or the end hosts.

5.3. In-network Implementations

This category describes the implementations where P4 switches are used to modify the dynamics of the traffic and perform actions to reduce congestion (e.g., rerouting).

Many authors proposed schemes to reduce congestion as a response to the continuously increasing demand for higher bandwidth, lower latency, and higher reliability. P4 switches facilitate the implementation of such schemes, providing the programmer the complete control of the scheme's design. With high-speed networks, developing a good congestion control algorithm depends on how timely and accurate the actions to control congestion are. Therefore, a precise observation loop must complement the control loop [83]. P4 switches provide the tools to extend congestion control algorithms and better understand the events in the network. Traditional congestion control identifies losses as a signal of congestion. However, other feedback signals contribute to improving the description of the congestion event (e.g., ECN, INT header, custom headers). With P4-programmable data planes, developers can define and manage how these signals are interpreted and create custom mechanisms to react against a broader range of congestion events. ed to lower delays, maximize throughput, and the incast problem (e.g., [92, 100]).

5.3.1. Literature Review

Turkovic and Kuipers [47] proposed P4air, a method that classifies the traffic based on their congestion control algorithm. The flows are allocated in different queues to isolate them from the dynamic of the other congestion control algorithms, therefore improving inter-protocol fairness and the network resources utilization in the presence of a large number of flows. Based on a previous study, Turkovic *et al.* [101] observed the interaction among different congestion control, dividing them into categories, namely loss-based algorithms, delay-based algorithms, and hybrid algorithms. Results remark that a good TCP congestion control algorithm is not enough to achieve low latency, high performance and optimize the link utilization. The authors recommend grouping the flows according to their congestion control algorithm to ensure the expected resource optimization. The system actively monitors the characteristics of the flows that pass through a switch. Then, the switch groups the flows based on the type of congestion control (e.g., Loss-based, Delay-based, Loss-delay, Hybrid). Figure 13 illustrates the components of the scheme, which is divided into two subsystems: 1) A fingerprinting or identification process that identifies the congestion control algorithm considering the metadata of every packet, such as timestamps from different stages of processing, queue depth. 2) A stateful processing that tracks the reactions of the flows in the occurrence of certain events, such as loss or an increase in queue size. The scheme aims at enforcing fairness by combining these two subsystems. P4air identifies the congestion control algorithm by analyzing packets' metadata and performing actions (i.e., dropping packets, delaying a packet, changing the advertised congestion window in the TCP header) and observing the reaction of TCP flows. The system uses recirculation to reallocate packets after being enqueued to put them in a different queue if needed.

Apostolaki *et al.* [49] presented a dynamic buffer sizing technique called Flow-Aware Buffer (FAB) sharing. The system addresses the problem of having multiple flows allocated in different egress ports. The algorithm

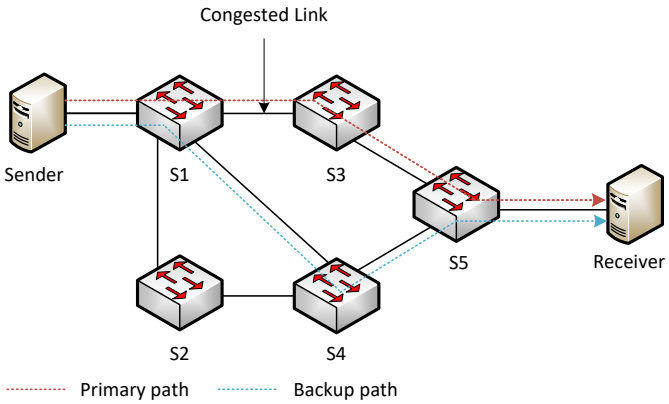

Figure 14: Simplified Fast-TCP mechanism overview. The scheme has a primary path and many backup paths. Upon the failure, it evaluates the weight of the backup paths to reroute the traffic.

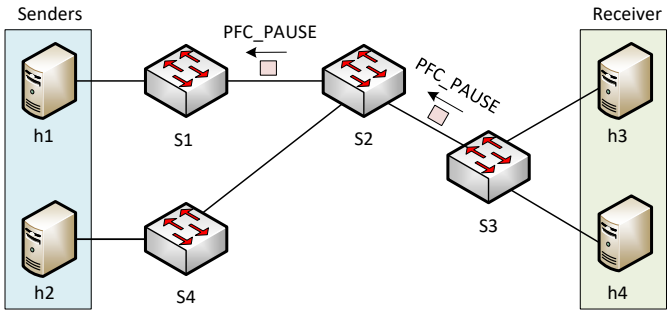

Figure 15: Priority Flow Control (PFC) head-of-line blocking. An excessive queue length in switch S3 produce a PFC_PAUSE signal that will hold all the traffic passing through switch S2. P4QCN mitigates this issue by implementing an improved version of the QCN standard using P4 switches.

assigns the flows according to their destination IP addresses to make more efficient use of the ASIC's memory. Moreover, it dynamically adjusts the buffer size for each output queue, which shows a considerable reduction of the packet loss and latency and improved FCT. The authors remark that allocating buffers across ports is not trivial. Therefore, choosing the resource sharing algorithm can alter metrics such as the throughput (e.g., when TCP incast occurs). Traditionally, routing devices allocate buffer space considering the available memory, but they do not consider the traffic type. Results show improved FCT using Tail-Drop (TD) by order of magnitude compared to conventional buffer management techniques.

In another work, Turkovic *et al.* [50] developed a congestion avoidance method that uses packets' metadata to track processing and queuing delays of latency-sensitive flows. The system is called Fast-TCP, and the main idea is rerouting traffic across a backup path when congestion occurs. Figure 14 depicts a scenario when the link between S1 and S3 is congested. S1 already has a backup path to reroute all the traffic to the same destination in such a situation. The authors created a P4 program to detect the congestion using digests to notify the participants when the delay surpasses a certain threshold. Experiments conducted on real hardware show decreased average and maximum delay and jitter compared to legacy approaches. This system aims to improve the QoS of time-sensitive applications such as audio, video, and haptic. The last one consists of transporting the sense of touch over the Internet. Such applications require very low latency ($\sim 1ms$), low jitter, high bandwidth, and high reliability.

Benet *et al.* [46], presented a scheme to load-balance Multi-Protocol TCP (MPTCP) sublfows. The scheme is called MP-HULA and associates MPTCP subflows to MPTCP connections by parsing MPTCP header fields. MP-HULA solves the particular case of load-balancing MP-TCP traffic by using the HULA data plane algorithm [102]. MP-HULA comprises an adaptive probing mechanism that keeps congestion state for the best-k next hops per destination. The system then uses the congestion state to route different MPTCP subflowlets towards different paths. Flowlets are packet bursts used to split TCP paths without causing packet reordering. Results show the effectiveness of MP-HULA to reduce the FCT 1.7× to HULA with MPTCP and uncoupled congestion control. In comparison to TCP, MPTCP combined with MP-HULA obtain from 2.9× to 3.4× reduced FCT.

Geng *et al.* [51] proposed a congestion control protocol that can alleviate the problems caused by the Priority-based Flow Control (PFC) mechanism. PFC is a link-level flow control mechanism that allows operators to selectively pause traffic according to their class, aiming at preventing packet losses. The mechanism consists of pausing the transmission when a queue exceeds a threshold. Once the queue length is restored, the transmission resumes. Figure 15 explain the Head-of-Line (HOL) blocking issue. Consider switch S3. When congestion occurs (i.e., the queue size of the ingress port exceeds a threshold), the PFC algorithm will pause the transmission by sending a PAUSE notification to switch S2 and consequently to switch S1 producing a cascading effect. Because PFC will stop all traffic from host h1 to host h3, traffic from host h2 to host h4 is also paused, even if the destination port that connects switch S3 to host h4 is not congested. Thus, the cascade effect of the PAUSE frame can also escalate the congestion.

This problem usually occurs in data centers when applications that involve large-scale online data-intensive services use PFC and Remote Direct Memory Access (RDMA). Therefore, the overhead produced by the PFC can lead to performance degradation, the spread of congestion, and even a permanent loss of communication. The authors proposed a system called P4QCN to mitigate the HOL blocking issue produced by PFC. P4QCN is a congestion control scheme that uses programmable switches to implement an improved version of Quantized Congestion Notification (QCN), a flow-level, rate-based congestion control mechanism. The authors implemented the QCN standard in P4 to make it compatible with the IP-routed network. Results show that P4QCN achieves the expected performance in terms of latency and throughput while mitigating the cascade effect produced by the HOL blocking issue. Results show that P4QCN reduces the packet loss rate and latency when congestion occurs. P4QCN also achieves a better bandwidth utilization than PFC under the same conditions (e.g., same thresholds, packet losses).

Jinhao and Yue [52] proposed a novel mechanism to regulate the size of the advertised TCP congestion windows. The authors argue that the current calculation of

Table 5: In-network Congestion Control Comparison.

	Ref	Name	Strategy	Uses Approx.	Traffic Information	Shared Buffer	Header Modification	Convergence	Tuning Parameters	Target
Traffic Isolation	[47]	P4air	Classifies flows according to their congestion control algorithm	✓	Congestion Control	✓	✓	Variable	None	Tofino
	[48]	N/A	Classifies traffic according to their QoS requirements	×	App ID	N/A	✓	Fast	None	Tofino
	[49]	FAB	Allocates buffer space as a function of the traffic	✓	Flow Duration	×	✓	Fast	Dynamic threshold	N/A
	[46]	MP-HULA	Determines the best load-balancing strategy to distribute MPTCP flows	×	Number of MPTCP subflows	×	✓	Variable	None	ns2
	Ref	**Name**	**Strategy**	**Control Plane Intervention**	**Congestion Event**	**Custom Header**	**Header Modification**	**Convergence**	**Tuning Parameters**	**Target**
Fast Rerouting	[50]	FastTCP	Reroutes traffic reacting to latency degradation	✓	Packet loss	×	×	Fast	Detection threshold ($t_p + t_q$)	Tofino
	[51]	P4QCN	Extends the QCN protocol to IP-routed networks	×	Head-of-line blocking event	✓	×	Fast	Q_{PFC}, Q_{P4QCN}	BMv2
	[52]	AAW	Adjusts the advertised TCP congestion window using INT data to calculate the network capacity	×	Packet loss	×	✓	Medium	ECN threshold	N/A
	[103]	PL2	Achieves low latency and high utilization by scheduling the transmission time in the switch	×	Packet loss	×	✓	Medium	ECN threshold	Tofino

the advertised window in the TCP header is inaccurate because the source node does not know the actual capacity of the network. The system is called Adjusting Advertised Window (AAW), and it dynamically updates ACK packets' advertised window to feedback the network capacity indirectly to the source nodes. Each P4 switch calculates the new AAW value and writes it into the packet header. AAW only requires the intermediate node to modify the advertised window size. Therefore, the scheme is compatible with existing congestion control implementations.

Le *et al.* [103] presented Predictable Low Latency or PL2, a rack-scale lossless scheme that achieves low latency and high throughput, which is independent of the workload and the transport protocol. PL2 reduces end-to-end latency by proactively avoiding losses by implementing a protocol where the end host schedule a timeslot to start a data transfer. PL2 is transport-agnostic and capable of accommodating flows at 100Gbps line rates. Additionally, the scheme does not require any prior knowledge regarding the workload characteristics, given that it is hard to anticipate traffic and workload patterns. Figure 16 illustrates PL2's behavior. In this example, the sender is sending a packet burst to the receiver. 1) The sender starts sending an RSV (i.e., reserve) message to the switch to schedule a reservation. 2) The switch keeps time slot reservations for every host's input and output ports. It reserves the earliest available timeslot on the input port of the sender (i.e., T=4) and the receiver's output port (T=5). 3) The switch notifies the available timeslots to the sender with GRT (i.e., Grant) message. 4) The sender then transmits at the maximum of the two timeslots indicated in the GRT, T=5. The authors evaluated PL2 using several transport protocols. Results show that TCP flows experience low latency (i.e., <25us) and high throughput even in noisy scenarios which involve background traffic.

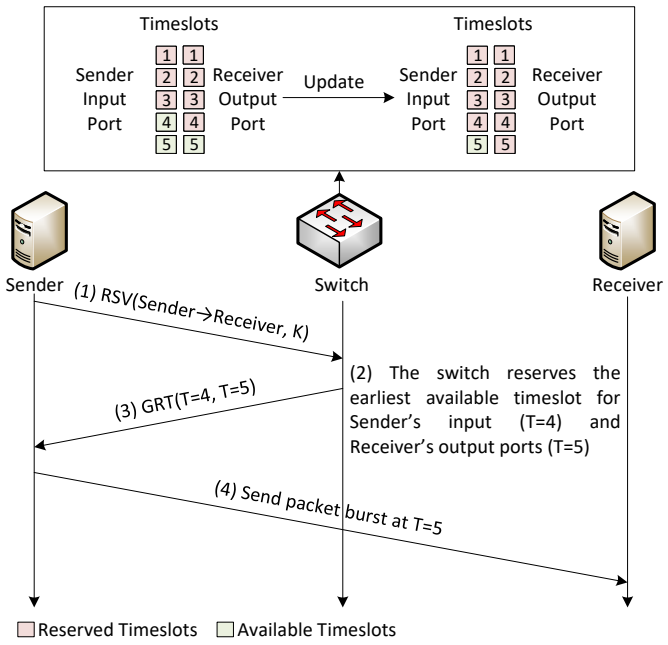

Figure 16: PL2 algorithm [103]. The switch manages the reservations to avoid collisions. PL2 reduces end-to-end latency by proactively avoiding losses by implementing a protocol where the end host schedule a timeslot to start a data transfer.

5.3.2. In-network Implementation Comparison, Discussions, and Limitations

Table 5 compares the schemes described in the previous section. P4air achieves significant improvements in fairness by applying traffic separation in comparison to current solutions. However, it requires allocating a queue for each congestion control algorithm group (e.g., loss-based (CUBIC), delay-based (TCP-Vegas), Hybrid (BBR), etc.). The latter might pose an issue due to the limited number of queues in switches, mainly used for QoS applications. A similar solution presented in [49]. FAB separates the traffic into multiple queues according to the destination IP. In this way, the system performs a

Table 6: Comparison between In-network Implementation Congestion Control and Traditional Approaches.

Characteristic	P4-based In-network Implementations	Legacy Implementations
Per-port Buffer Allocation	Dynamic; the user defines the way to load and write the buffer [49]	Fixed; a port cannot access the remaining buffer of another port
Traffic Isolation	Flexible; P4 enables the precise identification of flow characteristics	Restricted; vendors define traffic processing [104]
Flexibility	High; custom packet processing	Low; the close nature of legacy hardware restrict several functionalities
Security	Medium; attackers can exploit bugs or unintentional behaviors defined in the P4 program	Medium; operators follow vendor-defined security guidelines
Resource Utilization	Flexible; programmers can determine resource allocation	Predefined; vendors define the resources allocated to each protocol

better usage of the ASIC's memory. Another difference is that FAB dynamically adjusts the queue size resulting in a significant reduction of packet losses.

The method proposed in [48] classifies the traffic according to its QoS requirements. Experimental results show that the proposed design effectively limits the maximum allowed rate and guarantees each flow's minimum bandwidth. However, this approach does not consider more than three traffic categories (i.e., guaranteed, best effort, drop), limiting flexibility as network traffic becomes more heterogeneous.

P4QCN presents a congestion feedback mechanism that enables switches to check the egress port for congestion before forwarding packets. Fast-TCP reacts in the presence of congestion by rerouting traffic through a backup path. In comparison to Fast-TCP, P4QCN reacts faster before congestion escalates. Both schemes require establishing predefined parameters such as thresholds, maximum queue length, congestion sensitivity, etc. However, Fast-TCP suffers from some performance penalties due to disabled caching and the lack of information about the queueing delay.

5.3.3. Comparison with Legacy Approaches

Table 6 compares the main characteristics between P4-based and legacy in-network implementation congestion control schemes. P4-programmable data planes provide the tools to implement schemes that use various network metrics (e.g., queue length, packet losses, sequence, and acknowledgment numbers pairs and the metrics that can be inferred from the packet's header). Such inputs allow the programmer to have a better view of the network and perform more accurate actions to reduce congestion. Legacy devices are restricted to fixed-function, making it difficult to arbitrarily parse and modify TCP headers (e.g., AAW) or implementing custom congestion control mechanisms. Legacy devices can perform traffic separation using QoS features defined by the vendor. However, it requires configuring the QoS policies, which are limited to few categories. For example, separating real-time traffic such as video streaming from bulk data transfers can be performed using legacy devices. On the other hand, separating traffic according to its duration (e.g., splitting elephant from mice flows) or according to its congestion control is a more challenging task using legacy devices.

Moreover, the per-port buffer allocation observed in legacy devices does not consider the type of traffic. Therefore, this behavior resembles a dynamic buffer management technique limiting the buffer each queue can use to a fraction of the unused buffer size. Employing programmable data planes, the user can create a program that splits buffer cells among ports of a device while considering all ports' utilization and the expected benefit of buffering for each flow.

5.4. Summary and Lessons Learned

According to Li [83], a well-designed congestion control mechanism must guarantee adequate link utilization, including low-latency, near to zero-queue, robustness, and fast convergence. The capability of P4-programmable devices to perform packet processing at a line rate enables the design, test, and evaluation of precise congestion control algorithms. Moreover, programmable switches allow the development of custom congestion control algorithms that meet specific requirements such as isolating the dynamics of TCP flows, reducing latency, achieving better link utilization, and improving FCT. Congestion control schemes that use telemetry information to regulate the sending rate demonstrated reacting a the sub-millisecond level, making them suitable high-speed network environments such as science DMZ and data centers.

A limitation of INT-based schemes is that appending telemetry information reduces the payload size. There are new schemes that aim at reducing such overhead [45]. Future work should focus on incremental deployment (i.e., inter-networking with legacy schemes) to allow more experimentation and mitigate current network issues. Schemes that perform traffic isolation present favorable results in managing the congestion. Such schemes assume that competing flows have different congestion control algorithms, which leads to unfairness problems. For instance, if a flow releases throughput (e.g., due to a congestion signal) then, another flow with a congestion control with faster adaptation can occupy the freed resource, leading to an unfair resource allocation.

6. Active Queue Management (AQM)

The standard TCP strategy consists of regulating the sending rate once congestion occurs. Most of the traditional TCP congestion control mechanisms consider packet loss as a signal of congestion. A practical approach consists of avoiding congestion by informing the sending host that the network is close to experiencing congestion. This approach is implemented by AQMs, which consider that network switches are in a key position to detect and manage congestion (i.e., they know how much congestion the network is experiencing [105]). Although incorporating new features to the switch was not the preferred way to introduce improvements, programmable data planes can change this paradigm.

AQM algorithms cooperate with end host protocols such as TCP to manage congestion by increasing network utilization and limiting packet loss and delay. Nadas *et al.* [106] demonstrated that delegating all the management of the congestion to the congestion control algorithm is not suitable for heterogeneous networks. Therefore, congestion control algorithms are modeled by default without

considering feedback from the routers. Moreover, TCP congestion control algorithms assume that the switch implements a FIFO queueing discipline and TD as the dropping policy. AQM algorithms operate within the routers along the path by taking action such as dropping packets, allocating flows into different queues, and managing the buffer spaces used to store packets. Therefore, the cooperation between TCP and AQM is the best treatment for congestion. However, despite such a relationship, TCP congestion control must understand the feedback information provided by switches.

6.1. Challenges on Designing AQM schemes

6.1.1. Understanding Queues

Understanding the cause and meaning of queues is an important step to develop an effective AQM. The reason why network buffers should be employed is to smooth packet bursts. In other words, they act as shock absorbers that convert bursty packet arrivals into smooth, steady departures. Queues appear in buffers due to short-term mismatches in traffic arrival and departure rates, producing standing queues that cannot dissipate. The persistent problem of having a long-standing queue is also known as the bufferbloat problem. Standing queues creates considerable delays without improving the throughput. Usually, such an event is interpreted as congestion by traditional congestion control algorithms. BBRv1 and BBRv2 maintain a low queueing delay. However, in a recent study, Mishra *et al.* [107], measured that the Internet traffic is dominated by CUBIC (i.e., 36%), which is being known to produce bufferbloat [108].

6.1.2. Dropping the Right Packets

Undersized buffers do not solve the bufferbloat issue, [109, 110] instead, they increase the packet loss rate. AQM algorithms solve the issue by discriminating which packets are suitable to be dropped thus, making a more efficient use of the switch's buffer. According to Nichols and Jacobson [111], high RTTs (i.e., ∼1 second) are caused by bloated buffers and are not related to the path characteristics. Therefore, designers should consider these variations to design a robust algorithm that can dynamically manage switch buffers. For example, the TD queueing discipline drops packets as a consequence of a full queue. AQMs such as RED [8], ARED [112], WRED [113] require constantly adjusting tuning parameters as the network condition changes. Novel AQMs such as CoDel [111], and CAKE [11] do not require tuning any parameters and were designed to adapt to changing network conditions. For instance, CoDel AQM drops packets that exceeded the standing queue interval (default value target=∼ 5ms). P4-programmable data planes comprise primitives and constructs to implement schemes that identify which packets are more suitable for dropping.

6.2. RFC-standardized Algorithms

The following sections discuss the RFC-standardized AQM algorithms, which consist of implementing schemes that are standards defined by the Internet Engineering Task Force (IETF) (e.g., RED, CoDel, PI, PIE).

Traditional TCP congestion control presents a problem of adjusting the transmission rate only after a packet loss event. Therefore, there is a long time between a packet loss event and when the sender is notified about the loss [114]. During this period, the end host continues transmitting at the same rate that the network cannot support thus, producing even more packet losses. AQMs aim at alleviating packet losses by notifying the sender about the incipient congestion. Therefore, end hosts can reduce their sending rate before the switch's queue overflows.

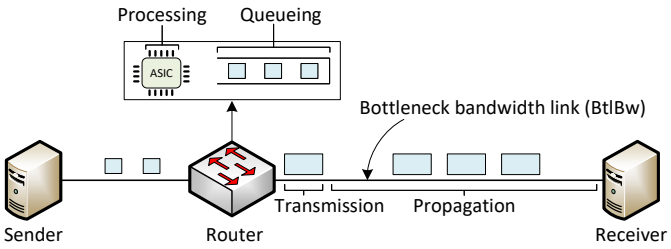

Figure 17: Delay components: processing, queueing, transmission, and propagation delays. The queueing delay is the one that has a significant impact on TCP performance.

When a packet traverses the network, it might encounter four types of network delay, Figure 17 illustrates the types of network delays and where they occur.

- *Processing delay*: the time taken by the router to process a packet as it traverses through the parser, ingress and egress pipelines, and deparser.
- *Queuing delay*: the amount of time a packet is waiting in a queue until it can be transmitted.
- *Transmission delay*: the time required to put all the bits of a packet in the wire.
- *Propagation delay*: it is given as a function of the physical distance and propagation speed of a link.

The processing and transmission delay has little impact on the network delay. On the other hand, the propagation delay depends on the type of network (e.g., LANs, WANs). The queueing delay is a component that can significantly increase the delay that a packet experience across the network. Gettys and Nichols [108] reported the presence of unnecessarily large buffers on the Internet that contribute to the queuing delay. Moreover, they also mention that routers that have an AQM enabled are rare.

The buffer size directly impacts the queueing delay. The buffer size directly impacts the queueing delay [115–117]. The rule of thumb proposed by Villamizar and Song [116] suggest that the amount of buffering (in bits) in a router's port should be greater or equal than the average Round-Trip Time (RTT) (in seconds) multiplied by the capacity C (in bits per seconds) of the port (i.e., $B \geq C \times RTT$). The resulting buffer size will ensure that the link is fully utilized with a small number of TCP flows. However, Appenzeller *et al.* [117] demonstrated that large static buffer sizes produce unnecessary high latency leading to a problem known as bufferbloat. They claimed that the buffer size could be reduced by a factor of the square root of the number of flows N (i.e., $B \geq C \times RTT/\sqrt{(N)}$ when the number of TCP flows is large. In this way, the link is fully utilized and the latency reduced.

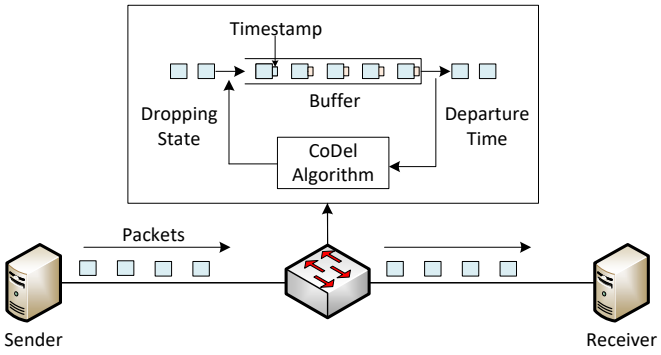

Figure 18: CoDel scheme and algorithm. 1) CoDel uses timestamps to determine when a packet should be dropped. 2) The algorithm actively measures the queueing delay to determine if a packet will be dropped.

The AQM algorithm adopted in many routers is TD, in which the maximum length of each queue is a fixed value. The packets from all sender are enqueued until the queue is full. All other incoming packets that found the queues full will drop. However, this approach presents several problems. For instance, TCP can rapidly fill the queue, causing a high loss rate to other flows.

Additionally, TD increases the delay since the queue can be persistently full. RED [8] has been an improvement of TD. RED drops packets according to predefined thresholds. With the increase of the queue length, the probability of dropping packets also increases. However, RED only considers the queue length as an indication of congestion. It does not infer the number of flows sharing the bottleneck link, and it requires constant parameter tuning to achieve good performance. The latter is a problem when the network condition changes.

CoDel [118] was proposed as an improvement of the existing AQMs. CoDel aims to control the queue length by dropping packets as a function of the sojourn time. If the queue length exceeds a threshold value (i.e., target) for some time (i.e., interval), CoDel starts dropping packets until the queue length settles below the threshold value. CoDel has been designed to work across a wide range of network conditions requiring little to no tuning.

Fair-queueing Codel (FQ-CoDel), [119] mitigates the unfairness problem resulting from having multiple flows sharing a single queue. FQ-CoDel uses a hybrid approach that consists of two queueing disciplines: Fair-queueing and CoDel. This approach separates the dynamic of competing flows by allocating them in different queues. PIE AQM [9] is a mechanism that controls the reference queue latency to a reference value based on the Proportional-integral (PI) controller. It aims at providing low latency control, high link utilization, simple implementation, stability, and fast responsiveness. PIE does not require any tuning parameter, making it robust and optimized for various network scenarios. The recently proposed CAKE [11] AQM controls the queue latency by using a rate-based shaper. This mechanism schedules the packet transmission at a precise interval using a virtual transmission clock. Similarly, CAKE does not require any tuning parameter.

6.2.1. Literature Review

Kundel *et al.* [53] implemented CoDel queueing discipline on a programmable switch. CoDel significantly reduces the latency of TCP connections and mitigates the bufferbloat problem. They also demonstrated that it is possible to implement AQMs in programmable data planes using approximation techniques. Figure 18 depicts the CoDel scheme that computes each packet's timestamp to establish the dropping policy. The algorithm considers three situations: 1) If the queueing delay is below a threshold, a packet is never dropped. 2) If the queuing delay exceeds the threshold for more than a predefined interval, packets will be dropped. 3) Packets will be discarded until the queueing delay is below a threshold. The same authors extended this work by implementing the CoDel algorithm in different P4-programmable data plane hardware targets [54]. They also discuss the challenges of implementing CoDel in different targets, usually limited by hardware constraints. Evaluations compare the performance of the CoDel algorithm running in the Linux Kernel, Intel Tofino ASIC, and Netronome NPU-based SamrtNIC.

Kunze *et al.* [55], recently implemented and tested three versions of the PIE AQM in a P4 Tofino target. They show the inherent trade-offs of all the implementations and the impact on the performance. Precisely, the mismatch between AQM design and the capabilities of the Intel Tofino switch. The authors argue that conceptual challenges make it impossible to implement a fully RFC-compliant PIE version on Tofino. Also, the authors found that transferring RFC schemes to resource-constrained P4 targets is a non-trivial task.

Papagianni and De Scheppe [56] implemented Proportional Integral PI^2 AQM in a programmable data plane. PI^2 is an extension of PIE AQM described in [120] that smoothes the non-linearities of the original PI controller. The main goal is to support the coexistence between classic and scalable congestion controls algorithms. The authors implemented the system in a P4 software switch (BMv2) and verified the design through a proof of concept implementation. Experimental evaluations report when the link capacity decreases, the queue latency is 10x slower, resulting in the TD case in a 10x higher delay (e.g., up to 200ms). On the other hand, the PI^2 controller keeps the queueing delay at the target levels. Furthermore, the authors argue that there is no need to tune the AQM parameters if the network condition changes.

Toresson [57] presented a packet value-based AQM using programmable data planes. It uses a combination of the PIE AQM scheme, and the Per-Packet Value (PPV) [121] concept to determine the dropping policy to reduce the queuing delay. PPV establishes the resources sharing policies by marking packets with packet values. This value determines the relative importance of one flow over another (e.g., different flows might have different throughput or delay requirements). The system was implemented using the Barefoot Tofino ASIC. Packet value statistics are collected through the P4 programmable data plane to maintain knowledge about pack- et value distribution. With the dropping probability calculated through the PIE AQM scheme, a decision can be made about which packets should be dropped. An evaluation shows that with the implemented PPV AQM, a low queu-

Table 7: Comparison of RFC-Standardized AQMs Schemes.

Ref	Name	Strategy	Thresholds	Dropping Policy	Multiple Queues	Approach	Queueing Discipline	Tuning Parameters	Target
[53, 54]	P4-CoDel	Implementation of RFC-standardized CoDel	3	Timeout	×	Heuristics	FIFO	Default Thresholds	Tofino
[55]	P4-PIE	Implements three variants of PIE AQM	5	Probability	×	Control Theory	FIFO	Default Thresholds, Sampling Rate	Tofino
[56]	PI^2	Implements PI^2 algorithm using bit manipulation	2	Probability	×	Control Theory	FIFO	α, β	BMv2
[57]	P4-PPV	Employs PPV to make drop decisions when queueing delay increases	2	Packet Value	×	Heuristics	FIFO	CTV	Tofino
[58]	AFQ	Approximates fair queueing using data plane constructs	1	Tail Drop	✓	Control Theory	Fair Queueing	None	Cavium Octeon

ing delay can be achieved by dropping an appropriate amount of packets. It also shows that the PPV AQM controls the resource-sharing between different traffic flows according to a predefined marking policy.

Sharma *et al.* [122] implemented a queueing discipline called Approximate Fair Queueing (AFQ) entirely in the data plane. The algorithm is based on the Fair Queueing discipline that ensures a fair bandwidth allocation for each flow. The implementation leverages programmable data planes' ability to maintain the switch state on a per-packet basis, perform limited computations, and dynamically select the egress port. Results show that AFQ allows incoming flows to achieve a fair share immediately and isolates them from other concurrent flows, leading to significantly more predictable performance. The authors report that the average FCT is improved for short TCP flows. Also, AFQ enhances the performance of DCTCP by 2x and TCP performance by 10x for both average and tail FCT in scenarios with high loads.

6.2.2. RFC-standardized AQM Algorithms Comparison, Discussions, and Limitations

Table 7 compares the surveyed schemes. RFC standardized AQMs can be described in P4, although some are not very precise due to the approximations techniques they use. For instance, in [53, 54] the CoDel algorithm uses simple arithmetic and packet counting to approximate the square root operation. However, results show that these limitations do not significantly impact the performance. Kunze *et al.* [55] demonstrated that implementing an RFC-standardized AQM in a programmable switch is constrained due to the resource limitation (i.e., the number of bytes allocated to queues) and the complexity of arithmetic operations such as multiplication and square root. They also described general implications for AQM design in hardware ASICs. First, gathering queueing information at the ingress facilitates the implementation of AQMs such as RED and PIE in the ingress pipeline. Second, reducing the complexity of data plane calculation makes the AQM control-law simple, so there is no need to rely on the complex operation. Flexible memory access allows early read access, and then one later write access enables the calculation of more complex control rules. Third, when designing an AQM, an AQM that is tunning-free or has concise and price ways to adopt parameters to new settings is preferred. Fourth, AQM algorithms should avoid underutilizing ASIC resources so that you fully leverage the capabilities of the switch. Finally, the authors conclude by remarking on a need for a general framework to directly manage the queue with P4.

Sharma *et al.*, [58] proposes a solution based on building blocks that mask arithmetic operations in the data plane. The goal is to provide the functional blocks (e.g., extern components) to perform complex arithmetic operations (e.g., division, multiplication, square root.) without using approximations or lookup tables (e.g., approximating the square root function can be achieved by counting the number of leading zeros through longest prefix match). These features can enable the development of more complex AQMs in the data plane.

With P4, programmers have access to packet metadata such as the queue depth, queueing time variations, ingress and egress timestamps, and packet length to implement AQM schemes. Although some packet metadata are only available at the egress control block (see Figure 2), the programmer can use cloning and recirculation primitives to reprocess packets that already include queueing information. Another challenge when implementing AQMs is that The use of complex arithmetic operations is constrained in P4 hardware targets.

6.2.3. Comparison with Legacy Approaches

Table 8 presents the limitations of the surveyed RFC-standardized AQMs. Several RFC-standardized algorithms were proposed since the first implementation of RED [8]. Many of the RFC-standardized AQMs are implemented in Linux as queueing disciplines. However, they can only be used with software switches. RFC-standardized can be implemented in P4 programmable data planes to complement and improve existing deployments.

On the other hand, legacy devices implement out-

Table 8: RFC-standardized Algorithms, Limitations and P4-Implementations.

Name	Limitations	P4-Implementation
Tail-Drop	- It does not provide traffic separation - Not suitable for providing QoS - High queue latency	Default
RED WRED ARED	- It requires adjusting tuning parameters - The queue length determines the dropping probability	[55, 63, 123]
CoDel FQ-CoDel	- It computes $\sqrt{n}$	[53, 54, 63]
PIE	- It requires adjusting tuning parameters (e.g., α, β) - It uses approximations	[55, 57, 123]

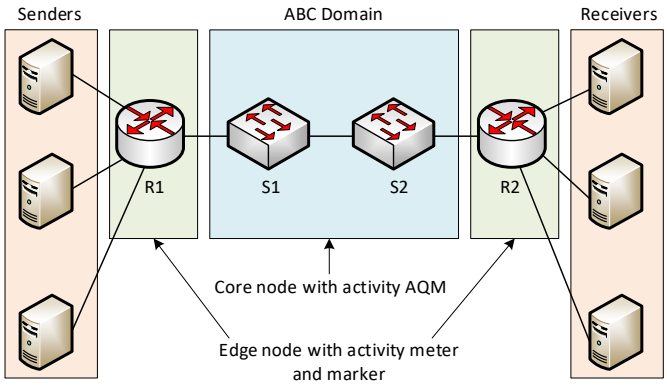

Figure 19: Activity metering and marking are performed only by ingress nodes. Both ingress and core nodes apply activity AQM during packet forwarding.

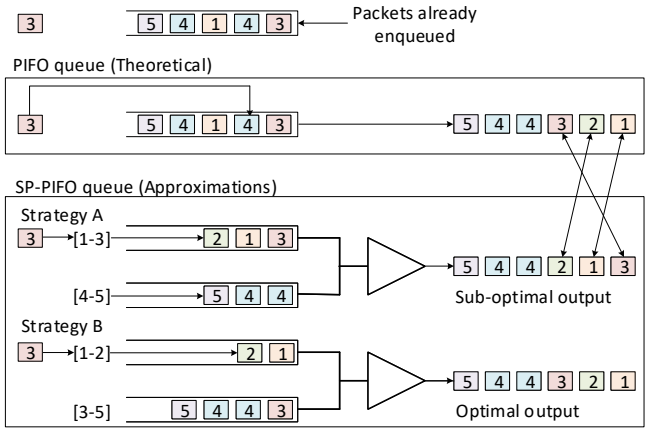

Figure 20: SP-PIFO approximates the behavior of PIFO queues by adapting how packet ranks are mapped to priority queues. Strategy A presents a sub-optimal solution. By re-assigning the ranks, the system produces an optimal solution (e.g., Strategy B) [61].

dated schemes based on RED (e.g., ARED, WRED), which require constant tuning of their parameters (e.g., thresholds). More recent AQMs such as FQ-CoDel and CAKE have already solved this issue of being parameterless by default. However, legacy devices do not adopt them due to the non-programmable nature of legacy routers, and the lack of adoption of AQM schemes in production deployments [108]. P4 can facilitate the adoption of AQMs to fulfill the requirements of a network by allowing the operator to update the data plane program.

6.3. Custom AQM Algorithms

Researchers have focused on improving TCP and the overall network performance by proposing novel AQM schemes to mitigate congestion and bufferbloat. With the advent of P4, the description, validation, and evaluation of custom AQM algorithms became faster. Although P4 was not designed to manage buffers and packet scheduling, developers can combine queueing disciplines and dropping policies to create AQMs that are more flexible compared to RFC-standardized AQMs. However, the match-action pipeline constraints, such as hard limits on multiplications and memory accesses or access to the queue state, can limit some schemes' implementation.

6.3.1. Literature Review

Mushtaq *et al.*, [59], argue that the traditional approach where end host-based congestion control algorithms respond to simple congestion signals from the network has a negative impact if the goal is to reduce the FCT. Therefore, they propose Approximate and Deployable SRPT (ADS), which uses approximations techniques to implement the Shortest Remaining Processing Time (SRPT). SRPT [124] is a queueing discipline that prioritizes flows with fewer outstanding bytes in the queue (i.e., with less remaining time). The authors claim that to minimize the average FCT in data centers, ADS can manage congestion. Their evaluations demonstrated that ADS achieves a performance close to SRPT, which has an advantage over the simple FIFO scheduling policy. Their approach leverage a small number of priority queues that require only software changes in the host and the switches.

Menth *et al.* [60] proposed a system called Activity-Based Congestion (ABC), based on a domain-based QoS mechanism. This scheme aims at providing more fairness among customers on bottleneck links using scalable bandwidth-sharing mechanisms for congestion management. Figure 19 depicts the system overview. Ingress nodes work as activity meters to measure the traffic rate that enters the ABC domain. They set an activity value for each packet and mark that value in its header. Such an aggregate may be, e.g., the traffic of a single user or a user group. Thus, ingress nodes require traffic descriptors for any aggregate that should be tracked. Ingress nodes and core nodes of an ABC domain are forwarding nodes. They use an activity AQM on each of their egress interfaces within the ABC domain to perform an acceptance decision for every packet. That means they decide whether to forward or drop a packet depending on its activity, enforcing fair resource sharing among traffic aggregates within an ABC domain. Egress nodes remove the activity information from packets leaving the ABC domain.

Alcoz *et al.* [61] tried to approximate the behavior of PIFO (Push-In First-Out) using programmable data planes. PIFO is a queueing discipline that allows enqueued packets to be pushed in arbitrary positions according to the packet's rank. However, PIFO implementation in hardware is challenging because sorting packets at line rate arbitrarily is not a trivial task. The authors leveraged the capabilities provided by programmable data planes to implement PIFO using approximations. The system is called SP-PIFO (Strict Priority PIFO). It consists of mapping between packet ranks and Strict Priority (SP) queues to minimize scheduling mistakes relative to the ideal PIFO implementation. Figure 20 depicts how PIFO works and its approximation scheme (SP-PIFO) where the number inside the boxes specifies the priority of the packets. PIFO schedules incoming packets serving them in a sorted way. Results show that SP-PIFO minimizes the FCT while enforcing fairness among competing flows. Although the scheme approximates PIFO behavior and matching it in many cases, the system cannot guarantee to perfectly emulate the behavior of a theoretical PIFO queue for all ranks.

Cascone *et al.* [62] proposed a P4-based system to mitigate the problem of a network switch enforcing fair bandwidth sharing of the same link among TCP and non-TCP senders. The scheme is called FDPA (Fair Dynamic Priority Assignment), based on the Fair Queuing (FQ)

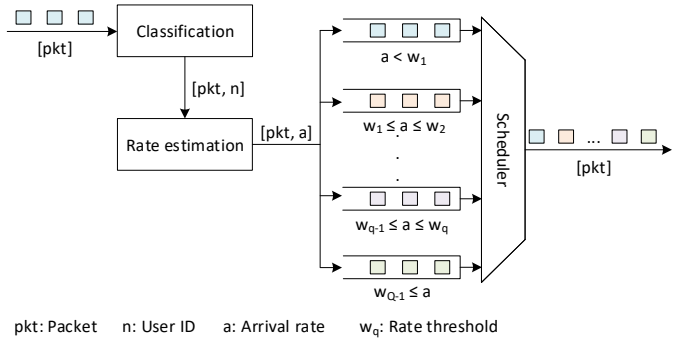

Figure 21: FDPA forwarding pipeline scheme. Packets are classified on a per-user basis then, a rate estimator which measures the arrival bit-rate processes each packet according to a specific user. Packets are then stored in one of the Q priority queues. Finally, a strict priority (SP) scheduler serves queues in priority order (i.e., the lowest bandwidth first) [62].

scheduling algorithm. The mechanism of FDPA consists of giving priority to the packets belonging to a user whose arrival bit-rate is equal to or less than its fair share over those users generating traffic at higher rates. However, FDPA does not provide precise bit-level or packet-level fairness. Instead, it approximates a fair share over longer timescales (e.g., in the order of few RTTs). Figure 21 shows the system's pipeline. Packets are first classified per user and then processed by a rate estimator, which measures the specific user's arrival bit rate. Then, packets are allocated in a priority queue Q_n, such that the higher is the arrival rate, the lower the priority will be. A strict priority (SP) scheduler serves queues in priority order: packets of priority are dequeued only if all other queues with higher priority are empty, where $q = 1$ is the highest priority. Experiments conducted at 10Gbps show that the performance of FDPA approximates the ideal Deficit Round-Robin (DRR) with dedicated per-user queues. However, FDPA doe not require implementing per-user queues.

Harkous *et al.* [63] propose a scheme that uses the target's match-action tables to implement virtual queues in P4-programmable data planes. The scheme aims to provide customizable traffic management by establishing how packets are served to the non-programmable traffic manager. The system works by allocating different network slices into virtual queues (vQueues) depending on the rules defined in the control plane (e.g., limiting the rate, controlling the delay, prioritizing flows). A slice is a unique logical, virtualized network built on top of a shared network. The scheme employs match-action tables to perform traffic management such as rate limiting and queue management. The system assumes that slice priorities are calculated and pushed top-down from the control plane to the data plane. Results show that policing approaches implemented in vQueues are effective in controlling TCP throughput per slice.

In recent work, Turkovic *et al.* [64] presented a scheme that implements a QoS-based forwarding called P4QoS. It allows applications to establish the QoS requirements by appending its condition to the packet header. The system combines information added by the application and the Link Latency Estimation Protocol (LLEP), which mitigates the lack of coordination among the clocks of the switches that add timestamps to the packet. Figure 22 shows how LLEP works. In step 1, the ingress switch S1 appends an LLEP header, containing an egress timestamp representing the time the packet left the switch, t_{send}. In step 2, switch S2 saves its ingress timestamp, representing the time the packet was received. Then, it clones the packet, and forwards the original application data packet further along the path, and calculates the processing plus queueing delay ($t_p + t_q$), which is the difference between the current egress and the saved ingress timestamp. Next, the information is added to the LLEP header. In step 3, the information produced in step 2 is sent back to the previous switch. Finally, in step 4, switch S1 uses the new information when the cloned packets are received to calculate the estimated link latency LL_{est}. With this new value, the switch updates the latency register at the ingress switch to estimate the latency register at the ingress switch is updated and used to estimate the packet deadlines for all the flows on the same output link. All the process is repeated periodically. Additionally, the authors developed a custom mechanism to determine the maximum allowed queuing delay satisfying the deadlines of all flows processed at the switch. Results show that P4QoS can significantly improve the performance in the network, particularly for low-latency traffic, by significantly reducing the number of marked packets without affecting the throughput.

Chen *et al.* [125] proposed QoSTCP, a P4-based scheme that enforces QoS policies on TCP flows. The system uses meters and Rate-Limiting Notification (RLN) to smoothly adjust the TCP congestion window. QoSTCP requires modifying the end-host congestion control to understand the RLN notification, which indicates the growth rate of the congestion window before reaching a threshold. In the switch, QoSTCP implements a Two Rate Three Color Marker meter (trTCM) and ECN using P4. Results show that QoSTCP prevents high variations in the throughput and reduces the average number of packet losses.

6.3.2. Custom AQM Algorithms Comparison, Discussions, and Limitations

Table 9 compares the schemes described in the previous section. FDPA and SP-PIFO use approximations in the data plane to implement a theoretical queueing model. Both schemes enforce fairness while minimizing the FCT. Although using approximations in the data plane can be inaccurate depending on the demands of the theoretical model, both schemes obtain good results. P4QoS and [63] leverage on multiple queues to implement custom policies. P4QoS requires tuning at least four parameters (i.e.,) to define the slices. In contrast, [63] does not require any tuning. Both schemes separate traffic, however, P4QoS enables applications to request specific QoS to the network (i.e., latency deadlines). On the other hand, the scheme in [63] overcomes a data plane architectural limitation by implementing virtual queues in the P4 pipeline. This solution is portable to other architectures.

6.3.3. Comparison with Legacy Approaches

P4 provides flexibility in the creation of custom AQMs. Therefore, more features can be added to programmable data planes compared to legacy devices. For instance, the behavior of schemes such as RED (includ-

Table 9: Comparison of Custom AQMs Schemes.

Ref	Name	Strategy	Thresholds	Multiple Queues	Dropping Signal	Integration with Legacy	Scheduling Algorithm	Tuning Parameters	Target
[59]	P4-SRPT	Approximate of SRPT using priority queues	2	×	minTCP SACKs ECN	×	FIFO FQ SJF	IW minRTT	ns2
[60]	P4-ABC	Implements ABC management in the data plane	1	×	Queue Length	×	ABC	None	BMv2
[61]	SP-PIFO	Approximates PIFO behavior using multiple queues	1	✓	N/A	✓	PIFO	None	Tofino
[62]	FDPA	Prioritizes traffic to enforce convergence	BW/a	✓	Packet losses	✓	Strict Priority	Number of Priorities	N/A
[63]	N/A	Establishes per slice traffic policing using vQeueus	>4	✓	TCM	✓	Multiple	Policy-based	Netronome Smart NIC and BMv2
[64]	P4QoS	Implements a custom QoS-based forwarding scheme	2	×	Timeout	✓	None	None	Tofino and BMv2
[125]	QoSTCP	Limits the rate of TCP flows using custom policier	2	×	Peak threshold	×	Max/min rate	None	BMv2

ing its variant ARED, and WRED) depends on tuning some parameters. Those parameters require adjustment depending on the network conditions. Therefore, the scheme becomes hard to manage and less autonomous. Operators continue using just the simple TD (i.e., no AQM). More recent AQMs such as FQ-CoDel and CAKE present fewer parameters, making them easier to manage. P4 approaches address many of the shortcomings observed in legacy AQMs. P4-based AQMs can dynamically adjust their parameters based on the data plane's measurements. With improved feedback, the switches and end hosts have a better view of the events in the network. Thus, they can perform more precise actions to network events.

6.4. Summary and Lessons Learned

The surveyed works agree that packet losses are an inaccurate indicator of congestion, suggesting that AQMs should complement congestion control schemes. Improving TCP congestion feedback via custom AQM schemes provides an efficient solution to the bufferbloat issue and other TCP performance metrics such as link utilization, fairness, and FCT. Most of the AQM summarized in previous sections consider packet loss or ECN as a congestion signal, making them TCP-oriented AQMs.

AQMs such as CAKE and FQ-CoDel, do not require an active adjustment of any parameters. Instead, they have an integrated dynamic feedback mechanism to ensure a low queueing delay. With P4, developers can implement RFC-standardized AQMs in a top-down approach and control the system's behavior under different network conditions on a per-packet basis. The use of approximations in the data plane demonstrated to be a feasible solution to match with theoretical models. Additionally, P4 facilitates measuring the queue metrics via the standard metadata and allows reprocessing packets invoking primitives such as resubmission and recirculation. These features enable programmers to find alternatives to using complex arithmetic operations.

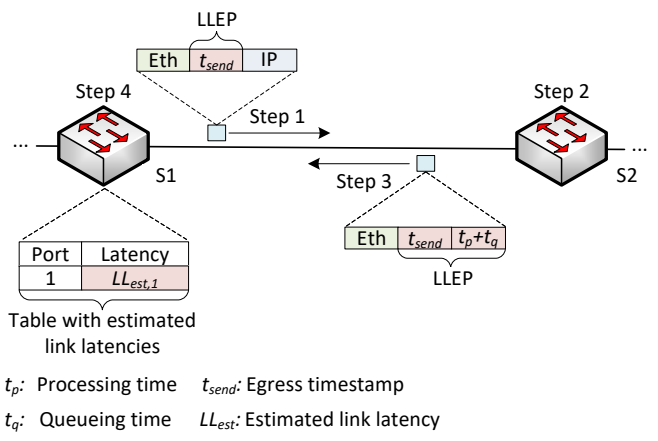

Figure 22: Overview of the Link latency estimation protocol (LLEP) [64]. LLEP mitigates the lack of coordination among the clocks of the switches.

7. TCP Offloading

The Network Interface Card (NIC) plays a vital role in network performance. Currently, network processing from low to medium speeds (<10Gbps) is possible through standard Operating System (OS) drivers and multicore processors available in the market. However, to process speed rates around 100Gbps, servers must bypass the kernel to save CPU cycles. Offloading processing and network functions to the NICs (e.g., SSL and TCP segmentation offloading), [126], can release the CPU from some network stack processes. Thus, improving the overall performance on the server. However, offloading operations come with some constraints, such as application data availability usually served by the general-purpose CPU. Table 10 compares programmable and non-programmable NICs. P4-programmable NICs offer great flexibility to perform custom packet processing.

Table 10: Comparison between P4-programmable and Non-programmable NICs.

Feature	P4-programmable	Non-programmable
Flexibility	High	Low
TCP Offloading	Full Protocol	e.g., TCP Fragmentation; TSO/GSO
Latency Scale	<1ms	[1ms-100ms]
General-purpose CPU Intervention	Low	High
Supports DPDK	✓	✓
QoS and ACL acceleration	✓	×
CPU bridging	Flexible	Limited

Such a feature enables the development of novel schemes that operate and scale faster than non-programmable approaches. Moreover, P4-programmable NICs can offload the server's operations by bypassing the OS kernel and communicating directly with the CPU.

7.1. TCP Offloading Limitations

7.1.1. Reducing TCP Overheads in Servers

The increasing gap between CPU capacity and network bandwidth makes it challenging to ensure TCP's desirable properties and save CPU cycles in the server. In [127], Mogul identified the complexities that legacy NIC faces to offload TCP in practice. A persistent problem described in his work is currently observed in short-lived TCP connections, which are prevalent in data centers and WANs [128–131]. Short-lived TCP connections produce significant overhead because the TCP stack must maintain the correct protocol behavior independently of the application implemented over it. Jacobson [132] proposed using header prediction to process common TCP scenarios using few instructions. Offloading TCP operations using P4-programmable NICs obtained more relevance in the last few years as they can be used for custom packet processing and offload server operations. Based on the flexibility provided by P4, many authors implemented schemes that offload transport protocols such as TCP.

7.1.2. Implementing Transparent Application Offloading Schemes

Offloading an application to the NIC without significantly modifying the original code is a challenging task that requires the development of compilation tools. Fully offloaded application schemes are constrained due to the intrinsic requirements of each application. Therefore, developers can decide which features are more suitable to be handled by the NIC. Sapio *et al.* [133] argue that a transparent in-network computation must be implemented with caution. Programmers have to identify what type of computation can be performed in the network to reduce communication overhead.

Several works propose using P4-programmable NICs to support and run microservices. The microservice architecture consists of an application comprising smaller components coupled independently over well-defined APIs. Although modern non-programmable NICs partially support TCP offloading operations (e.g., checksum calculation, TCP Segmentation Offload (TSO), and Large Receive Offload (LRO)), they mainly benefit large data transfers. P4-programmable NICs allow programmers to define whether an operation can be fully or partially offloaded to reduce the timescale of a process. P4-programmable NICs can perform computation closer to the network edge, offering a better bandwidth/cost efficiency than legacy approaches.

7.2. Protocol Offloading

According to [136], TCP protocol processing becomes the dominant overhead comparable to application processing and other systems overhead. Therefore, to reduce this trend, the network nodes' hardware must handle this issue to fulfill the speed requirements. The origin of this issue resides in the TCP stack design, which is implemented in the end devices and traditionally handled by the OS kernel. Consequently, allocating a considerable amount of computing resources results in a reduced network packet processing efficiency. Therefore, a software approach for handling packet processing cannot satisfy the increasing network transmission speed.

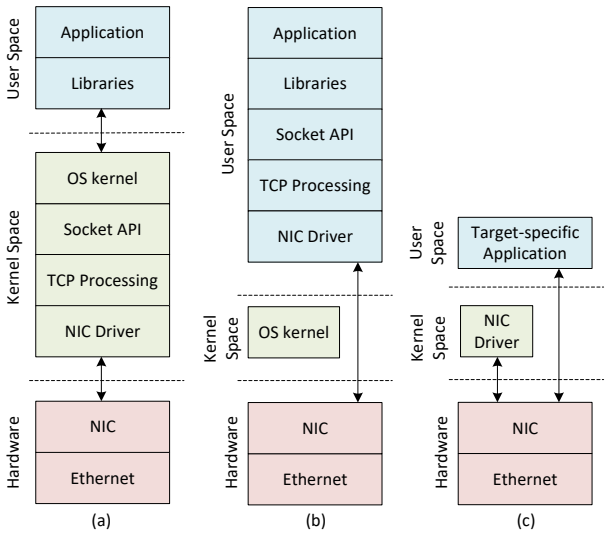

Figure 23: Conventional and kernel bypassing OS architectures. (a) Conventional operating systems implement all network processing in the kernel space. (b) Kernel bypass architectures avoid overheads for kernel crossings by moving the protocol implementation into the application and providing applications with direct access to the NIC for sending and receiving packets [134]. (c) Hardware accelerated data-path architecture requires a target-specific application [135].

Several efforts have been made to optimize TCP packet handling by improving what is known as TCP Offload Engines (TOEs) adapters. A TOE refers to a dedicated network function that handles a significant part of the TCP protocol in hardware to offload a general-purpose CPU from TCP processing. TOE devices can parse and remove TCP headers from incoming packets before transferring the data to the host, which reduces the burden of processing data in the kernel memory. A TOE device can buffer incoming packets and transfer them directly to the application's buffer bypassing the kernel. Similarly, the application can directly transmit data via the TOE's buffer instead of being held in the kernel's buffer.

Figure 23 illustrates the differences among a commodity, a kernel bypassing OS architecture, and a hardware accelerated data-path architecture. Traditionally, the OS handles all network operations (e.g., implementing TCP congestion control, encapsulating packets, and checking protocol correctness) to facilitate data exchange between applications. On the other hand, bypassing the kernel significantly reduces overheads and permits applications to define how network processing is performed. Therefore, the processing time is shorter and more deterministic, which enables applications to get better performance. However, developers should implement all the network protocols in the user space, which is not scalable. P4-programmable NICs can facilitate the implementation of TCP functionalities such as a congestion control algorithms and permit transparent application development in the userspace. The hardware-accelerated data-path archi-

tecture enables target-specific applications that run with high performance. One of the most popular accelerated data-path architecture is the Data Plane Development Kit (DPDK) [137]. DPDK is based on a run to completion model for packet processing. A run to completion model refers to a scheduling model that runs a task until it finishes. In DPDK, execution units such as CPU cores and memory are reserved before calling data plane applications. However, the latter implies that DPDK applications require a dedicated CPU core to poll packets, limiting the number of programs running on the architecture. architecture.

The DPDK implements a run to completion model for packet processing, where all resources must be allocated prior to calling Data Plane applications, running as execution units on logical processing cores.

NICs rapid evolution enabled the chances to support new features to offload the general-purpose CPU from network processing. NICs' role in modern server systems became more relevant than merely transferring data between the server's CPUs and the network link. Modern NICs comprise advanced features, namely protocol offloading, packet classification, rate limiting, virtualization, and cybersecurity. This type of device is referred to as smart NICs. Han *et al.* [138], identified three factors that contributed to this trend: 1) New applications whose performance requirements cannot be achieved with legacy networking stacks, requiring hardware augmentation, 2) The increasing adoption of virtualization, which requires the NICs to support switching functionalities, and 3) The growth of multitenant data centers and cloud computing where network isolation is necessary. However, the rapid evolution of smartNICs presented some challenges, such as the hardware complexity and the type of applications supported by the platforms.

The advent of P4-programmable NICs made it possible to achieve better network and processor utilization. P4-enabled NICs facilitate implementing schemes involving traffic encapsulation, load balancing, and security protocols (e.g., TLS handshaking, IPSec, and other encryption schemes) due to their flexibility and reduced complexity. Modern NICs such as Pensando [139], Netronome [140], Xilinx [141] and Innovium [142] have fully P4-programmable, energy-efficient, multi-core processors on which many packet processing functions, including a full-blown programmable switch, can run.

7.2.1. Literature review

Moon *et al.* [65] identified two scenarios where TCP servers suffer from poor performance: when handling short-lived flows and handling Layer 7 proxy (L7 proxy). Short-lived flows suffer from severe overhead in processing small control packets, and L7 proxy demands extensive CPU usage and memory allocation for handling packets between two connections. To solve both issues, the authors presented a hardware-assisted TCP stack architecture implemented using programmable NICs. The system is called AccelTCP, and it offloads TCP complex operations by reducing the CPU cycles used by the application's processes. Results show that short-lived connections perform similarly to persistent connections. It also significantly improves the performance of Redis [143], and HAProxy [144] which are L7 load balancers.

Yan *et al.* [66] used P4-enabled NICs to provide a solution designed and implemented to meet the 5G networking requirements. The authors described the design, implementation, and experimental results of P4-enabled smart NICs, and it is used to perform network slicing. Network slicing is a 5G terminology used to describe the sub-components of a network. These sub-components are referred to as the IP and optical network and their way of transport. The authors implemented a SmartNIC, a P4 program that inserts Segment Routing Multi-Protocol Labe Switching (SR-MPLS) header in ingress and deletes in egress. The system presented results showing that the SmartNIC can achieve a maximum of 84.8Gbps utilizing only one CPU core. With P4 SR-MPLS SmartNIC header insertion, the bandwidth performance can be up to 30% higher than legacy approaches.

Harkous *et al.* [67] present P8, a method to estimate the packet forwarding latency using P4 programmable NICs and DPDK-based software switch. The authors analyze the impact of different P4 constructs on packet processing latency for three state-of-the-art P4 devices: Netronome SmartNIC, SUME NetFPGA, T4P4S DPDK-based software switch. Results reveal that the forwarding latency varies depending on the P4 constructs that are adopted.

Qiu *et al.* [68] developed a tool called Clara, which aims to help developers evaluate the performance of a Network Function (NF) before offloading it to a SmartNIC. Clara takes as an input the original unsupported NF and predicts its performance after offloading it. Also, it can determine whether to offload an NF or not and how to perform an effective port. The system facilitates the developer to evaluate offloading strategies, obtain performance insights, and identify suitable SmartNIC models for specific workloads. The downside is that the validation was limited to a particular SmartNIC (i.e., Netronome Agilio CX).

7.2.2. Protocol Offloading, Comparison, Discussion and Limitations

Table 11 compares the schemes described previously. Performing TCP stateful operations in the NIC requires maintaining consistency of transmission control blocks in the NIC and the server. This issue occurs due to target constraints. The operations on the NIC's stack deviates from the state of usually another operation, which means that the P4 program must consider the target hardware architecture. Moreover, stateful TCP processing increases the complexity and resource consumption of the implementation running in the NIC. For example, TCP SYN packets require flow state initialization and setup. Therefore, depending on the packet size, such an operation might take a non-deterministic amount of time. This variation in the processing time occurs due to the packet length and/or checksum computation [68]. In [145], Grey *et al.* highlighted the gap between the expected and actual behavior in P4-programmable devices. The authors discussed the consequences of only relying on the network programming language when ignoring the limitations imposed by the target's hardware.

Table 11: Protocol Offloading Schemes Comparison.

Ref	Name	Strategy	Kernel Bypassing	Mode of Operation	Performance Speedup	Custom Header	General-Purpose CPU Intervention	Latency Scale	Target
[65]	AccelTCP	Offloading TCP startup and teardown to the NIC	✓	Endpoint	∼12x	✓	✓	<1ms	Netronome Agilo LX
[66]	N/A	Reducing end hosts and ToR switch interaction using the NIC	×	Middlebox	∼1.3x	×	×	<400us	Xilinx Ultrascale+ NetFPGA
[67]	P8	Using latency as a metric of performance in different targets	×	Middlebox	Variable	×	×	<50us	Netronome SmartNIC NetFPGA SUME
[68]	Clara	Provide performance evaluation for SmartNIC offloading operations	×	Middlebox	N/A	✓	✓	<1ms	Netronome SmartNIC
[69]	FlowBlaze	An open abstraction for building stateful packet processing functions in hardware	✓	Endpoint	N/A	×	✓	<1us	NetFPGA SUME

7.2.3. Comparison with Legacy Approaches

Legacy NICs performs TCP offloading operations such as the TCP Segmentation Offload (TSO) [146], and Generic Receive Offload (GRO) [147]. However, these methods do not support complex policies such as TCP flow state tracking and Single Root Input/Output Virtualization (SR-IOV). SR-IOV leverages the NIC to bypass the hypervisor and send packets directly to the virtual machine [148]. P4-programmable NICs can support custom implementations, and thus deliver more sophisticated features to perform advanced operations on packets. Therefore, P4-programmable NICs are appropriate to optimize network performance in data centers. Consider the example of AccelTCP [65] which aims at offloading TCP startup and teardown. The system is fully implemented on a P4-programmable NIC rather than as a server-based appliance. Traditional NICs (including hardware and virtual NICs) do not have such flexibility, restricting the schemes that can be implemented.

Moreover, in some cases, the NIC becomes a bottleneck [149] due to the physical input/output operations that involve processes copying packets from NIC to upper layers. Another existing drawback present in traditional NICs that perform offloading operations is the lack of network visibility. This issue happens because the operations are performed out of the OS memory space. P4-programmable NICs solve this issue by providing a clear and well developed interface with the control plane that allows programmers to query data from the data plane without degrading the performance.

7.3. Application Offloading

In the last two decades, network speed became significantly faster than CPU performance, forcing many data center applications to sacrifice application cycles for packet processing. The literature shows that there is a gap between a general-purpose CPU and a network processor ASIC. Exploiting SmartNICs capabilities to improve TCP performance is a challenging task due to the network protocol stack design. This means keeping the protocol's properties regardless of what the application does, which is usually a significant challenge when bypassing the kernel to send the application's data to the NIC directly.

As a result, developers have started to offload computation to programmable NICs, dramatically improving the performance and energy efficiency of many data center applications, such as search engines, key-value stores, real-time data analytics, and intrusion detection. However, implementing data center network applications in a combined CPU-NIC environment is challenging. It often requires many design-implement-test iterations before the accelerated application can outperform its CPU-only version. These iterations involve non-trivial changes: programmers may have to move portions of application code across the CPU-NIC boundary and manually refactor the program.

7.3.1. Literature Review

Choi *et al.* [70] present λ-NIC, a system that aims at offloading serverless workloads (i.e., λs) to SmartNICs. Serverless workloads are referred to as fine-grained functions with short service times and memory usage. If correctly balanced and distributed, serverless applications can be more efficient and cost-saving than buying or renting a fixed quantity of servers, which generally involves significant periods of underutilization or idle time. Developers can take advantage of the flexibility provided by serverless computation, focusing solely on creating custom programs (i.e., λs) without worrying about the infrastructure they use. However, many of these workloads, which are small functions, run on virtual machines. Virtualization technology becomes inefficient (i.e., processing delays and memory overheads)in these cases where the service's cost depends on the usage. The architecture of a typical server CPU cannot handle thousands of simultaneous threads. Therefore, when a function interrupts the CPU, it has to store the current process's state

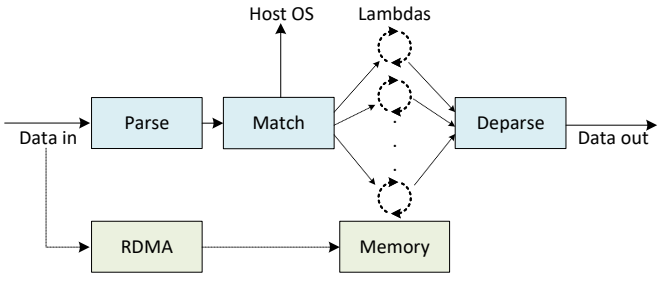

Figure 24: λ-NIC's abstract machine model. λs are independent programs that only a matching rule can invoke. The matching stage forwards the packets to the corresponding λ or the OS. The deparser associates the result of each λ to its corresponding header.

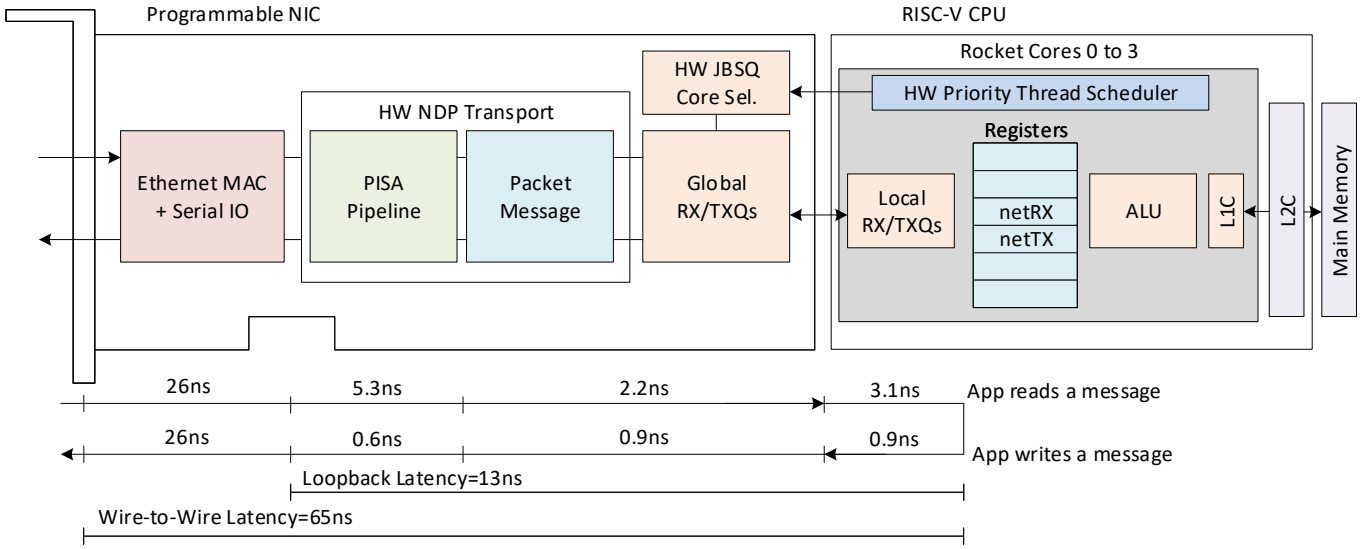

Figure 25: The nanoPU design. The NIC includes ingress and egress PISA pipelines, a hardware-terminated transport, and a core selector with global RX queues; each CPU core is augmented with a hardware thread scheduler and local RX/TX queues connected directly to the register file. Total wire-to-wire latency is 65ns [74].

and retrieve it back, wasting tens of milliseconds of CPU cycles. The authors used SmartNICs, or more specifically ASIC-based NICs, to run these *lambda*s more efficiently. ASIC-based NICs consist of hundreds of Reduced Instruction Set Computer (RISC) processors (i.e., NPU) with independent local instruction storage and memory. Consequently, these SmartNICs-unlike GPUs and FPGAs, optimized to accelerate specific workloads, are more suitable for running many discrete functions in parallel at high speed and low latency. Consider Figure 24, λ-NIC implements a Match-Lambda programming abstraction similar to the Match-Action Table (MAT) abstraction but, in this case, the λs performs more complicated operations.

In this model, λs are completely independent programs that do not share states and are isolated from each other. The matching stage operates as a scheduler that forwards packets to the matching lambdas or the host OS. In the last stage, a parser handles packet operations (e.g., header identification), and lambdas operate directly on the parsed headers. Therefore, the abstract machine model provides developers with an optimal way to run serverless workloads in parallel without any other interference. The authors developed an open-source implementation of λ-NIC using P4-enabled SmartNICs. They also implemented the methodologies to optimize λs to utilize the SmartNIC resources efficiently. Results show that λ-NIC achieves a maximum of 880x in workloads' response latency and 736x more throughput reducing the main CPU and memory usage.

Ibanez *et al.* [74] presented a scheme called nanoPU, which minimizes the tail-latency for Remote Procedure Calls (RPC). The system mitigates the causes of high RPC tail latency by bypassing the cache and memory hierarchy. The nanoPU directly places arriving messages into the CPU register file. Figure 25 shows the block diagram of the nanoPU. Notice that all the elements are implemented in hardware. The system implements NDP [41] due to its low-latency performance. A message buffer serves packets at line rate and a core selection algorithm responsible for allocating incoming processes to idle cores. The system supports a priority thread scheduling algorithm that supports up to four threads and a register file interface. Finally, the scheme has a hardware/software interface to debug and test custom programs. Results show that the wire-to-wire latency through the application is just 65ns, about 13× faster than the current state-of-the-art approaches. The proposed hardware transport layer responds faster than software, leading to a tighter congestion control loop between end-points. Additionally, nanoPU can handle many simultaneous RPC communications that scale linearly with the number of outstanding messages rather than hosts in a data center.

Gao *et al.* [71] proposed a system called OVS-CAB that applies an efficient rule-caching to offload an Open Virtual Switch (OVS) to a SmartNIC. The system can handle traffic at a high line rate and offload rule lookup functions SmartNIC keeping a low CPU usage. Figure 26 depicts the scheme behavior. Moreover, by caching

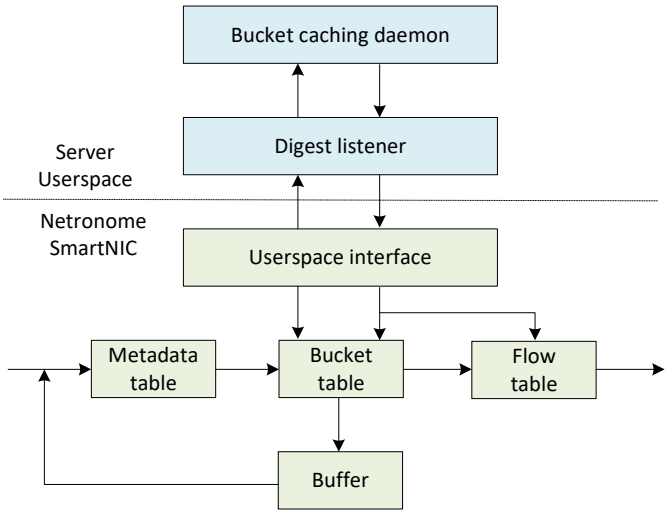

Figure 26: System overview. OVS-CAB hardware offloading system consists of three principal components: 1) A P4-based hardware table pipeline that serves as the hardware data plane, 2) A userspace control plane module that implements OVS-CAB rule-caching logic, and 3) An RPC-based message system that communicates between the data plane and the control plane.

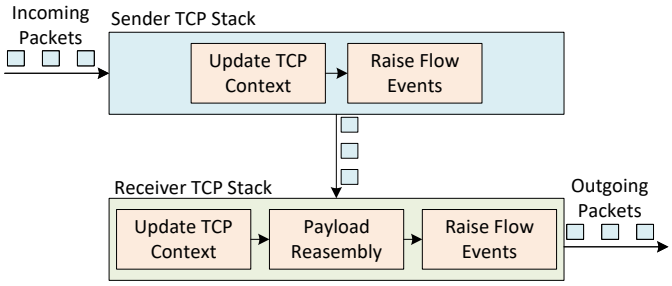

Figure 27: Packet processing steps in mOS. Upon packets' arrival, mOS first updates its TCP context for the packet sender. Then, it sends the records of all flow events that must be triggered [150].

selected rules, the system achieves a high data plane hit rate while using low memory. Results show that OVS-CAB improves the throughput of OVS and significantly reduces the host CPU utilization.

Kohler *et al.* [72] developed an in-network computing system for Complex Event Processing (CEP). CEP is a method that consists of tracking and analyzing streams of information about events to infer conclusions from them. This method requires real-time processing to extract the data from events such as text messages, social media posts, stock market feeds, traffic reports, weather reports, or other data types. The authors demonstrate that it is feasible to express CEP operations in P4 and develop a tool to compile CEP operations. Furthermore, they identified challenges and problems to suggest future research directions for implementing full-fledged in-network CEP systems.

Mohammadkhan *et al.* [73] offloaded NFVs by using P4-programmable NICs. NFVs are widely adopted in large-scale enterprise and data center networks to deliver more complex and diverse in-network processing. The scheme is called P4NFV and aims at determining the partitioning of the P4 tables and optimal placement of NFs to minimize the overall delay and reduce resource utilization. P4NFV offers a unified host and NIC data plane environment to the SDN controller.

P4NFV's framework considers both host and NIC capabilities to provide a unified view of a P4-capable NFV processing engine. For example, in [151], the authors describe how NFs can use high-performance userspace TCP stacks to simplify operations such as TCP byte stream reconstruction. This relatively heavyweight function involves copying packet data into a buffer, incurring unnecessary latency. This latency is due to the processing delay of NFs chain that perform TCP byte stream reconstruction using mOS [150]. mOS is a reusable networking stack for stateful flow processing in middlebox applications. Figure 27 shows the packet processing steps of mOS. Upon packets' arrival, mOS first updates its TCP context for the packet sender. Then, it sends the records of all flow events that must be triggered. Results with P4NFV demonstrate that as the chain length increases, the latency for the NFs performing TCP processing increases significantly compared to layer-2 NFs. This additional latency can be avoided by performing TCP processing only once and then exposing the resulting stream to the sequence of NFs.

Pontarelli *et al.* [69] presents FlowBlaze, an abstraction that extends match-action languages such as P4, to simplify the description of a large set of layer 2 to layer 4 stateful functions, making them available for implementations on FPGA-based SmartNICs. FlowBlaze adapts match-action tables to describe the evolution of a network flow's state using Extended Finite State Machines (EFSM). The framework aims to provide a system that allows a program with little hardware design expertise to implement quickly, update, stateless, and stateful packet processing functions at high speed on FPGA-based SmartNICs.

7.3.2. Application Offloading Comparison, Discussions, and Limitations

Table 12 compares the schemes described in previous sections. P4-based NICs are continually evolving to support new features. The role of NICs in modern server systems has changed. NICs can now host advanced features, such as protocol offloading, packet classification, rate limiting, cryptographic functions, and virtualization. Jang *et al.* [138] identified three factors that contributed to this trend: 1) The development of new applications whose performance requirements cannot be met by legacy networking stacks, demanding hardware augmentation, and 2) The increasing popularity of virtualization, where NICs should support switching functionality, and 3) the rise of multi-tenant data centers and cloud computing for which NICs must provide isolation mechanisms. However, a drawback of P4-based NICs is the complexity of the platforms. High-level programmable platforms support limited network functions. Existing NICs do not offer a systematic methodology for chaining multiple offload capabilities. According to Le *et al.* [152], they cannot fully implement complex network functions required by some encryption schemes or deep packet inspection. P4-programmable NICs present a higher level of flexibility and programmability. These programmable NICs can offload almost any packet processing function from the server. Moreover, they employ energy-efficient processors compared to x86-based server processors, achieving higher energy efficiency in packet processing.

7.3.3. Comparison with Legacy Approaches

P4-programmable NICs offer more flexibility in offloading TCP functionalities than traditional NICs, which have hardwired offloading modules (e.g., TSO, GSO, checksums). P4-programmable NICs present specialized packet engines that allow the implementation of a variety of protocol accelerators. Therefore, P4-programmable NICs can be considered as a general-purpose offloading platform to develop key/value store applications, microservices, and other types of network functions. P4-programmable NICs can be considered network accelerators capable of offloading communication routines and computational kernels from the CPU. On the other hand, programmable NICs that do not use P4 language (e.g., Mellanox BlueField SmartNICs [153]) present a lower degree of flexibility, making the development process more difficult and closed to a vendor technology. P4-programmable NICs offer similar performance and a scalable solution for large and diverse deployments. Moreover, with P4, the NIC's functionalities can be extended to support applications, thus minimizing CPU intervention.

Table 12: Application Offloading Schemes Comparison.

Ref	Name	Strategy	Kernel Bypassing	Mode of Operation	Number of Threads	Performance Speedup	Custom Header	GP-CPU Intervention	Latency Scale	Target
[70]	λ-NIC	Offloading serverless operations to the NIC	✓	Endpoint	>4000	∼880×	×	×	<1us	Netronome SmartNIC
[71]	OVS-CAB	Offloading an OVS to the NIC	✓	N/A	Endpoint	∼1.9×	×	✓	<1ms	Netronome SmartNIC
[72]	P4-CEP	Offloading rule lookup functions to the NIC	×	Middlebox	N/A	N/A	N/A	✓	<600us	Netronome SmartNIC
[73]	P4-NFV	Providing a unified P4 switch abstraction framework to simplify the SDN control plane	×	Middlebox	≥8	∼2.5×	✓	✓	<1ms	Netronome SmartNIC
[74]	nanoPU	Minimizing tail latency for RPCs by bypassing the cache and memory hierarchy	✓	Endpoint	≤4	N/A	×	×	<70us	Netronome SmartNIC + RISC-V

7.4. Summary and Lessons Learned

P4-programmable NICs complement server operations by not just simple forwarding packets. Instead, they can perform complex operations and speed up the delivery of services that otherwise are affected due to the usage of the general-purpose CPU. Offloading computation from a server's CPU to a programmable NIC releases a substantial amount of the server's CPU resources, making programmable NICs attractive for cloud operators to save costs. Firestone [154], considers that programmable data planes improve cloud computation due to their high performance and low overhead. These are two essential characteristics for data centers to reduce costs and increase performance. In the last decade, the network speed increased around 100Gbps and included new services and applications. This increase demands an efficient solution that reduces CPU cycles raises the customer's overall cost, and reduces the available processing power. To this extent, ensuring desirable TCP behavior usually entails performance issues, which increases the gap between CPU capacity and network bandwidth. McKeown [155] remarked this difference. Future work should consider the security implications of bypassing kernel functionalities that involve checksum computation and firewalls policies.

8. Network Measurements

Applications that use TCP as their transport protocol demand high throughput and low latency. Eventually, these applications can experience performance problems that are hard to diagnose due to the complexity of the TCP stack. Such complexity implies using heuristics to understand network conditions and application behavior. Therefore, there is no optimal setting to deal with and optimize TCP traffic which usually depends on application requirements.

Events that affect TCP performance are getting harder to diagnose because of the increasing link speed used in environments such as data centers. Moreover, the tools used to diagnose TCP performance are still the same as ten years (e.g., capturing packet traces and capturing TCP executions). It has been shown that these tools [156–158] are effective to diagnose individual TCP connections. However, they are not scalable to diagnose problems in large data centers, where the number of flows can easily surpass one million.

8.1. Challenges on TCP Performance Monitoring

A TCP connection might encounter performance issues at the sender, receiver, or network. Table 13 summarizes some examples of performance problems corresponding to each element of the network. Considering the significant amount of applications that use TCP, it is hard to select the right metrics to identify TCP performance issues.

8.1.1. Identifying Issues in the Sender

TCP performance problems at the sender may occur due to limited resources. For example, a slow disk, slow CPU produce data at a lower rate than the maximum capacity the network can handle. This limitation is also known as the application is non-backlogged. To identify this issue, measurement schemes can count sent packets and compare them to the estimated sending rate, which is defined by receive and congestion windows. Notice that the resource's limitation on the receiver side can affect the performance too.

8.1.2. Inferring Network Statistics

Network problems degrade TCP's performance. For example, TCP reduces its sending rate as a result of packet losses and high latency. The sender determines its sending rate based on its congestion control algorithm, which comes in different flavors. That means that the dynamics of a TCP flow vary depending on the congestion control algorithm. Identifying the TCP congestion control and the number of TCP flows using a specific congestion control provide relevant information to notify network elements (i.e., sender, receiver, router, and switches) about the performance. Experimental evaluations [159, 160] reported that BBRv1 present poor coexistence with algorithms such as CUBIC. However, legacy

Table 13: Elements of a TCP Connection and its Point of Failure [75].

Element	Performance Problem	Proposed Solution
Sender	- Data rate limited by resource constraints - Not enough data to send (non-backlogged) - Bottleneck produced by the server's processing speed	Estimating the right sending rate
Network	- Congestion (high loss and latency) - Queueing delay - Routing changes - Limited bandwidth	Providing enhanced feedback
Receiver	- Delayed ACK - Small receive buffer	Ensure the receiver is not the bottleneck

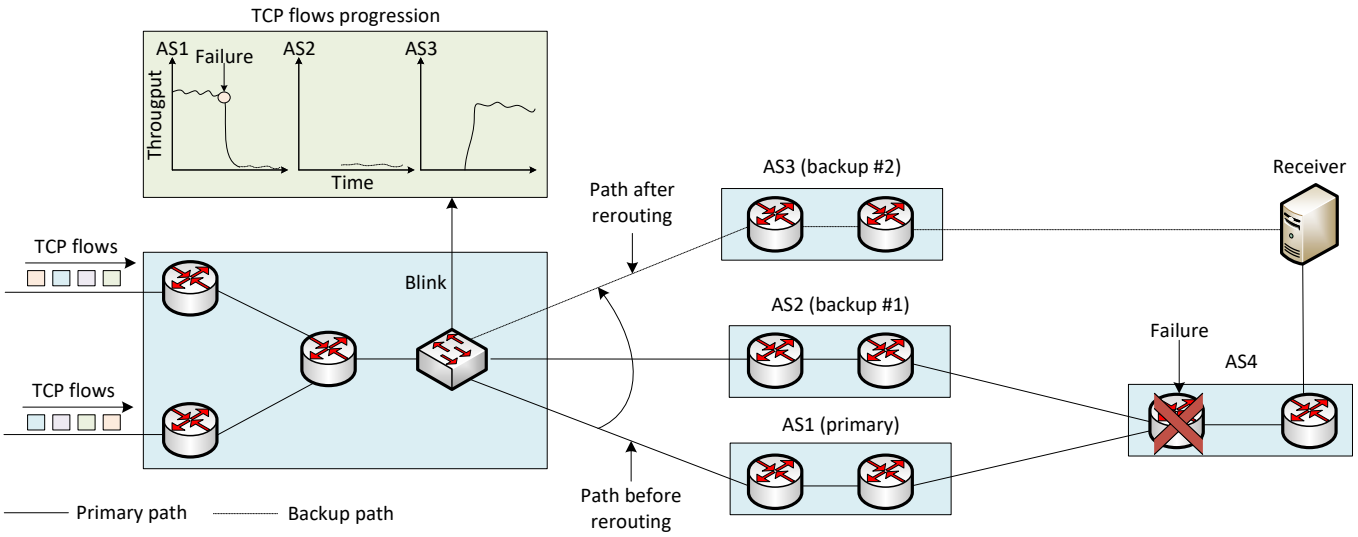

Figure 28: Blink mechanism. Before the failure, the primary path is through AS1. Blink identifies the failure in AS4 measuring the retransmissions. Blink immediately reroutes TCP flows to both AS2 and AS3. The Blink switch determines that AS2 is not working and forwards all the traffic through AS3 [77].

devices are unable to provide such information. With programmable data planes, developers can create schemes to infer the congestion control [47] as well as many other metrics such as RTT, retransmission rate, link utilization, etc. Programmable data plane can provide real-time monitoring by extracting packet header information (i.e., packet parsing), observing the information carried by packets across multiple stages (i.e., P4 metadata), and track the state of TCP flows using registers.

8.1.3. Tuning Receiver's Parameters

The receiving TCP window determines the amount of data the receiver can hold. The larger the buffer, the more data can be in flight between two hosts. If the TCP receiving buffer is smaller than the BDP, the sender waits longer for the receiver to acknowledge the data, resulting in degraded performance. Identifying the limitations of a TCP connection on the receiver side is essential to tuning the buffers correctly.

8.1.4. Network Diagnosis

Network operators need to know how well their network performs to determine what services they can offer their customers. However, visualizing the network behavior is not an easy task. Legacy measurement schemes rely on polling and sampling methods. Thus, they cannot provide a granular view of the events that affect the performance. As the volume of TCP traffic keeps increasing, reporting network performance and detecting failures requires efficient schemes capable of processing, characterizing, and modifying TCP dynamics.

According to [161], the global IP traffic was 1.2 Zettabytes (ZB) per year (i.e., 1.2 billion Gigabytes (GB)) per month in 2016. They estimated that by 2021, the global IP traffic would reach 3.3 ZB per month, which means that the global IP traffic in 2021 is $\times$ 127 larger than in 2005. Recently, during the COVID-19 pandemic, the Internet traffic increased between 25% and 35% in march 2020. This is due to the worldwide lockdowns and the shift to remote working policies [162].

The abrupt increase of Internet traffic demands an efficient way to measure network performance and detect failures. P4-programmable devices emerge as a promising solution to implement novel measurement schemes by addressing the inherent shortcomings of legacy networks. P4-based schemes can accurately report network issues and take countermeasures oriented to mitigate network problems. The source of such failures might have different and diverse sources (e.g., traffic bursts, resource allocation, protocol limitations, malicious or misconfigurations) being a challenge to implement an efficient network diagnosis scheme.

Sampling and polling methods do not have the resolution to detect events in the order of microseconds (e.g., typically, 1/30,000 packets), which provides coarse-grained visibility. The inherent high sampling and polling rate thwarts the development of measurement applications. For instance, it is impossible to measure the dynamics of TCP parameters such as congestion window, receive window, sending rate, and others. P4 Programmable data planes enable the possibility to perform fine-grained measurements in the data plane at a line rate (e.g., ~100Gbps). P4 Programmable switches provide flexibility to inspect traffic with high accuracy. They also allow programmers to take actions almost in real-time (e.g., dropping a packet after a threshold is surpassed, as done in AQMs).

8.1.5. Literature Review

Holterbach *et al.* [163] designed a system that recognizes a disruption in the TCP behavior, facilitating its tracking. The system is called Blink, and it is based on the predictable behavior of a TCP connection upon a packet loss. Figure 28 shows a scenario where a Blink switch redirects the traffic upon detecting a failure. Whenever Blink detects a failure, backup paths that are pre-populated by the control-plane, are immediately activated. Blink also permits policy-based rerouting. Thus, network operators can adapt the system according to their needs. The authors also considered security aspects. For instance, if an attacker generates significant traffic to redirect all the traffic through a malicious path, Blink can differentiate whether the traffic is legitimate and decide

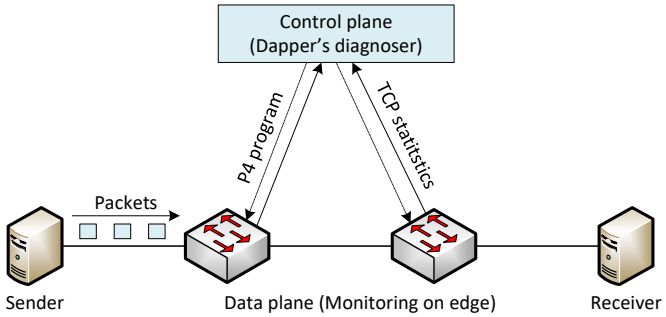

Figure 29: Dapper's architecture : 1) Data plane monitoring on edge, 2) Control plane diagnosis techniques.

accordingly. The authors evaluated Blink's performance considering a large number of flows. Results indicate that the system achieves a fast reaction time to redirect Internet traffic. It prevents unnecessary traffic shifts even in noisy scenarios.

Ghasemi *et al.* [75] presented a system to diagnose cloud performance using programmable data planes. The system is called Dapper, and it analyzes TCP performance in real-time at the hypervisor, NIC, or top-of-rack switch. The system identifies when a connection is limited by the sender, the network, or the receiver. For instance, when a slow server competes for shared resources or the network experiences congestion, the receiver has a small receive TCP buffer. The authors evaluate the accuracy of the system emulating three scenarios: 1) The sender presents a low performance (e.g., slow disk or busy CPU), 2) The receiver has limited resources such as small TCP receiving buffer size, 3) The network experience congestion. Figure 29 depicts the system's architecture. Dapper measures the metrics produced by the sender, network, and receiver to detect end-to-end performance issues. It helps to identify which entity is responsible for poor performance. Such procedure is usually the most challenging part of failure detection and can take from an hour to days in data centers [164]. Once the system identifies the bottleneck, specialized tools within that component can pinpoint the root cause. Dapper infers key TCP metrics such as counting the number of bytes, packets sent or received, congestion, and receive windows. Dapper achieves an average accuracy of 94% detecting problems regardless of their nature. The authors also mention that the accuracy is proportional to the severity of the problem. However, Dapper provides an average accuracy of around 94%. Liu *et al.* [165] propose a sketch-based performance monitoring scheme that allocates sublinear memory as a function of the number of flows. A sketch is a reduced data structure that comprises the main characteristics of a more extensive set of data. Typically, performance monitoring requires combining information across pairs of packets (e.g., matching a packet with its acknowledgment to compute the RTT). The authors tested the system in six platforms (e.g., P4, FPGA, GPU, multi-core CPU, and OVS), being the one implemented in P4 the fastest. Results show that the system detects ∼82% of top 100 problematic flows among real-world packet traces using just 40KB memory in the Tofino ASIC.

Wang *et al.* [79] proposed a P4-based TCP-friendly meter that does not degrade TCP performance. They

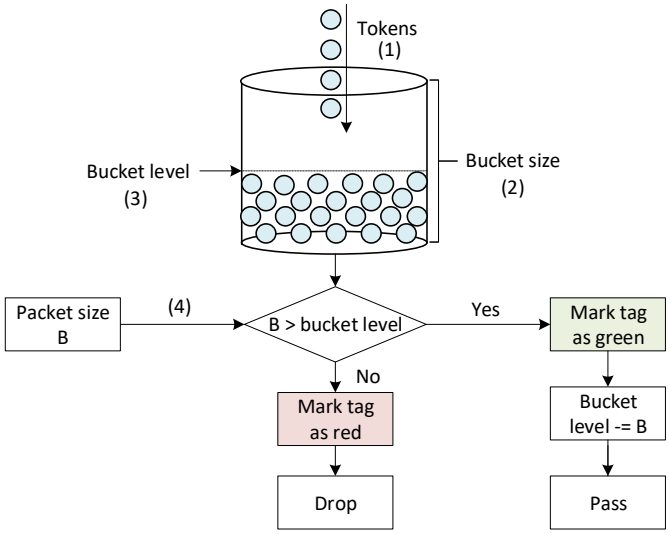

Figure 30: Scheme of a single rate two color meter algorithm. The meter uses a bucket to accumulate tokens that are generated at the speed of the meter. The meter marks incoming packets on red or green depending on the packet's size. If the size of a packet exceeds the current level of tokens in the bucket, it is marked red and then dropped. Otherwise, the packet is marked green and passes the meter.

found that meters in legacy switches interact with TCP flows so that these flows can only reach about 10% of the target rate. Based on their study, the authors found that the TCP performance degradation problem is caused due to the harmful interaction between the metering algorithm and the TCP congestion control. The system integrates a single rate two color meter algorithm, ECN, RED [8], and an adaptive control scheme [166]. Figure 30 describes the flow chart of a single rate two color meter algorithm used in the scheme. The meter uses a token bucket to accumulate tokens that are generated at the rate of the meter. The meter marks incoming packets on red or green depending on the packet's size. If the size of a packet exceeds the current level of tokens in the bucket, it is marked red and then dropped. Otherwise, the packet is marked green and passes the meter. Experimental evaluations show that the proposed system achieves rates of up to 85% of the target rate.

Chen *et al.* [78] leverage on programmable data plane to implement a passive tool to measure the RTT. The proposed algorithm calculates the RTT by matching the sequence number Seq of each packet with its corresponding acknowledgment ACK. Compared to traditional measurement systems, which are mainly based on active probing, the proposed system produces many samples for long TCP connections. Due to the memory limitations, the authors implemented the records in a multi-stage hash table that assigns an expiration time for each connection. Therefore, the entries matching with incoming packets produce RTT samples and consequently are deleted. On the other hand, those packets that never matched with their corresponding ACK are tagged as expired based on their timestamps and are overwritten when hash collisions occur. Results report over 99% of all RTT samples obtained in a campus network. The evaluation also reports that the data structure employed to store the sequence numbers achieves the best performance for RTT monitoring. Future works consider including anomaly de-

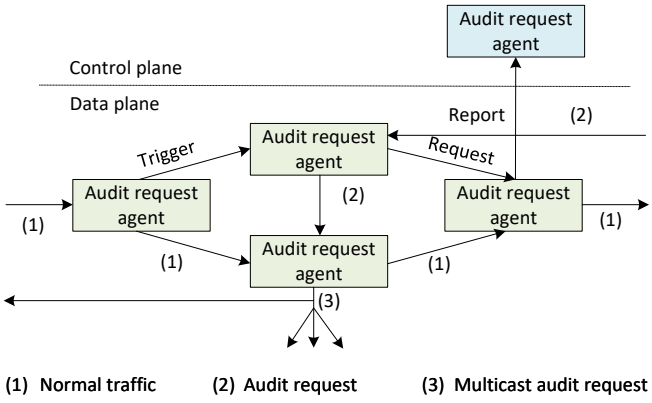

Figure 31: System architecture. SpiderMon keeps track of the most recent contention information. The telemetry data structure preserves the most recent packet-level information in a logically circular buffer.

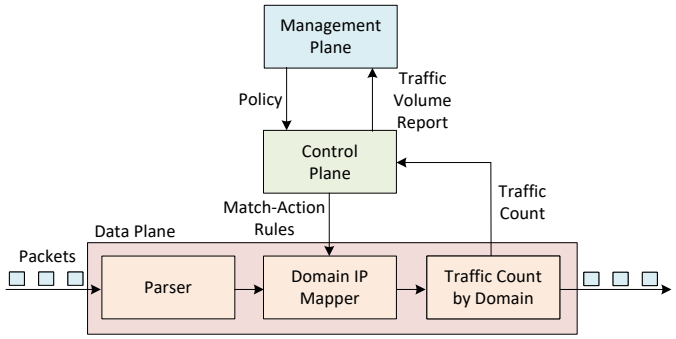

Figure 32: Meta4 architecture. The system comprises three levels: the network operator, the control plane, and the data plane. The network operator defines high-level policies that the control plane converts into match-action rules for the data plane [81].

tection.

Wang *et al.* [76] contributed with a similar work that addresses a limitation of the previous system, detecting the root cause of performance degradation in data center networks. The authors analyzed similar solutions and pointed out weaknesses such as the overhead produced by the data collection, the difficulties in locating the point of failure, and the tuning requirements of query-driven solutions. The proposed solution is called SpiderMon, and it is composed of two phases. The first phase keeps track of the performance of every flow until it detects a problem. When this happens, the system passes to the second phase and triggers a debugging process that uses a causality analyzer to find the root cause of the performance degradation. Figure 31 shows the system architecture. SpiderMon precisely collects diagnostic information without producing excessive overhead, i.e., it can accurately find the root cause with a limited number of telemetry data.

Chen *et al.* [80] presented ConQuest, a compact data structure implemented in the data plane that identifies the flows making a significant contribution to the queue. Excessive queueing leads to higher delay and eventually to packet drops. This problem escalates in the presence of microbursts, which is a phenomenon that increases the queue abruptly in a short period (i.e., in a sub-millisecond scale), exhausting the egress buffer. ConQuest can identify contributing flows with 90% precision on a 1 ms timescale, using less than 65 KB of memory. The system is implemented using a measurement data structure that operates at line rate to detect and reduce even short-lived queue buildup as it forms. The authors determined the optimal queue length to allow a single TCP connection to reach line rate based on the minimum congestion window size required based on the RTT.

Kim *et al.* [81] introduce Meta4, a framework used to implement network monitoring by the domain name in the data plane by collecting and organizing information from the DNS response into match action tables. The system aims at reporting statistics such as the traffic volume based on a user-defined policy. Figure 32, the system is divided into three levels, the management plane, the control plane, and the data plane. In the management plane, the network operator defines high-level policies that the control plane converts into match-action rules that will run in the data plane. Therefore, the network operator can measure policy-based metrics such as the traffic volume per domain. Additionally, the scheme ensures privacy by not exposing individual IP addresses. The scheme can be expanded to cover other applications that include implementing custom traffic monitoring policies, firewalls, limiting the rate, and rerouting.

In another work, Chen *et al.* [167] developed a system that uses programmable data planes to monitor the queue latency in a legacy switch. The programmable data plane taps both ingress and egress ports and computes the time difference between incoming and outgoing packets to infer the queue length. Results collected in a campus network demonstrated the heterogeneous nature of the events that lead to high queueing delay. Such events result from 1) Steady-state congestion, which consists of high queueing delays for a long time. 2) Bursty traffic makes the queueing delay oscillate due to concurrent large flows competing for bandwidth. Notice that bursty events diminish the capacity of TCP congestion control algorithms to probe the bottleneck bandwidth correctly. Therefore, also affecting other TCP flows and leading to poor link utilization. 3) Outlier high-delay packets, the authors detected link underutilization and packets with a low queuing delay caused by a full queueing buffer. The latter requires further research.

In a recent work, Kfoury *et al.* [168] designed a P4-based scheme that dynamically adjusts the buffer size of legacy routers. The programmable switch is employed as a passive tool that measures the RTT and the number of flows to set the buffer size of a legacy router and reduce unnecessary delays. Figure 33 shows a high-level overview of the proposed system. The dynamic buffer adjusting process consists of three steps: (1) A copy of the traffic is received by the P4 switch. (2) The RTT of individual TCP flows is computed at the P4 switch's data plane. (3) The P4 switch modifies the routers buffer size according to the equation $RTT/\sqrt{N}$ [117]. Results show an improvement in metrics such as the RTT, fairness, and the loss rate. Moreover, the FCT of short flows is also improved compared to a scenario where the buffer size remains fixed.

8.1.6. Network Performance Measurements Schemes Comparison, Discussions, and Limitations

Table 8.1.5 compares the schemes described in the previous section. Considering that TCP is the most

Table 14: Network Measurement Schemes Comparison.

	Ref.	Name	Strategy	Measured Information	Introduces Overhead	Reactive Processing	Network Wide	Integration with Legacy	Analysis	Target
Diagnosis	[78]	N/A	Passively monitors the RTT in the data plane	RTT	×	N/A	✓	✓	Data Plane	Tofino
	[75]	Dapper	Analyzes TCP performance in real time near the end hosts	RTT	×	N/A	×	×	Control Plane	N/A
	[81]	Meta4	Implements network monitoring by domain name	DNS response	×	Collecting packet size	✓	✓	Control Plane	Tofino
	[79]	N/A	Implementing a TCP-friendly meter in the data plane	Traffic rate	✓	N/A	×	×	Data Plane	Ivantec
	Ref.	**Name**	**Strategy**	**Measured Information**	**Passive Measurement**	**Mitigation Strategy**	**Network Wide**	**Failure Detection Method**	**Analysis**	**Target**
Performance Degradation Handling	[163]	Blink	Using TCP induced signals to detect failures directly in the data plane	Flow size and duration, packet size, RTT	×	Rerouting	×	Heuristics	Control Plane	Tofino
	[76]	SpiderMon	Identifies the root cause of performance degradation problems with minimal overhead	Queue latency	×	Reporting Flow ID	✓	Provenance Graph Model	Control Plane	SUME Switch
	[80]	ConQuest	Identifies the flows making a significant contribution to the queue	Queue length	✓	Adjusting the Rate	✓	Data Structure	Data Plane	Tofino
	[168]	N/A	Uses a P4 switch as a measurement tool to dynamically modify the buffer size	RTT, Number of flows	✓	Adjusting the buffer size	✓	N/A	Control & Data Planes	Tofino

widely adopted transport protocol, many performance diagnosis schemes track TCP behavior to infer performance issues. Table 8.1.5 compares the approaches presented in the previous section. For instance, to identify the point of failure, Dapper consider variables such as the sending rate, Maximum Segment Size (MSS), sender's reaction time (time between received ACK and new transmission), loss rate, latency, congestion window (CWND), receiver window (RWND), and delayed ACKs. Dapper infers sending rate, Maximum Segment Size (MSS), sender's reaction time (time between received ACK and new transmission), loss rate, latency, congestion window (CWND), receiver window (RWND), and delayed ACKs. Based on the inferred variables, Dapper can identify the root cause of the bottleneck. However, Dapper does not provide information about the root cause or location of the failure. The work presented by [76] (SpiderMon) incorporates such capability and reports the root cause and the location of the failure. Moreover, SpiderMon collects diagnosis information without causing any overhead, which makes the system more scalable.

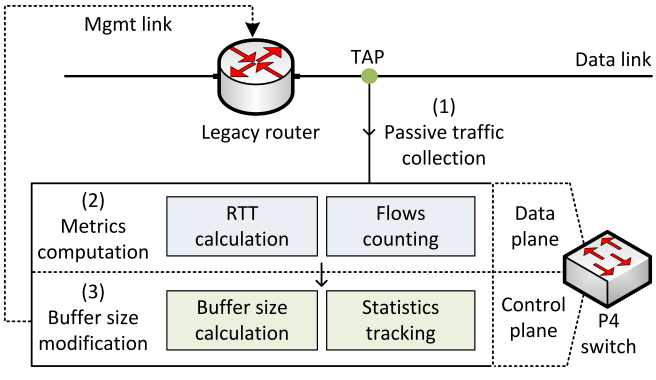

Figure 33: High-level system overview. The P4 switch collects traffic and calculates the RTT per flow. Then, the P4 switch sets the buffer size of a legacy router [168].

Similarly, Blink considers metrics such as the retransmissions rate, packet loss rate, round-trip-time, and out-of-order packets to identify the top-k problematic flows. Moreover, Blink identifies failures considering the predictable behavior of TCP, which retransmits packets at epochs exponentially spaced in time in the presence of failure. For instance, Blink promptly reroutes traffic whenever failure signals are generated by the data plane, while SpiderMon limits the root cause of hosts' sending rate. Schemes such as SpiderMon base the failure identification considering latency increase. Other schemes (e.g., Dapper) leverage reactive processing to identify and mitigate failures. Finally, it is worth mentioning that some systems (e. g., Blink, Dapper) used real-world data, such as the ones provided by CAIDA, to evaluate their systems. Using real-world traces helps to adjust the proposed solution in production network scenarios.

8.1.7. Comparison with Legacy Approaches

The principal difference in network diagnosis schemes between P4-programmable and legacy approaches is the granularity provided by the data plane. Such features enable the implementation of diagnosis schemes that can detect and execute actions at a line rate. Moreover, P4-programmable data planes can process failure events before informing the control plane, such as in [163]. The drawback of P4 schemes against legacy devices is the lack of a protocol that defines the interaction between the control and management planes. Vendors have defined solutions such as NetFlow, JFlow, or Internet standards such as the Simple Network Management Protocol (SMNP). However, these schemes lack granularity. Lastly, P4-based diagnosis schemes can take custom actions (e.g., rerouting) to react to failure events. In legacy devices, packet manipulation is restricted, limiting the control plane's capacity to perform actions.

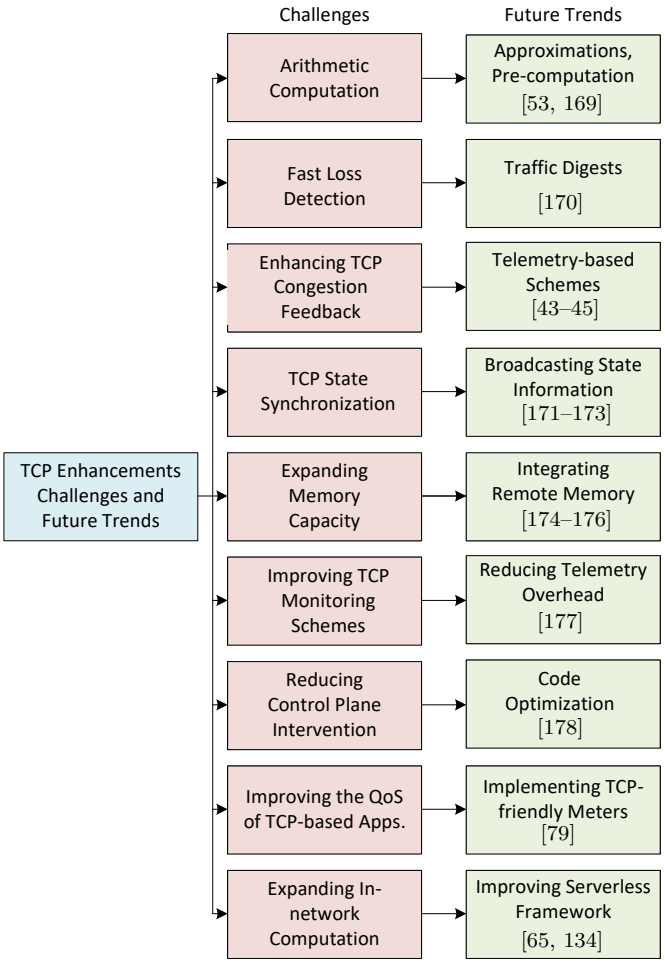

Figure 34: Challenges and future trends. The references represent examples of existing works that tackle the corresponding future trends.

8.2. Summary and Lessons Learned

Network monitoring and measurements are a valuable tool for diagnosing TCP performance degradation and locating the point of failure. The measurements work presented and discussed in previous sections try to minimize overheads, evaluate the severity of the problem, and identify where the failure occurs. Future works should consider integrating with legacy networks, reducing storage requirements, improving the accuracy of the issue evaluation, extending monitoring primitives, and minimizing controller intervention. Some schemes focus on providing a response mechanism upon failure detection to improve QoS by applying traffic policing and management. Techniques adopted include application-layer inspection, traffic metering, traffic separation, and bandwidth management.

9. Challenges and Future Trends

This section presents the challenges and future trends of P4-based schemes that improve TCP performance. The challenges are inferred from the limitations of the surveyed works. The trends include the initiatives that aim at solving those challenges. Figure 34 shows the challenges and their corresponding future trends.

9.1. Arithmetic Computation

Programmable data planes are limited in arithmetic computation, such as floating-point operations, which consume more hardware resources than integer operations and process the inputs in a highly sequential and multistage manner. Therefore, programmers recur to approximation techniques that involve integers, which produce an output at a predictable pace. Furthermore, there are data plane architectures that use approximations to implement filters [179]. For instance, a scheme can use a low-pass filter to smooth the queue occupancy. However, approximating arithmetic operations in the data plane consume significant hardware resources and restricts concurrent execution of data plane programs. Operations demanded by AQMs (e.g., square root function in the CoDel algorithm) are challenging to be implemented with P4 [11, 53, 54]. Extending the supported arithmetic operations will lead to a more accurate estimation of the traffic entropy and computation of TCP statistics (e.g., RTT calculation, CWND's size estimation, and congestion control inference).

Current and Future Initiatives. Approximation and pre-computation methods have been implemented to overcome such limitations. Schemes that use approximations rely on a small set of supported operations to approximate the desired value with a precision penalty. For instance, approximating the square root function can be achieved by counting the number of leading zeros through the longest prefix match [53]. The pre-computation method involves the participation of the control plane to provide the set of values. The results are stored in registers and match action tables, which present limited storage and incurs additional processing time when the control plane intervenes to update them.

Current initiatives such as the one proposed by Swamy *et al.* [169] exploits parallelism based on a map-reduce abstraction. This approach will help programmable network devices, such as switches and NICs, identify and offload complex computations that can be pre-evaluated in the control plane, reducing computation and communication overhead simultaneously. Ding *et al.* [180] implemented in P4 the logarithmic and exponential function entirely in the data plane without using TCAMs, which are scarce resources. Both functions aim at calculating the entropy, which measures the traffic distribution [181]. The entropy calculation contributes to determine performance issues derived from TCP congestion control algorithms [182], load-balancing [183] and network anomalies detection [184, 185].

9.2. Fast Loss Detection

Packet losses are expected in the network, and they can happen for several reasons. In [186], the author quantifies the impact of packet losses in data center environments. Packet losses significantly affect TCP performance, and consequently, operators are interested in identifying the root cause of losses to recover from them. Therefore, it is essential to detect losses fast and independently of their origin to diagnose the network and mitigate the issue. However, legacy monitoring schemes often fail to capture losses quickly, with enough details and low overhead. Current P4 implementations [43, 44, 187] do

not infer the root cause of packet losses, and only react to congestion.

Current and Future Initiatives. P4-programmable data planes provide the flexibility to use custom data structures, enabling the possibility of implementing loss detection schemes. Li *et al.* [170] implemented Lossradar, a fast packet loss detection scheme implemented using programmable data planes that identifies the location of the packet losses. The system is based on traffic digests resulting from comparing traffic meters. If a digest corresponds to the same packet, it cancels each other. Otherwise, it will persist and indicate that a loss has occurred. The system presents low overhead, which is proportional to the number of packet losses. Additionally, the digest includes flow-level information and a packet identifier, which contributes to identifying the source of packet losses. Future works should focus on improving the root case inference algorithm. An option is adding machine learning techniques instead of threshold-based approaches. In this way, the mechanism will classify the type of losses with more accuracy. Furthermore, it could also categorize the type of losses and react according to the event. A future work, should consider a scheme that tolerates losses induced by non-loss based congestion control algorithm (e.g., BBR) since it does not affect the throughput significantly.

9.3. Enhancing TCP Congestion Feedback

Traditional TCP congestion control algorithms do not differentiate the cause of packet drops. The sender considers timeouts of a retransmit timer or receiving three duplicate ACKs as a signal of congestion and reacts by reducing the rate usually according to a multiplicative decrease function. However, packet drops can have different causes such as packet corruption, excessive delay due to the bufferbloat, small TCP receive buffer, and other misconfigurations. Therefore, an improper reaction to losses can lead to the underutilization of network resources. Moreover, with the advent of 5G deployments, there is a need for enhanced mechanisms that reduce packet losses to fulfill the QoS requirements. Although P4-programmable switches enable the possibility of enhancing TCP congestion feedback, the sender must cooperate by understanding the congestion signals. However, the literature reports that even RFC-standardized congestion feedback such as ECN is rarely enabled in production networks [83].

Current and Future Initiatives. Many authors leveraged on P4-programmable data planes aim at enhancing TCP congestion feedback [43–45]. They propose novel schemes that use, for example, INT, EECN, and custom headers to notify the sender about the cause of the congestion promptly. The main goal of such schemes consists of making congestion control algorithms to take agile control decisions and improve robustness. Moreover, P4-based schemes are not restricted to observe local failure. Rather, they can provide a global view of the congestion event.

Schemes that aim at enhancing TCP congestion feedback demonstrated a better performance than legacy approaches. However, many of these schemes are designed and implemented on data center networks, where delays are lower than WANs. Recently, Chen and Nagaoka [188] reported that ECN is not widely adopted in the Internet. They highlight the need to incentivize ISP providers to enable ECN in their deployments. Therefore, future works can explore how P4-based schemes that implement an enhanced feedback mechanism affect TCP performance in a WAN, where the conditions (e.g., delay, loss rate, number of flows, type of traffic) differ from data center environments. Moreover, implementing in P4 congestion controls mechanisms such as eXplicit Control Protocol (XCP) [189] and Rate Control Protocol (RCP) [190], could evidence how enhanced feedback impacts in networks with large BDP.

9.4. TCP State Synchronization

The processing speed mismatch and communication overheads between data and control plane establish a challenge to achieve TCP state synchronization. The data plane operates at line rate and constantly updates the state of packets in the switch. On the other hand, the control plane executes operations to read and manipulate data plane states. Therefore, applications requiring state synchronization to work demand a mechanism that provides a better view of the network. Nevertheless, it is challenging to reach an accurate state synchronization in programmable networks because legacy implementations usually cannot take actions at line rate. For example, tracking the dynamics of TCP flow characteristics is an application that requires accurate state synchronization [191]. Another practical example involves ensuring per-flow consistency, which occurs when load balancers ensure that all TCP packets are delivered to the receiver before changing the route. Such action can only be achieved if the switches along the path have precise information about the state of a TCP connection.

Current and Future Initiatives. Luo [171] *et al.* proposed Swing State, a data plane framework to consistently move/replicate states among devices. The system uses each packet to record state values and pass them to the next P4 switch. Once the states are synchronized, flows can be migrated using any existing network update technique such as [172, 173]. State synchronization contributes to the accurate detection of events such as microbursts and TCP incast. Synchronized measurements facilitate applications to find issues, especially when TCP incast and microburst are present in the network. Yaseen *et al.* [172] highlighted that current schemes that detect such kinds of events are empirical (e.g., packet counting, TCP timeouts, and packet drops detection). Consequently, with an accurate TCP state synchronization, operators can have a global view of the dynamics of the network.

9.5. Expanding Memory Capacity

P4 allows the programmers to store and retrieve data of packets using registers, counters, and meters. This feature enables stateful processing to implement features such as queueing algorithms, classifiers, and policiers, which are essential to implement programs in the data plane. Future works aim to expand the memory capacity to handle a more significant number of flows. However,

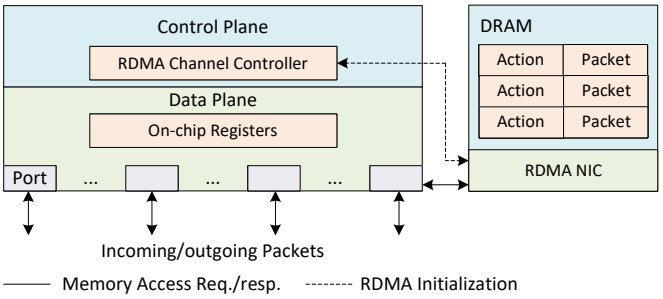

Figure 35: External memory for data plane scheme. The data plane can utilize remote memory region registered RNICs by servers that are connected to the switch. The system comprises a remote lookup table, as an example. Other data structures can also be used [174].

stateful processing is limited by the memory capacity of the switch. For example, schemes like Dapper [75] can not scale to handle more than 10,000 flows because each flow consumes 67 bytes (i.e., sixteen four-byte registers to keep the flow state, a two-byte register to track MSS, and a one-byte register for scale), in addition of 40 bytes of metadata used to carry packet's information. Therefore, to scale a scheme like Dapper, additional fast access memory is required to reduce collisions in the table. Expanding the switch's memory capacity will allow integrating more features in the device.

Another example is BurstRadar [192], that with 300 table entries can handle up to ten simultaneous microbursts with a packet loss rate below 1%. The authors mention that 1000 entries are required to reduce the miss rate to absolute 0%. Microbursts are events in the network that lasts from 10us to 100us and produce intermittent congestion. That congestion increases latency, cause jitter and packet losses in data center networks [193]. Events that produce microbursts include TCP incast and TCP segment offloading. To conclude, the authors remark that microburst is becoming more common as the link speeds are increasing over 10G, while switch buffers sizes do not change.

Current and Future Initiatives. Kim *et al.* [174, 175] proposed expanding the memory capacity of the switch by accessing Dynamic Random Access Memory (DRAM) installed on data center servers directly from the data plane. This approach is flexible since it uses existing resources available in commodity hardware without adding additional costs. The approach presented in Figure 35, is implemented using RoCE, and it only suffers a latency penalty of an extra 1-2 us. Tierney *et al.* [176], tested TCP performance using a similar infrastructure. The experiment consisted of using a dedicated path for RoCE traffic and simultaneously generating TCP traffic. The authors concluded that the ability of RoCE to provide low latency and system overhead makes TCP a compelling technology for high-resolution data transfers. Future works should also consider supporting common data structure in the remote memory. The scheme discussed in this section only implements address-based memory access, meaning they do not support ternary matching and other data layouts in remote memory.

9.6. Improving TCP Monitoring Schemes

Monitoring and diagnosing TCP performance issues is essential for an efficient network operation. Current monitoring system are mainly query [194–196] or sample based [23, 38, 39]. However, such schemes lack from accuracy to correctly track the dynamics of TCP flows (e.g., microburst detection). Network telemetry provides the fundamentals for network management applications, for example, network performance monitoring, debugging, failure localization, load balancing, and security. INT is a network monitoring framework available in programmable data planes that enhances network visibility making it attractive for production networks [43]. However, a major drawback is the telemetry overhead produced on packets due to each switch adds telemetry information to packets which reduces the MSS linearly. Consequently, there could be the case where there is more telemetry information added to the packet than the packet payload, resulting in the application fragmenting a payload into multiple packets and consequently experience reduced performance.

Current and Future Initiatives. Future initiatives should focus on reducing telemetry overhead. A solution could be limiting the telemetry information that each packet can afford. Therefore, developers establish the trade-off between the MSS and the telemetry headers. This is supported by the fact that applications can have good network visibility with approximated telemetry information. For instance, checking a flow's path conformance is possible by analyzing a collection of packets. Another consideration regarding TCP congestion control algorithms is that they can achieve their goal if packets convey information about the path's bottleneck without requiring knowledge about all hops. Lin *et al.* [177] tacked this problem differently by proposing a fine-grained real-time telemetry scheme. The scheme relies on the observation and recording (O&R) of the states of the packet being forwarded by the data plane. The scheme does not measure conventional performance parameters (i.e., end-to-end delay, jitter throughput, and packet loss rate). However, it has a clock offset elimination algorithm to reduce the time synchronization of two adjacent switches. Future works could integrate various monitoring features (e.g., RTT, packet loss, link utilization, failures, etc.) implemented in P4 to monitor if the traffic meets the Service Level Agreements (SLA).

9.7. Reducing Control Plane Intervention

Control-plane operations are essential to support and complement network protocols. However, control plane intervention can result in additional latency and consequently affect TCP performance. For example, rerouting-based monitoring queries often need more storage or access to complex computation. The hardware of the data plane is limited regarding storage and, it is not designed to perform complex operations. Therefore, the control plane intervenes at the cost of losing performance, which should be minimized when it is possible.

Current and Future Initiatives. Control plane intervention can be reduced by optimizing the P4 program to

reduce the communication overhead between the control and data plane. This task is performed by programmers with enough background to create schemes that depend more on the data plane for taking actions. Although schemes that aim at improving TCP can be implemented in the data plane, actions such as rerouting [50, 163] and complex arithmetic operations [54] require control plane intervention. Future initiatives should consider developing tools that detect and reduce the interaction between control and data plane by suggesting alternative workflows and code optimization to minimize such interaction [178].

9.8. Improving the QoS of TCP-based Applications

Providing QoS requires metering the traffic and arbitrarily dropping packets when the rate exceeds the target. Major vendors on the market offer meter functions. Meters measure the arrival rate of a flow and determine its priority as a function of the predefined rate. Packets that exceed the specified rate for the flow are marked with a lower priority and eventually discarded. Nevertheless, meters are usually not TCP-friendly, meaning that they do not regulate the rate properly or increase the loss rate. As a result, the achieved rate of a TCP flow is usually lower than the meter's target rate.

Current and Future Initiatives. Wang *et al.* [79] designed and implemented TCP-friendly meters using programmable data planes. The authors mitigated the bad interaction between the metering algorithm and the TCP congestion control in this work. The scheme ensures that the achieved rate of a TCP flow is close to the target rate established by the meter. The authors mention that future works in this area should consider optimizing data plane resource utilization and test the scheme with different TCP variants. The author also highlighted that the scheme's performance could be further improved if the data plane allows floating-point operations.

9.9. Expanding In-network Computation

With the increasing adoption of high-speed networks and cloud-based applications, server-based processing has become a bottleneck due to its high overhead [197–199] (i.e., processing and management overhead). Kernel bypassing is a solution that reduces overheads, improves performance and flexibility by providing direct communication between the NIC and the application. In this approach, the kernel communication overhead is reduced, and the application performs TCP handling. In this way, multiple applications can rapidly access the NIC. However, with this approach, each application must enforce the correctness of the underlying protocols such as TCP. This limitation implies that TCP congestion control algorithms are implemented in the application. Moreover, since the kernel is being bypassed, the OS cannot perform additional packet processing. Such processing includes demultiplexing packets, egress packet filtering, and resource allocation [134].

Current and Future Initiatives. P4-programmable NICs offer the tools to offload the server's operations, thus performing the computation closer to the network. With the granularity provided by P4, cloud providers could have the ability to provide fast and reliable services. Moreover, cloud providers can use data plane resources to measure resource usage accurately. Current initiatives in P4 propose a serverless framework to offload workloads [65, 72, 200], which includes TCP and other applications. An example of a P4 serverless implementation is λ-NIC, which runs microservices entirely in the NIC. Future works could consider expanding the type of application that can be handled by the NIC, considering that the migration from server to serverless application increases the performance at a marginal cost [201].

Atutxa *et al.* [202] performed in-network computation using P4-programmable data planes. The system aims to reduce time-constrained applications' response times by performing in-network computation. The system processes Message Queueing Telemetry Transport (MQTT) packets originated by an IoT device and generates an alarm when a threshold is exceeded. Results show that response times are reduced by 74% by offloading the computation to the P4 switch.

10. Conclusion

This paper presents a survey on TCP enhancement using P4-programmable devices. It summarizes and discuss recent works on P4-programmable devices, focusing on schemes aimed at enhancing TCP performance. A taxonomy classifying the aspects that affect TCP's behavior, such as congestion control, Active Queue Management (AQM) algorithms, TCP offloading, and network measurements schemes. Then, it compares the works pertaining to each category and with the legacy implementations. Lastly, the challenges and future trends are discussed.

11. Acknowledgement

This work was supported by the U.S. National Science Foundation under grant number 2118311, funded by the Office of Advanced Cyberinfrastructure (OAC).

Table 15: Abbreviations used in this Survey.

Abbreviation	Term
5G	Fifth Generation
ABC	Activity-Based Congestion
ACK	Acknowledgement
AFQ	Approximate Fair Queueing
AIMD	Additive-Increase Multiplicative-Decrease
ALU	Arithmetic Logical Unit
API	Application Programming Interface
AQM	Active Queue Management
ASIC	Application-specific Integrated Circuit
AWW	Adjusting Advertised Window
BBR	Bottleneck Bandwidth and Round-trip Time
BDP	Bandwidth-Delay Product
BMv2	Behavioral Model Version 2
CAIDA	Center for Applied Internet Data Analysis
CAKE	Common Application Kept Enhanced
CDN	Content Delivery Network
CLI	Command-line Interface
CoDel	Controlled Delay
CPU	Central Processing Unit
CWND	Congestion Window
DCQCN	Data Center Quantified Congestion Notification
DCTCP	Data Center Transmission Control Protocol
DDL	Distributed Deep Learning
DMA	Direct Memory Access
DMZ	Demilitarized Zone

Abbreviation	Term
DNS	Domain Name Server
DPDK	Data Plane Development Kit
DRAM	Dynamic Random-Access Memory
DRR	Deficit Round-Robin
ECN	Explicit Congestion Notification
EECN	Enhanced Explicit Congestion Notification
FAB	Flow-Aware Buffer
FDPA	Fair Dynamic Priority Assignment
FCT	Flow Completion Time
FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
FPGA	Field-Programmable Gate Array
FQ	Fair Queueing
FQ-CoDel	Fair Queueing Controlled Delay
GPU	Graphics Processing Unit
GSO	Generic Segmentation Offload
HOL	Head-of-Line
HPCC	High Precision Congestion Control
IETF	Internet Engineering Task Force
INT	In-band Network Telemetry
IP	Internet Protocol
L4S	Low-Latency Low-Loss Scalable-throughput
LRO	Large Segment Offload
LLEP	Link Latency Estimation Protocol
MIMD	Multiplicative Increase Multiplicative Decrease
MQTT	Message Queueing Telemetry Transport
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NACK	Negative Acknowledgement
NCF	Network-assisted Congestion Feedback
NDP	Novel Data center Protocol
NetFPGA	Network Field Programmable Gate Array
NF	Network Functions
NFV	Network Functions Virtualization
NIC	Network Interface Controller
NPU	Network Processing Unit
NSF	National Science Foundation
OAC	Office of Advanced Cyberinfrastructure
OS	Operating System
OVS	Open Virtual Switch
P4	Programming Protocol-independent Packet Processors
PCC	Performance-oriented Congestion Control
PFC	Priority-based Flow Control
PIE	Proportional-Integral controller Enhanced
PIFO	Push-In First-Out
PINT	Probabilistic In-band Network Telemetry
PISA	Protocol Independent Switch Architecture
PL2	Predictable Low Latency
PPV	Per-Packet Value
PS	Parameter Server
QCN	Quantized Congestion Notification
QoS	Quality of Service
RDMA	Remote Direct Memory Access
RISC	Reduced Instruction Set Computer
RoCE	RDMA over Converged Ethernet
RED	Random Early Detection
RFC	Request For Comments
RLN	Rate-Limiting Notification
RPC	Remote Procedure Call
RTT	Round-Trip Time
RWND	Receiver Window
SDE	Software Development Environment
SDN	Software-Defined Networking
SDL	Secure Development Life-cycle
SLA	Service Level Agreements
SP-PIFO	Strict Priority Push-In First-Out
SPS	Strict Priority Scheduler
SRAM	Static Random-access Memory
SRPT	Shortest Remaining Processing Time
SR-IOV	Single-Root Input/Output Virtualization
SSL	Secure Sockets Layer
TCM	Three-Colors Marker
TCP	Transmission Control Protocol
TD	Tail-Drop
TLS	Transport Layer Security
TM	Traffic Management
TOE	TCP-Offload Engine
ToR	Top of Rack
trTCM	Two Rate Three Color Marker meter
TSO	TCP Segmentation Offloading
TTL	Time to Live
WAN	Wide Area Network

References

[1] J. Postel, "RFC 793: Transmission control protocol (TCP)," September 1981.

[2] F-stack team, "Website of F-stack." [Online]. Available: `http://www.f-stack.org/`, Accessed on 07-01-2021.

[3] OpenOnload project, "Website of OpenOnload." [Online]. Available: `https://tinyurl.com/ykvc3ctc`, Accessed on 07-01-2021.

[4] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "MTCP: a highly scalable user-level TCP stack for multicore systems," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014.

[5] I. Marinos, R. N. Watson, and M. Handley, "Network stack specialization for performance," *ACM SIGCOMM Computer Communication Review*, 2014.

[6] 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*, 2014.

[7] Nick McKeown, "How we might get humans out of the way, ONF connect 19," 2019. [Online]. Available: `https://tinyurl.com/y4dnxacz`, Accessed on 07-01-2021.

[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, 1993.

[9] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, 2013.

[10] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "RFC 8290: The flow queue CoDel packet scheduler and active queue management algorithm," 2018.

[11] T. Hoeiland-Jorgensen, D. Taht, and J. Morton, "Piece of CAKE: a comprehensive queue management solution for home gateways," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2018.

[12] S. Keshav, "On the efficient implementation of fair queueing," *Internetworking: Research and Experience*, 1991.

[13] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *ACM SIGCOMM Computer Communication Review*, 2003.

[14] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Communications Surveys & Tutorials*, 2017.

[15] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: abstractions, architectures, and open problems," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018.

[16] A. Satapathy, "Comprehensive study of P4 programming language and software-defined networks," 2018. [Online]. Available: `https://tinyurl.com/y4d4zma9`, Accessed on 07-01-2021.

[17] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, 2019.

[18] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, 2020.

[19] 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.

[20] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, 2021.

[21] 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.

[22] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński, "Enhancing 5G SDN/NFV edge with P4 data plane programmability," *IEEE Network*, 2021.

[23] R. Sommer and A. Feldmann, "NetFlow: Information loss or win?," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002.

[24] B. Lee, H. Son, S. Yoon, and Y. Lee, "End-to-end flow monitoring with IPFIX," in *Asia-Pacific Network Operations and Management Symposium*, 2007.

[25] P. W. Group, "Charter of the P4 architecture WG." [Online]. Available: `https://tinyurl.com/4c8yt67v`, Accessed on 07-01-2021.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[27] T. P. A. W. Group, "P4$_{16}$ Portable Switch Architecture (PSA)." [Online]. Available: `https://tinyurl.com/tszc5d9c`, Accessed on 07-01-2021.

[28] N. McKeown, "Why does the Internet need a programmable forwarding plane." [Online]. Available: `https://tinyurl.com/y6x7qqpm`, Accessed on 07-01-2021.

[29] H. Garcia and M. Naercio, "A state consistency framework leveraging packet cloning and piggybacking for programmable network data planes," in *2021 IFIP Networking Conference*, 2021.

[30] Barefoot Networks, "P4 Intel Tofino native architecture - public version." [Online]. Available: `https://tinyurl.com/5d7nznwd`, Accessed on 03-12-2022.

[31] A. Agrawal and C. Kim, "Intel Tofino2-A 12.9 Tbps P4-Programmable Ethernet switch," in *Hot Chips Symposium*, 2020.

[32] P4.org, "P4 language tutorial, slide 7." [Online]. Available: `https://tinyurl.com/udonnx5`, Accessed on 07-01-2021.

[33] Open Virtual Switch Community, "P4-OVS." [Online]. Available: `https://tinyurl.com/x45c5pfk`, Accessed on 07-01-2021.

[34] Orange Labs, "P4rt-OVS." [Online]. Available: `https://github.com/Orange-OpenSource/p4rt-ovs`, Accessed on 07-01-2021.

[35] Open Netorking Foundation (ONF), "Website of the open network fundation." [Online]. Available: `https://opennetworking.org`.

[36] P4 Consortium, "Website of the P4 language consortium." [Online]. Available: `https://p4.org`.

[37] C. Kim, "Evolution of Networking, Networking Field Day 21, 2:01," 2019. [Online]. Available: `https://tinyurl.com/y9fkj7qx`, Accessed on 07-01-2021.

[38] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, 2004.

[39] B. Claise, M. Fullmer, P. Calato, and R. Penno, "IPFIX protocol specification," *Internet-draft*, 2005.

[40] 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*, 2019.

[41] 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*, 2017.

[42] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, and G. Srivastava, "Enabling TCP pacing using programmable data plane switches," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, 2019.

[43] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, *et al.*, "HPCC: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.

[44] 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)*, 2020.

[45] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," 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*, 2020.

[46] 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*, 2018.

[47] B. Turkovic and F. Kuipers, "P4air: Increasing fairness among competing congestion control algorithms," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020.

[48] Y.-W. Chen, L.-H. Yen, W.-C. Wang, C.-A. Chuang, Y.-S. Liu, and C.-C. Tseng, "P4-enabled bandwidth management," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019.

[49] M. Apostolaki, L. Vanbever, and M. Ghobadi, "FAB: Toward flow-aware buffer sharing on programmable switches," in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019.

[50] 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*, 2018.

[51] J. Geng, J. Yan, and Y. Zhang, "P4QCN: Congestion control using P4-capable device in data center networks," *Electronics*, 2019.

[52] 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)*, 2019.

[53] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, "P4-CoDel: Active queue management in programmable data planes," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018.

[54] R. Kundel, A. Rizk, J. Blendin, B. Koldehofe, R. Hark, and R. Steinmetz, "P4-CoDel: Experiences on programmable data plane hardware," *arXiv preprint arXiv:2010.04528*, 2020.

[55] I. Kunze, M. Gunz, D. Saam, K. Wehrle, and J. Rüth, "Tofino+ P4: A strong compound for AQM on high-speed networks?," 2021.

[56] C. Papagianni and K. De Schepper, "PI2 for P4: An active queue management scheme for programmable data planes," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, 2019.

[57] L. Toresson, "Making a packet-value based aqm on a programmable switch for resource-sharing and low latency," *Karlstad University, Master's Thesis*, 2021.

[58] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.

[59] A. Mushtaq, R. Mittal, J. McCauley, M. Alizadeh, S. Ratnasamy, and S. Shenker, "Datacenter congestion control: Identifying what is essential and making it practical," *ACM SIGCOMM Computer Communication Review*, 2019.

[60] M. Menth, H. Mostafaei, D. Merling, and M. Häberle, "Implementation and evaluation of activity-based congestion management using P4 (P4-ABC)," *Future Internet*, 2019.

[61] A. G. Alcoz, A. Dietmüller, and L. Vanbever, "SP-PIFO: approximating push-in first-out behaviors using strict-priority queues," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[62] C. Cascone, N. Bonelli, L. Bianchi, A. Capone, and B. Sansò, "Towards approximate fair bandwidth sharing via dynamic priority queuing," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2017.

[63] H. Harkous, C. Papagianni, K. De Schepper, M. Jarschel, M. Dimolianis, and R. Preis, "Virtual queues for P4: A poor man's programmable traffic manager," *IEEE Transactions on Network and Service Management*, 2021.

[64] B. Turkovic, S. Biswal, A. Vijay, A. Hüfner, and F. Kuipers, "P4QoS: QoS-based packet processing with P4," in *NetSoft 2021-IEEE International Conference on Network Softwarization*, 2021.

[65] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "Acceltcp: Accelerating network applications with stateful TCP offloading," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI20)*, 2020.

[66] Y. Yan, A. F. Beldachi, R. Nejabati, and D. Simeonidou, "P4-enabled smart NIC: Enabling sliceable and service-driven optical data centres," *Journal of Lightwave Technology*, 2020.

[67] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with predictable packet processing performance," *IEEE Transactions on Network and Service Management*, 2020.

[68] Y. Qiu, Q. Kang, M. Liu, and A. Chen, "Clara: Performance clarity for SmartNIC offloading," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020.

[69] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani,

V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, *et al.*, "FlowBlaze: Stateful packet processing in hardware," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019.

[70] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive serverless compute on SmartNICs," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, 2019.

[71] P. Gao, Y. Xu, and H. J. Chao, "OVS-CAB: Efficient rule-caching for open vswitch hardware offloading," *Computer Networks*, 2021.

[72] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, and K. Rothermel, "P4CEP: Towards in-network complex event processing," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 33–38, 2018.

[73] A. Mohammadkhan, S. Panda, S. G. Kulkarni, K. Ramakrishnan, and L. N. Bhuyan, "P4NFV: P4 enabled NFV systems with SmartNICs," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2019.

[74] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, N. McKeown, and C. Kim, "The nanoPU: Redesigning the CPU-network interface to minimize RPC tail latency," *arXiv preprint arXiv:2010.12114*, 2020.

[75] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: data plane performance diagnosis of TCP," in *Proceedings of the Symposium on SDN Research*, 2017.

[76] 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*, 2020.

[77] ETH Zurich, "Blink." [Online]. Available: `https://blink.ethz.ch/`, , Accessed on 07-01-2021.

[78] X. Chen, H. Kim, J. M. Aman, W. Chang, M. Lee, and J. Rexford, "Measuring TCP round-trip time in the data plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, 2020.

[79] S.-Y. Wang, H.-W. Hu, and Y.-B. Lin, "Design and implementation of TCP-Friendly meters in P4 switches," *IEEE/ACM Transactions on Networking*, 2020.

[80] 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*, 2019.

[81] J. Kim, H. Kim, and J. Rexford, "Analyzing traffic by domain name in the data plane," [Online]. Available: `https://tinyurl.com/2zyxt67k`.

[82] Parallel data lab, "Incast." [Online]. Available: `https://www.pdl.cmu.edu/Incast/`, Accessed on 07-01-2021.

[83] Y. Li, *Thesis: Hardware-Software Codesign for High-Performance Cloud Networks*. PhD thesis, Harvard University, 2020.

[84] J. Zhang, F. Ren, L. Tang, and C. Lin, "Modeling and solving TCP incast problem in data center networks," *IEEE Transactions on Parallel and Distributed systems*, 2014.

[85] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, 1989.

[86] R. K. Jain, D.-M. W. Chiu, W. R. Hawe, *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[87] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, 2016.

[88] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *Proceedings of the International Conference on Communications*, 1979.

[89] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, 1994.

[90] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "Proactive congestion avoidance for distributed deep learning," *Sensors*, 2021.

[91] A. Laraba, J. François, S. R. Chowdhury, I. Chrisment, and R. Boutaba, "Mitigating TCP protocol misuse with programmable data planes," *IEEE Transactions on Network and Service Management*, 2021.

[92] 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*, 2010.

[93] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, 2015.

[94] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "ECN or delay: Lessons Learnt from Analysis of DCQCN and TIMELY," in *Proceedings of the 12th International on Conference on emerging Networking Experiments and Technologies*, 2016.

[95] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP congestion control with a misbehaving receiver," *ACM SIGCOMM Computer Communication Review*, 1999.

[96] K. Ramakrishnan, S. Floyd, D. Black, *et al.*, "The addition of explicit congestion notification (ECN) to IP," 2001.

[97] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, 2006.

[98] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, 2000.

[99] J. Crichigno, E. Bou-Harb, and N. Ghani, "A comprehensive tutorial on science DMZ," *IEEE Communications Surveys & Tutorials*, 2018.

[100] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009.

[101] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Interactions between congestion control algorithms," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019.

[102] 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*, 2016.

[103] Y. Le, R. N. Mysore, L. Suresh, G. Zellweger, S. Banerjee, A. Akella, and M. Swift, "PL2: Towards predictable low latency in rack-scale networks," *arXiv preprint arXiv:2101.06537*, 2021.

[104] J. Networks, "Junos OS CoS components." [Online]. Available: `https://tinyurl.com/Junos-CoS`, Accessed on 07-01-2021.

[105] S. Arslan and N. McKeown, "Switches know the exact amount of congestion," in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019.

[106] S. Nádas, G. Gombos, P. Hudoba, and S. Laki, "Towards a congestion control-independent core-stateless AQM," in *Proceedings of the Applied Networking Research Workshop*, 2018.

[107] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great internet TCP congestion control census," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2019.

[108] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Queue*, 2011.

[109] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM computer communication review*, 1988.

[110] G. Vu-Brugier, R. Stanojevic, D. J. Leith, and R. N. Shorten, "A critique of recently proposed buffer-sizing strategies," *ACM SIGCOMM Computer Communication Review*, 2007.

[111] K. Nichols and V. Jacobson, "Controlling queue delay: A modern AQM is just one piece of the solution to bufferbloat.," *Queue*, 2012.

[112] S. Floyd, R. Gummadi, S. Shenker, *et al.*, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," 2001.

[113] C. Press, "Cisco IP telephony flash cards: Weighted random early detection (WRED)." [Online]. Available: `https://tinyurl.com/2f5st8rd`, Accessed on 07-01-2021.

[114] S. Ryu, C. Rump, and C. Qiao, "Advances in active queue management (AQM) based TCP congestion control," *Telecommunication Systems*, 2004.

[115] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with very small buffers.," in *INFOCOM*, 2006.

[116] C. Villamizar and C. Song, "High performance TCP in ANSNET," *ACM SIGCOMM Computer Communication Review*, 1994.
[117] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, 2004.
[118] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "RFC 8289: Controlled delay active queue management," 2018.
[119] D. Taht, J. Gettys, and E. Dumazet, "RFC 8290: The flow queue CoDel packet scheduler and active queue management algorithm," 2018.
[120] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, "PI2: A linearized AQM for both classic and scalable TCP," in *Proceedings of the 12th International on Conference on emerging Networking Experiments and Technologies*, 2016.
[121] S. Nádas, Z. R. Turányi, and S. Rácz, "Per packet value: A practical concept for network resource sharing," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016.
[122] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.
[123] S. Laki, P. Vörös, and F. Fejes, "Towards an AQM evaluation testbed with P4 and DPDK," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, 2019.
[124] D. R. Smith, "A new proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, 1978.
[125] C. Chen, H.-C. Fang, and M. S. Iqbal, "QoSTCP: Provide consistent rate guarantees to TCP flows in software defined networks," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, 2020.
[126] Microsoft, "TCP/IP offload." [Online]. Available: `https://tinyurl.com/yedss7sn`, Accessed on 07-01-2021.
[127] J. C. Mogul, "TCP offload is a dumb idea whose time has come.," 2003.
[128] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.
[129] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.
[130] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 2009.
[131] L. Quan and J. Heidemann, "On the characteristics and reasons of long-lived internet flows," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.
[132] V. Jacobson, R. Braden, and D. Borman, "RFC 1323: TCP extensions for high performance," tech. rep., 1992.
[133] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017.
[134] A. Kaufmann, "Efficient, secure, and flexible high speed packet processing for data centers," *University of Washington, Ph.D. Thesis*, 2018.
[135] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Accelerating virtual network functions with fast-slow path architecture using express data path," *IEEE Transactions on Network and Service Management*, 2020.
[136] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "TAS: TCP acceleration as an OS service," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019.
[137] The Linux Foundation Projects, "Data plane development kit (dpdk)." [Online]. Available: `http://dpdk.orgd`, Accessed on 03-14-2022.
[138] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155*, 2015.
[139] Pensando, "The Pensando distributed services platform." [Online]. Available: `https://pensando.io/our-platform/`, Accessed on 07-01-2021.
[140] Open compute project SAI, "Programming NFP with P4 and C." [Online]. Available: `https://www.netronome.com/products/agilio-cx/`, Accessed on 07-01-2021.
[141] Xilinx, "Xilinx solutions." [Online]. Available: `https://www.xilinx.com/products/silicon-devices.html`, Accessed on 07-01-2021.
[142] Innovium, "Teralynx switch silicon." [Online]. Available: `https://www.innovium.com/teralynx/`, Accessed on 07-01-2021.
[143] Redis Labs, "Redis." [Online]. Available: `https://redis.io/`, Accessed on 07-01-2021.
[144] HAProxys, "HAProxy: The reliable, high performance TCP/HTTP load balancer." [Online]. Available: `http://www.haproxy.org/`, Accessed on 07-01-2021.
[145] N. Gray, A. Grigorjew, T. Hosssfeld, A. Shukla, and T. Zinner, "Highlighting the gap between expected and actual behavior in P4-enabled networks," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.
[146] G. W. Connery, W. P. Sherer, G. Jaszewski, and J. S. Binder, "Offload of TCP segmentation to a smart adapter," 1999.
[147] J. Corbet, "JLS2009: Generic receive offload." [Online]. Available: `https://tinyurl.com/3bsvrwyk`, Accessed on 07-01-2021.
[148] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.
[149] T. Osiński, M. Kossakowski, H. Tarasiuk, and R. Picard, "Offloading data plane functions to the multi-tenant cloud infrastructure using P4," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019.
[150] M. A. Jamshed, Y. Moon, D. Kim, D. Han, and K. Park, "mOS: A reusable networking stack for flow monitoring middleboxes," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.
[151] G. Liu, Y. Ren, M. Yurchenko, K. Ramakrishnan, and T. Wood, "Microboxes: high performance NFV with customizable, asynchronous TCP stacks and dynamic subscriptions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.
[152] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. Lakshman, "UNO: Uniflying host and smart NIC offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017.
[153] Mellanox, "BlueField SmartNIC Ethernet." [Online]. Available: `https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet`, Accessed on 07-01-2021.
[154] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, *et al.*, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.
[155] N. McKeown, "Programming the forwarding plane. Why is there a blackbox in my whitebox?." [Online]. Available: `https://forum.stanford.edu/events/2016/slides/plenary/Nick.pdf`, Accessed on 07-01-2021.
[156] R. H. Inc., "ftrace - function tracer." [Online]. Available: `https://tinyurl.com/2z55r78r`, Accessed on 07-01-2021.
[157] T. T. Group, "Manual's page of tcpdump." [Online]. Available: `http://www.tcpdump.org/tcpdump_man.html`, Accessed on 07-01-2021.
[158] T. linux foundation, "TCP probe." [Online]. Available: `https://wiki.linuxfoundation.org/networking/tcpprobe`, Accessed on 07-01-2021.
[159] E. F. Kfoury, J. Gomez, J. Crichigno, and E. Bou-Harb, "An emulation-based evaluation of TCP BBRv2 alpha for wired broadband," *Computer Communications*, 2020.
[160] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "BBRv2: a model-based congestion control," *Presentation in the Internet Congestion Control Research Group (ICCRG) at IETF 105 Update, Montreal, Canada, July, 2019.*
[161] I. Cisco, "Cisco visual networking index: Forecast and methodology, 2011–2016," *CISCO White paper*, 2012.
[162] Cisco, "COVID-19 network traffic patterns: A worldwide perspective from our customers," *URL:*

https://tinyurl.com/ytmr56ka, 2020.

[163] 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)*, 2019.

[164] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred, "Taking the blame game out of data centers operations with netpoirot," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[165] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, and J. Rexford, "Memory-efficient performance monitoring on programmable switches with lean algorithms," in *Symposium on Algorithmic Principles of Computer Systems (APoCS)*, 2020.

[166] H. Unbehauen, *Control systems, robotics and automation*. Eolss Publishers Company Limited Oxford, 2009.

[167] X. Chen and H. Kim, "Technical report: Measuring queues in campus network via link tapping," *Princeton University*, 2019.

[168] E. Kfoury, J. Crichigno, and G. Srivastava, "Dynamic router's buffer sizing using passive measurements and P4 programmable switches," in *IEEE GLOBECOM (Global Communications Conference)*, 2021.

[169] T. Swamy, A. Rucker, M. Shahbaz, and K. Olukotun, "Taurus: An intelligent data plane," *arXiv preprint arXiv:2002.08987*, 2020.

[170] Y. Li, R. Miao, C. Kim, and M. Yu, "Lossradar: Fast detection of lost packets in data center networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016.

[171] S. Luo, H. Yu, and L. Vanbever, "Swing state: Consistent updates for stateful and programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2017.

[172] N. Yaseen, J. Sonchack, and V. Liu, "Synchronized network snapshots," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.

[173] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, 2012.

[174] 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*, 2018.

[175] 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*, 2020.

[176] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul, "Efficient data transfer protocols for big data," in *2012 IEEE 8th International Conference on E-Science*, 2012.

[177] W.-H. Lin, W.-X. Liu, G.-F. Chen, S. Wu, J.-J. Fu, X. Liang, S. Ling, and Z.-T. Chen, "Network telemetry by observing and recording on programmable data plane," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021.

[178] E. C. Molero, S. Vissicchio, and L. Vanbever, "Hardware-accelerated network control planes," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018.

[179] O. Foundation, "P416 programming for Intel Tofino using Intel P4 studio." [Online]. Available: `https://tinyurl.com/jvtvypdw`, Accessed on 07-01-2021.

[180] D. Ding, M. Savi, and D. Siracusa, "Estimating logarithmic and exponential functions to track network traffic entropy in P4," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, 2020.

[181] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," *ACM SIGMETRICS Performance Evaluation Review*, 2006.

[182] L. Jiang, D. Shah, J. Shin, and J. Walrand, "Distributed random access algorithm: scheduling and congestion control," *IEEE Transactions on Information Theory*, 2010.

[183] K. Wu, L. Chen, S. Ye, and Y. Li, "A load balancing algorithm based on the variation trend of entropy in homogeneous cluster," *International journal of grid and distributed computing*, 2014.

[184] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[185] J. Crichigno, E. Kfoury, E. Bou-Harb, N. Ghani, Y. Prieto, C. Vega, J. Pezoa, C. Huang, and D. Torres, "A flow-based entropy characterization of a NATed network and its application on intrusion detection," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019.

[186] J. Dean, "Designs, lessons and advice from building large distributed systems," *Keynote from LADIS*, 2009.

[187] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communications surveys & tutorials*, 2010.

[188] C.-X. Chen and K. Nagaoka, "Analysis of the state of ECN on the Internet," *IEICE Transactions on Information and Systems*, 2019.

[189] D. Kitabi, M. Handley, and C. Rohrs, "Internet congestion control for high bandwidth-delay product networks," 2002.

[190] N. Dukkipati, *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.

[191] B. Tschaen, Y. Zhang, T. Benson, S. Banerjee, J. Lee, and J.-M. Kang, "SFC-checker: Checking the correct forwarding behavior of service function chaining," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[192] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, 2018.

[193] D. Shan, F. Ren, P. Cheng, and R. Shu, "Micro-burst in data centers: Observations, implications, and applications," *arXiv preprint arXiv:1604.07621*, 2016.

[194] 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)*, 2018.

[195] 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*, 2018.

[196] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[197] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[198] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Communications of the ACM*, 2019.

[199] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in high-speed NICs," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[200] Y. G. Moon, I. Park, S. Lee, and K. S. Park, "Accelerating flow processing middleboxes with programmable NICs," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, pp. 1–3, 2018.

[201] A. Goli, O. Hajihassani, H. Khazaei, O. Ardakanian, M. Rashidi, and T. Dauphinee, "Migrating from monolithic to serverless: A fintech case study," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2020.

[202] A. Atutxa, D. Franco, J. Sasiain, J. Astorga, and E. Jacob, "Achieving low latency communications in smart industrial networks with programmable data planes," 2021.