future internet

MDPI

Article

Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)

Michael Menth [1], Habib Mostafaei [2,*,†], Daniel Merling [1] and Marco Häberle [1]

[1] Department of Computer Science, University of Tuebingen, 72076 Tubingen, Germany
[2] TU Berlin, 10623 Berlin, Germany
* Correspondence: habib@inet.tu-berlin.de
† Part of this work was conducted while this author was with Roma Tre University, Italy.

Received: 20 June 2019; Accepted: 17 July 2019; Published: 19 July 2019

Abstract: Activity-Based Congestion management (ABC) is a novel domain-based QoS mechanism providing more fairness among customers on bottleneck links. It avoids per-flow or per-customer states in the core network and is suitable for application in future 5G networks. However, ABC cannot be configured on standard devices. P4 is a novel programmable data plane specification which allows defining new headers and forwarding behavior. In this work, we implement an ABC prototype using P4 and point out challenges experienced during implementation. Experimental validation of ABC using the P4-based prototype reveals the desired fairness results.

Keywords: Activity-based congestion management (ABC); programmable data plane; QoS

1. Introduction

Future mobile networks like 5G consist of small cells that issue large traffic rates with high fluctuations [1]. As quality of service (QoS) is required, economic provisioning of the transport network is a challenge. Datacenter networks and residential access networks of Internet service providers (ISPs) have similar requirements [2–4].

To avoid QoS degradation, scalable bandwidth-sharing mechanisms for congestion management may be helpful, but they need to be simple and effective. That means, light users should be protected against overload caused by heavy users while avoiding per-user signaling and information within the transport network for complexity reasons.

In [5], activity-based congestion management (ABC) was initially suggested for that purpose. It implements a domain concept where edge nodes run an activity meter that measures the traffic rates of users and add activity information to their packets. Forwarding nodes leverage activity-based active queue management (activity AQM) which uses this information to preferentially drop packets from most active users in case of congestion. In [6], ABC has been proposed in its current form and extensive simulation results have demonstrated that ABC can effectively protect light users against heavy users to such an extent that a single TCP connection from a light user does not significantly suffer in the presence of congestion caused by an aggressive non-responsive traffic stream of a heavy user.

As ABC requires additional header information and new features in edge nodes and forwarding nodes, it cannot be configured on conventional networking gears. However, advances in network programmability support the definition of new headers and node behavior. The network programming language P4 is a notable example [7].

In this work, we report about a P4-based prototype for ABC. It demonstrates the technical feasibility of ABC while revealing challenges and giving hints for the enhancement of P4 support on switches. Furthermore, we present experimental results which confirm the simulative findings in [6].

The remainder of the paper is structured as follows. Section 2 briefly reviews related work. Section 3 explains the ABC concept in detail. Section 4 gives an introduction to SDN and P4. Section 5 describes the P4-based ABC implementation. Section 6 presents our evaluation methodology and reports experimental results. Finally, Section 7 concludes this work.

2. Related Work

A comprehensive overview of congestion management techniques can be found in [8]. Here, we discuss only approaches that are highly related to ABC.

Scheduling algorithms like Weighted Fair Queueing (WFQ) also manage congestion among traffic aggregates in a fair way. However, they require per-aggregate state information. In contrast, ABC requires that information only on ingress nodes of a domain, which keeps the core network simple and allows better scaling.

Core-Stateless Fair Queueing (CSFQ) [9] also improves fairness in core networks without per-flow state. Edge nodes meter the traffic rate of flows or users and record them in packet headers. Forwarding nodes leverage this information together with online rate measurement to detect congestion and to determine suitable drop probabilities for packets. In contrast to ABC, CSFQ requires more complex actions in core nodes and is less efficient [6].

Rainbow-Fair Queuing (RFQ) [10] marks packets at the edge with different colors whereby colors correspond to activities in ABC. A major difference to ABC is that packets of a flow are colored with random values from a range instead of with one specific value. The same is pursued by the fair activity meter in ABC [6], but simulations have shown that fair activity metering degrades fairness for congestion-controlled traffic. PPV [11] adopts the ideas of RFQ and is discussed as a base mechanism for "Statistical Behaviors" which are new service level specifications that are currently discussed in Metro Ethernet Forum (MEF). They differentiate the impact of congestion among different flows within a single service class while avoiding per-aggregate states in core networks. The same authors adapt PPV to deploy it in broadband access networks [12], take user activity on various time scales into account [13], and target 5G networks [14].

Fair Dynamic Priority Assignment (FDPA) [15] is a fair bandwidth sharing method for TCP-like senders which is implemented in OpenFlow and P4. Its objective is to make scheduling more scalable, but it still requires per-user state in forwarding nodes. While FDPA is applicable only for responsive traffic, ABC works with responsive traffic, non-responsive traffic, and combinations thereof.

Approximate per-flow fair-queueing (AFQ) is proposed in [16] to maintain bandwidth fairness on flow level. The authors describe how AFQ can be implemented on configurable switches in general and provide an implementation in P4. In contrast to ABC, AFQ requires per-flow state in core devices and guarantees per-flow fairness while ABC offers per-user fairness.

In [17], the authors propose CoDel (controlled delay). CoDel minimizes the time packets spend in queue by periodically checking whether the smallest sojourn time of all queued packets is below a certain threshold. If the threshold is exceeded, a packet is dropped and the time interval for the next check is decreased. As soon as the threshold is not exceeded anymore, dropping packets stops and the time interval is reset. In [18] CoDel is implemented in P4 for the software switch BMv2. Although CoDel requires floating point calculations, the authors avoid extern functions by leveraging a workaround that requires a significant amount of table entries.

In [19] the authors propose a congestion avoidance mechanism that they implement in P4. It leverages pre-established alternative paths to decrease the load on congested routes. When a network device detects that a latency-critical flow is in danger to be delayed due to queueing delay, it redirects the traffic of the affected flow to a pre-established alternative path. The authors evaluate their P4 implementation, which does not require extern functions on either the P4 software switch BMv2 or the Netronome Agilio CX SmartNIC. In contrast to ABC, the implementation utilizes a local agent on each network device to reply to congestion in a timely manner.

So far, there is only a small number of P4 implementations for congestion management mechanisms in the literature, and most of them have been demonstrated on software switches. In contrast, for monitoring purposes, there are many novel concepts based on P4 technology, and quite a few have been prototyped on hardware platforms. The work in [20] proposes a monitoring mechanism for detecting microbursts in datacenter networks at line rate. It has been implemented for the P4 hardware switch Tofino. In [21] the authors present a P4 implementation for the BMv2 software switch that monitors the network state in real-time without requiring probing packets. In a similar context the authors of [22] introduce a P4-based mechanism for the BMv2 that leverages bloom filter for both flow tracking on a single device and full path tracking. The work in [23] proposes IDEAFIX, a monitoring mechanism that identifies elephant flows in IXP networks by analyzing flow features in edge switches. IDEAFIX has been implemented for the P4 BMv2 software switch.

3. Activity-Based Congestion Management (ABC)

We give an overview of ABC, introduce the activity meter and the activity AQM in more detail, and discuss properties of ABC.

3.1. ABC Overview

ABC features a domain concept which is shown in Figure 1. Ingress nodes leverage activity meters to measure the rate of traffic aggregates that enter the ABC domain. They derive 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.

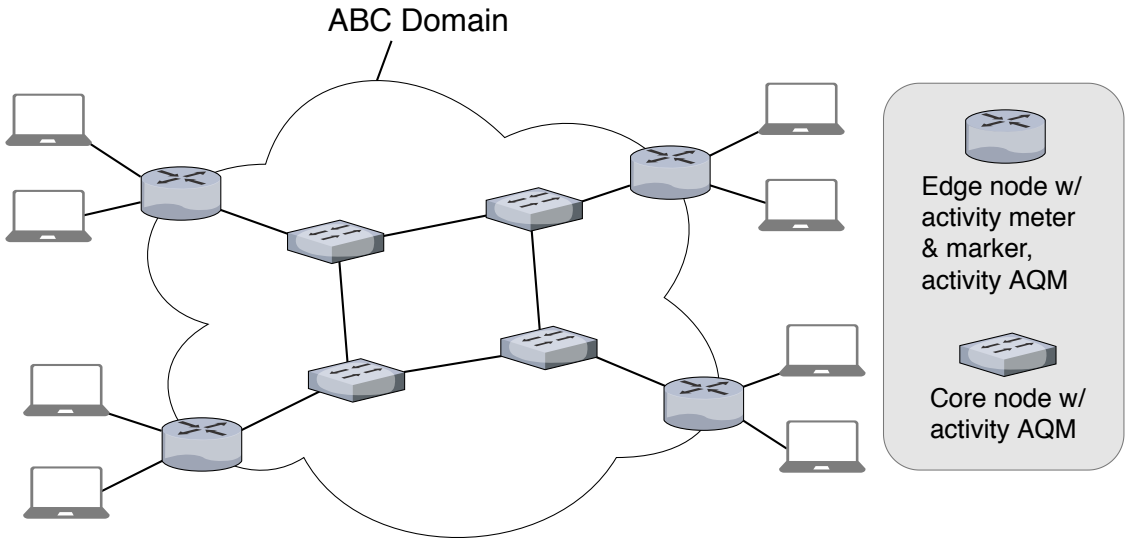

Figure 1. Activity metering and marking is performed only by ingress nodes. Both ingress and core nodes apply activity active queue management (AQM) during packet forwarding.

Ingress nodes and core nodes of an ABC domain are forwarding nodes. They use an activity AQM on each of their outgoing 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. This enforces fair resource sharing among traffic aggregates within an ABC domain.

Egress nodes just remove the activity information from packets leaving the ABC domain.

3.2. Activity Meter

Ingress nodes run an activity meter per monitored traffic aggregate. The activity meter measures a time-dependent traffic rate R_m over a short time scale M_{AM} which is called memory. We chose the

TDRM-UTEMA method [24] for this purpose. The meter is configured with a reference rate R_r and computes the activity of a packet by

$$A = \log_2 \left(\frac{R_m}{R_r} \right). \quad (1)$$

The activity is written into the header of the packet before passing it to the ABC domain.

3.3. Activity AQM

An activity AQM takes acceptance decisions for packets. If the current queue size Q exceeds a computed drop threshold T_{drop}, the packet is dropped, otherwise it is forwarded. The drop threshold is computed as follows:

$$T_{drop}(A) = \max \left(Q_{min}, Q_{base} - \gamma \cdot (A - A_{avg}) \right). \quad (2)$$

We explain the components of that formula. Q_{min} prevents packet dropping in the absence of congestion. Q_{base} is a configured baseline value around which packets are dropped. A is the packet's activity and A_{avg} is a moving average of the activity of recently accepted packets. We utilize the UTEMA method [24] with memory M_{AA} for the computation of that average. The drop threshold is proportional to the packet's activity so that the loss probability of a packet increases with its activity. The parameter $\gamma > 0$ allows to tune that effect.

3.4. Discussion

Ingress nodes are configured per controlled aggregate with traffic descriptors, the memory for rate measurement, and a reference rate, and they require measurement state for each aggregate. If the number of traffic aggregates on ingress nodes is moderate, this seems feasible. Forwarding nodes are configured per egress port with parameters for activity averaging and activity AQM, and they require averaging state for each egress port. As the number of egress ports is low, ABC scales well for core nodes.

As packet dropping depends on activity values contained in packet headers, activity meters and forwarding nodes should be trusted devices. Otherwise, malicious users can avoid packet drops by inserting low activity values and obtain unfairly high throughput at the expense of other users.

Reference rates R_r specific to aggregates may be used to differentiate their achievable throughput in case of congestion. Moreover, ABC has been extended to support different delay classes, i.e., aggregate-specific throughput and forwarding delay can be controlled independently of each other. Detailed simulation results backing these claims and recommendations for parameter settings are provided in [6].

We conclude that ABC provides scalable, QoS-aware congestion management for closed networking domains.

4. Data Plane Programmability Using P4

We give an overview of data plane programmability using the programming language P4 [25] and its processing pipeline. P4 allows the definition of new header fields and forwarding behaviour, which makes it attractive for the implementation of novel forwarding paradigms. P4 programs are compiled for so-called targets, i.e., P4-capable switches, and offer a program-specific application programming interface (API) for their configuration. This API serves for either manual configuration or automatic configuration using a controller. Due to the latter, P4 is often leveraged for software-defined networking (SDN).

4.1. P4 Processing Pipeline

A P4 program defines a pipeline for packet processing which is visualized in Figure 2. It is structured into the parser, the ingress pipeline, the egress pipeline, and the deparser.

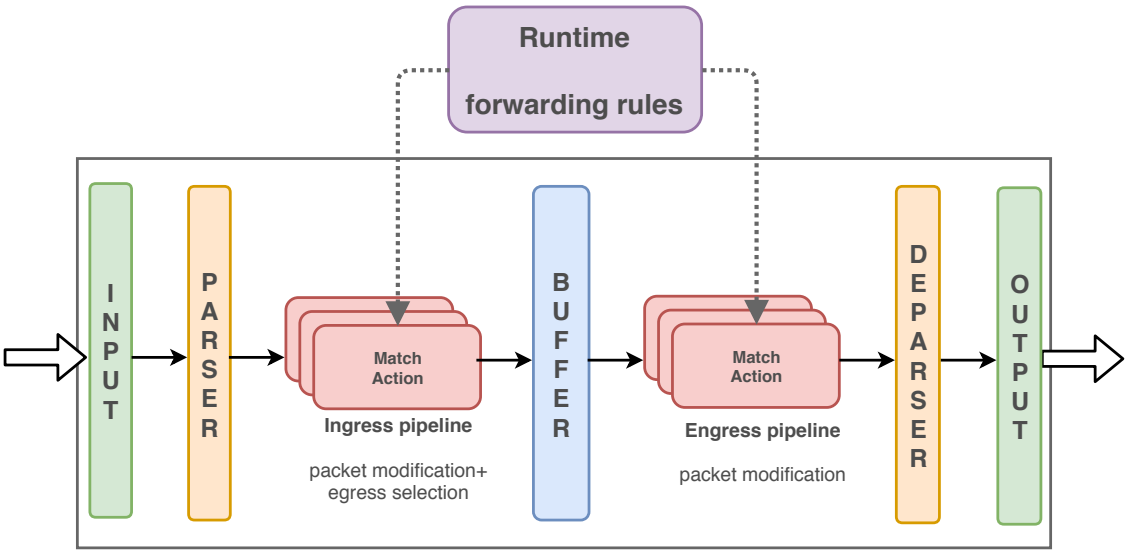

Figure 2. P4 processing pipeline.

The parser reads packet headers and stores their values in header fields. They are carried through the entire pipeline together with packets and standard metadata, e.g., the port on which the packet has been received. In addition, user-defined metadata may store values calculated during processing, e.g., flags required for decisions later in the pipeline. The ingress and egress pipeline may modify header fields, add or remove headers, clone packets, and perform many more actions useful for flexible packet processing. Packets or clones may be even processed several times by the ingress or egress pipeline, but we do not go into details here as we do not leverage these features for ABC. The ingress pipeline typically determines the output port for a packet. After completion of the egress pipeline, packets are deparsed, i.e., their headers are assembled from the possibly modified header fields, and sent.

4.2. Match Action Tables

Within the ingress and egress pipeline, packets are processed by a programmable sequence of match action tables. Their entries are called rules and each rule consists of a match part and a set of actions. Rules are installed, modified, or deleted during runtime through the API. When a packet is processed, its header or metadata fields are compared by the match part of each rule until a matching rule is found. There are different kinds of match types: exact, longest-prefix, and ternary. In case of a match, no further matching within this table is performed, and the actions in the action set of the corresponding rule are executed.

An action set consists of pre-defined primitives like adding or removing a header, reading and writing header or metadata fields, adding or subtracting values, updating counters, or dropping the packet. Custom functions, so-called externs, may be utilized within actions. Examples are encryption or decryption of fields. While the set of supported externs is target-specific, software switches even allow the definition of new externs. It is possible to define multiple match action tables. One action is applying another match action table to a packet. Thereby, packets may be processed by a chain of match action tables. To prevent processing loops, a packet can be processed at most once by any table within a pipeline.

As an example, IP forwarding can be implemented using a longest-prefix match for the destination IP address of an IP packet. In case of a match, actions are called to decrement the TTL field, adapt the IP checksum, and forward the packet to the appropriate egress port.

For the sake of readability we omit technical details about P4 programming, the P4 code, and P4 syntax. For details we refer to the $P4_{16}$ specification [25].

4.3. Variables

For arithmetic operations, P4 supports only signed integers. Therefore, we utilize extern functions for floating point operations and store floating point numbers as fixed-size bit strings. Metadata and header fields carry information throughout the processing pipeline, but they are bound to individual packets. To store information persistently, registers may be used. They can be allocated at program start and accessed and updated in actions of the ingress and egress pipeline. An example for the use of registers is keeping connection state.

5. P4-Based Implementation of ABC

We first discuss some issues that impact the overall design of the prototype. Then, we present the ingress and the egress control flow which define the behavior of the ingress and egress pipeline.

5.1. Design Considerations

We use the software switch BMv2 in the version of 10/15/2018 (unnumbered) as target. As the software switch and the transmission link are only loosely coupled, we do not have access to the buffer occupancy of the link and packets are lost in the link's buffer if the software switch sends too fast. To cope with this problem, we apply the following workaround. The BMv2 has a "packet buffer" between ingress and egress pipeline and an "output buffer" after the egress pipeline. The latter is used only for communication, not for buffering. We limit the packet rate of the egress pipeline to 4170 packets/s, which allows a throughput of 50 Mb/s for packets that are 1500 bytes large. This ensures that the transmission link cannot get overloaded and that a potential queue builds up in the BMv2's packet buffer. With ABC, packets are accepted before being buffered. Therefore, we perform packet acceptance decisions in the ingress pipeline. It requires the queue length for the egress port to which the respective packet is destined. However, this value is accessible only in the egress pipeline. Therefore, the P4 egress pipeline, whenever called to process a packet, copies the egress-port specific queue length to a register which is also accessible in the ingress pipeline. To keep that register value up-to-date, the ingress pipeline increments that register value by one whenever a new packet is accepted for a specific egress port. This design is due to the lack of BMv2 to access the buffer occupancy of the outgoing link. On hardware switches, it is desirable to have access to queue lengths of transmission links so that AQMs can be efficiently implemented.

ABC requires externs for floating point operations to support rate measurement, activity computation, activity averaging and computation of drop threshold. The externs operate on state variables. These variables are interpreted by the externs as floating point numbers but are kept as bit strings within P4. As the state variables are related to traffic aggregates or egress ports, we store their bit string values in registers. As registers cannot be read or written by externs, we copy register values to local variables and pass them to externs when they are called. Likewise, local variables can be passed to externs, modified by them, and copied back to registers after the call.

We wrote externs in C++ and manually added their code to the source code of BMv2. After recompilation, the user-defined externs were available within the P4 program. Adding externs is more difficult for hardware switches and may require vendor support.

In our experiments we study the congestion management performance of ABC on a single bottleneck link. In this particular case, only the transmitting side of the link performs activity metering and runs an activity AQM. Therefore, packets do not need to carry activity information, which removes the need for coding it in packet headers.

P4 programs provide an API for external control. We leverage this API and a script to populate match action tables and to initialize state variables in our experiments. Therefore, a controller is not needed for the prototype.

5.2. Ingress Control Flow

The ingress control flow of the ABC prototype comprises activity metering, determination of the egress port, and acceptance decision through activity AQM. It leverages two match action tables.

The first table holds rules for each aggregate with exact match on IP source address as this defines the aggregates in our experiments. The rules contain configuration parameters for activity metering and register numbers related to state variables. The associated logic calls an extern for activity metering and stores the resulting activity value in user-defined metadata that is carried with the packet. Activity metering is implemented as extern because our rate measurement requires an exponential function and floating point division, and the activity calculation further requires a logarithm (see Equation (1)). The extern leverages the packet's arrival date and size which are available as standard metadata, the configuration parameters passed as table entries, and three state variables for aggregate-specific rate measurement.

The second table determines the egress port and performs the acceptance decision. Determination of the egress port works like IP forwarding described above. Then, the egress port specific queue length is read and an extern is called that performs activity AQM including activity averaging. Besides the egress port, the rules contain configuration parameters for activity averaging and activity AQM, and register numbers related to state variables for the purpose of activity averaging. The extern accepts or denies the packet. In case of acceptance, the register for the egress port specific queue length is incremented and the packet is forwarded to the egress control flow. Otherwise, the packet is dropped.

5.3. Egress Control Flow

The egress control flow sends any received packet. In addition, it copies the egress port specific queue length to the corresponding register.

6. Performance Evaluation

We first present our evaluation methodology and then demonstrate the fairness achieved without and with ABC.

6.1. Evaluation Methodology

We describe the experimental design of our evaluation, the experimental environment, and summarize applied parameters.

6.1.1. Experimental Design

Our study quantifies the goodput achieved by two clients uploading traffic to a server over a joint bottleneck link with 50 Mb/s. Client 0 is mostly a heavy user in our experiments and Client 1 a light user. We utilize the experimental setup depicted in Figure 3. Clients with different IP addresses send traffic over fast access links with 100 Mb/s via a switch to a server behind a slow bottleneck link. Traffic from each client constitutes a traffic aggregate identified by its source IP address. In this specific experiment, only the client side of the bottleneck link requires ABC functionality. It meters the activity of packets coming from the clients and drops them depending on that value. We renounce ABC on the return path because it carries only little traffic, e.g., TCP acknowledgements in some experiment series.

6.1.2. Test Environment

Our testbed is hosted by a virtual machine with Ubuntu 16.04, 4 CPU cores with hyperthreading, 3.5 GHz, and 8 GB RAM. We utilize Mininet version 2.3.0d4 to emulate the mentioned experimental network. Clients, server, and the switch are implemented as virtual machines. We leverage Iperf 3.6 for TCP and UDP traffic generation between client and server and measure the goodput in terms of transport layer payload. One run takes 300 s and 10 runs were carried out per data point.

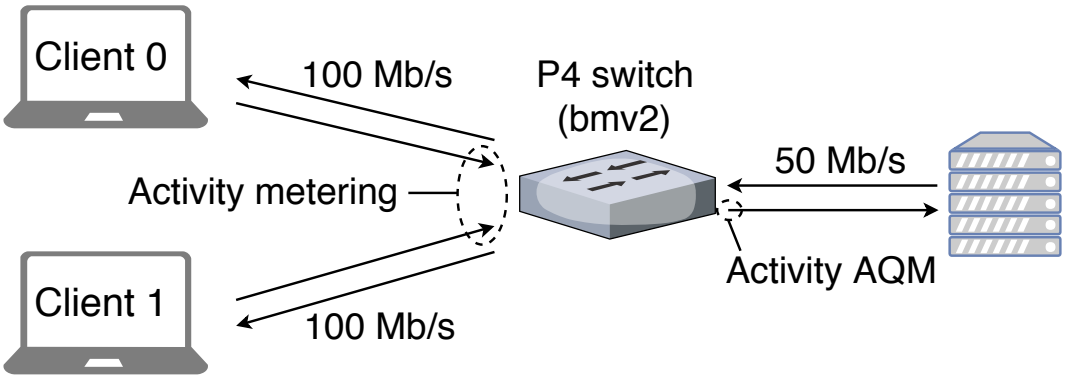

Figure 3. Experimental setup.

6.1.3. Parameters

For the sake of comparability, we choose most experimental parameters as in [6]. The clients are connected to the switch via 100 Mb/s links with a sufficiently large buffer size, and the switch is connected to the server via a 50 Mb/s link with a buffer size of 24 packets. All links are configured with a one-way delay of 5 ms.

Activity meters are configured with a memory of $M_{AM} = 3$ s and a reference rate of $R_r = 10$ kb/s. The activity averager is configured with a memory of $M_{AA} = 0.3$ s and the activity AQM utilizes the parameters $Q_{min} = 6$ packets, $Q_{base} = 20$ packets, and $\gamma = 16$ packets.

The results in this study differ from those in [6] in that we measure goodput instead of throughput, which is due to the Iperf tool, and that we utilize a larger bottleneck bandwidth of 50 Mb/s instead of 10 Mb/s in [6]. We cannot go to larger bottleneck speeds than 50 Mb/s in our experiments as the software switch BMv2 cannot read input traffic fast enough and drops packets at the ingress at higher speeds. Furthermore, we work now with $Q_{min} = 6$ packets instead of $Q_{min} = 12$ packets.

As ABC artificially limits the queue size, the transmission of a single flow may be hampered, which is undesirable. However, we validated that with the chosen parameter set; the bottleneck link can be fully utilized by a single TCP flow.

6.2. Experimental Results

We evaluate the bandwidth sharing performance without and with ABC in three different cases that were investigated by simulation in [6].

6.2.1. Resource Sharing with CBR Traffic

In a first experiment series, both clients send constant bit rate (CBR) traffic using UDP packets with 1448 bytes payload. Client 0 transmits at different rates R_0 while Client 1 sends at $R_1 = 40$ Mb/s. Figure 4 shows the obtained goodput for both clients.

Without ABC, the goodput of Client 0 continuously increases with increasing traffic rate R_0 while the goodput of Client 1 decreases. In particular, the goodput is proportional to the sent traffic rate of both clients. Thus, unfair sending behaviour is rewarded with higher goodput. This is different with ABC. The goodput of Client 0 continuously increases with increasing traffic rate R_0 as long as Client 0 sends less traffic than Client 1. As a consequence, the goodput of Client 1 continuously decreases. If Client 0 sends more traffic than Client 1, the traffic of Client 0 is preferentially dropped due to increased activity so that the goodput of Client 0 becomes clearly smaller than the goodput of Client 1. Thus, ABC creates an ecosystem in which senders benefit from decreased activity in case of congestion instead of being rewarded for sending at high rate. This incentivizes the use of congestion-controlled transport protocols.

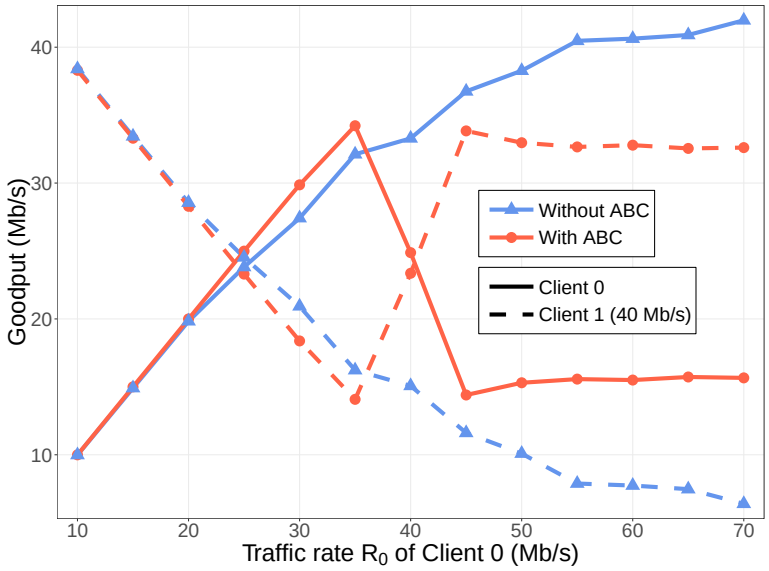

Figure 4. Resource sharing with constant bit rate (CBR) traffic; Client 0 sends CBR traffic as indicated on the *x*-axis and Client 1 sends CBR traffic at 40 Mb/s.

6.2.2. Resource Sharing with TCP Traffic

In a second experiment series, both clients send TCP traffic. We vary the number of saturated TCP connections of Client 0 while Client 1 has only a single saturated TCP connection. Thus, Client 0 is a heavy user while Client 1 is a light user. Figure 5 shows the obtained goodput for both clients.

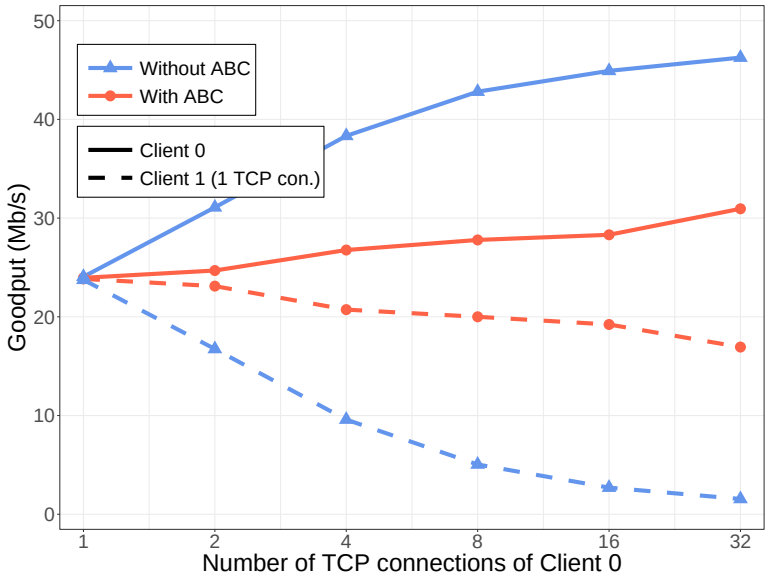

Figure 5. Resource sharing with TCP traffic; the number of saturated TCP connections of Client 0 is indicated on the *x*-axis while Client 1 has only one saturated TCP connection.

Without ABC, the goodput of Client 0 increases with increasing number of TCP connections while the goodput of Client 1 significantly decreases. With ABC, the goodput of both clients remains in the order of 25 Mb/s if the number of TCP connections for Client 0 increases. Thus, fair resource sharing is approximated.

6.2.3. Resource Sharing with CBR and TCP Traffic

In the third experiment series, Client 0 sends CBR traffic at different rates R_0 while Client 1 has a single saturated TCP connection. Figure 6 shows the obtained goodput for both clients.

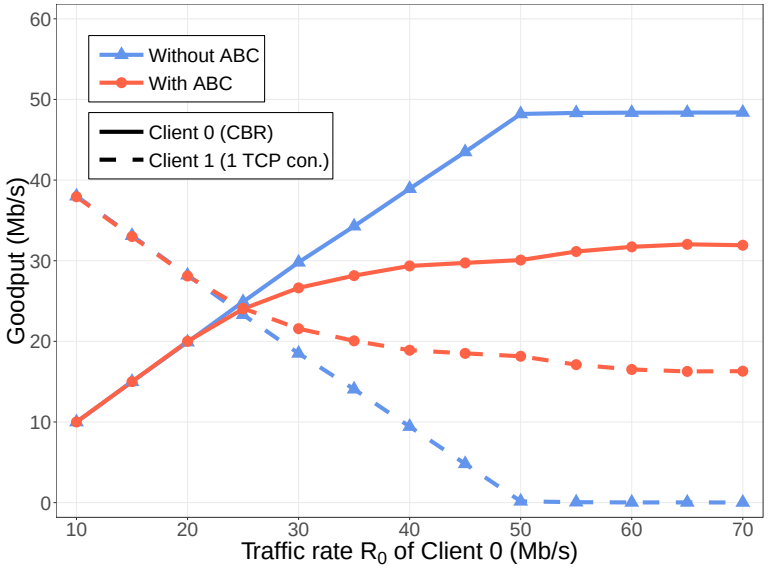

Figure 6. Resource sharing with CBR and TCP traffic; Client 0 sends CBR traffic as indicated on the *x*-axis while Client 1 has one saturated TCP connection.

Without ABC, the goodput of Client 0 increases with increasing transmission rate while the goodput of Client 1 decreases because it can only use the bandwidth left over by Client 0. If the transmission rate of Client 0 exceeds the capacity of the bottleneck link, Client 1 achieves hardly any goodput. This is different with ABC. Client 0 can increase its goodput only up to 32 Mb/s by increasing its transmission rate to very large values, but the remaining capacity is utilized by Client 1. Thus, ABC approximates fair resource sharing even under challenging conditions.

7. Conclusions

We reviewed activity-based congestion management (ABC) for fair resource sharing, gave an introduction to P4, and demonstrated the technical feasibility of ABC on programmable software switches by a prototype implementation in P4. We presented performance results illustrating the ability of ABC to enforce fairness.

The implementation leveraged several extern functions that can be utilized on a software switch but may not be available on hardware switches. Thus, P4-capable hardware switches should provide a wider range of externs to support richer use cases. Moreover, access to queue lengths of transmission interfaces are desirable to support implementation of simple AQMs.

Experimental results with the prototype implementation illustrated that ABC provides an ecosystem where users can maximize their throughput by sending at their fair share in case of congestion, which incentivizes the use of congestion controlled transport protocols. Moreover, the experimental results are in line with a more comprehensive simulation study [6].

As ABC does not require per-aggregate states in core nodes, it is a scalable technology for core networks. As it requires trusted network devices, it should be applied only in closed networks. Extensions for QoS differentiation exist. Thus, ABC provides scalable, QoS-aware congestion management for closed networking domains. Therefore, it may be an attractive technology for 5G transport networks, data center networks, or residential access networks of ISPs.

Author Contributions: Conceptualization, M.M.; Investigation, H.M.; Project administration, M.M.; Software, H.M. and M.H.; Supervision, M.M.; Validation, D.M.; Visualization, D.M. and M.H.; Writing—original draft, H.M. and D.M.; Writing—review and editing, M.M and H.M and D.M.

Funding: This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant ME2727/2-1.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jaber, M.; Imran, M.A.; Tafazolli, R.; Tukmanov, A. 5G backhaul challenges and emerging research directions: A survey. *IEEE Access* **2016**, *4*, 1743–1766. [CrossRef]
2. Kutscher, D.; Mir, F.; Winter, R.; Krishnan, S.; Zhang, Y.; Bernados, C.J. Mobile Communication Congestion Exposure Scenario. 2015. Available online: http://tools.ietf.org/html/draft-ietf-conex-mobile (accessed on 1 July 2019).
3. Briscoe, B. Initial Congestion Exposure (ConEx) Deployment Examples. 2012. Available online: http://tools.ietf.org/html/draft-briscoe-conex-initial-deploy (accessed on 1 July 2019).
4. Briscoe, B.; Sridharan, M. Network Performance Isolation in Data Centres using Congestion Policing. 2014. Available online: http://tools.ietf.org/html/draft-briscoe-conex-data-centre (accessed on 1 July 2019).
5. Menth, M.; Zeitler, N. Activity-based congestion management for fair bandwidth sharing in trusted packet networks. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), Istanbul, Turkey, 23–25 April 2016.
6. Menth, M.; Zeitler, N. Fair resource sharing for stateless-core packet-switched networks with prioritization. *IEEE Access* **2018**, *6*, 42702–42720. [CrossRef]
7. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
8. Broadband Internet Technical Advisory Group (BITAG). *Real-Time Network Management of Internet Congestion*; Technical report; Broadband Internet Technical Advisory Group (BITAG): Denver, CO, USA, 2013.
9. Stoica, I.; Shenker, S.; Zhang, H. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Trans. Netw.* **2003**, *11*, 33–46. [CrossRef]
10. Cao, Z.; Zegura, E.; Wang, Z. Rainbow fair queueing: Theory and applications. *Comput. Netw.* **2005**, *47*, 367–392. [CrossRef]
11. Nádas, S.; Turányi, Z.R.; Rácz, S. Per packet value: A practical concept for network resource haring. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016.
12. Laki, S.; Gombos, G.; Hudoba, P.; Nádas, S.; Kiss, Z.; Pongrácz, G.; Keszei, C. Scalable per subscriber QoS with core-stateless scheduling. *ACM SIGCOMM Demo* **2018**, *1*, 84–86.
13. Nádas, S.; Gombos, G.; Fejes, F.; Laki, S. Towards core-stateless fairness on multiple timescales. In Proceedings of the ACM/IRTF/ISOC Applied Networking Research Workshop (ANRW), Montreal, QC, Canada, 22 July 2019.
14. Nádas, S.; Turányi, Z.; Gombos, G.; Laki, S. Stateless resource sharing in networks with multi-layer virtualization. In Proceedings of the IEEE International Conference on Communications (ICC), Shangai, China, 20–24 May 2019.
15. Cascone, C.; Bonelli, N.; Bianchi, L.; Capone, A.; Sanso, B. Towards approximate fair bandwidth sharing via dynamic priority queuing. In Proceedings of the IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), Osaka, Japan, 12–14 June 2017.
16. Sharma, N.K.; Liu, M.; Atreya, K.; Krishnamurthy, A. Approximating fair queueing on reconfigurable switches. In Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI), Renton, WA, USA, 9–11 April 2018; pp. 1–16.
17. Nichols, K.; Jacobson, V. Controlling queue delay. *ACM Queue* **2012**, *10*, 1–15. [CrossRef]
18. Kundel, R.; Blendin, J.; Viernickel, T.; Koldehofe, B.; Steinmetz, R. P4-CoDel: Active queue management in programmable data planes. In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 27–29 November 2018.

19. Turkovic, B.; Kuipers, F.; van Adrichem, N.; Langendoen, K. Fast network congestion detection and avoidance using P4. In Proceedings of the ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies (NEAT), Budapest, Hungary, 20 August 2018.
20. Joshi, R.; Qu, T.; Chan, M.C.; Leong, B.; Loo, B.T. BurstRadar: Practical real-time microburst monitoring for datacenter networks. In Proceedings of the 9th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys), Jeju Island, Korea, 27–28 August 2018.
21. Geng, J.; Yan, J.; Ren, Y.; Zhang, Y. Design and implementation of network monitoring and scheduling architecture based on P4. In Proceedings of the 2nd International Conference on Computer Science and Application Engineering, Hohhot, China, 22–24 October 2018.
22. Hill, J.; Aloserij, M.; Grosso, P. Tracking network flows with P4. In Proceedings of the IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), Dallas, TX, USA, 11 November 2018.
23. da Silva, M.V.B.; Jacobs, A.S.; Pfitscher, R.J.; Granville, L.Z. IDEAFIX: Identifying elephant flows in P4-based IXP networks. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Abu Dahbi, UAE, 9–13 December 2018.
24. Menth, M.; Hauser, F. On moving averages, histograms and time-dependent rates for online measurement. In Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE), L'Aquila, Italy, 22–27 April 2017.
25. The P4 Language Consortium. $P4_{16}$ Language Specification. Available online: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf (accessed on 1 June 2018).