A Novel Active Queue Management Scheme for Programmable Data Planes using P4

Meer M. Khan
Department of Electrical and Computer Engineering
University of Campinas
Campinas, Brazil
meer.khan78@gmail.com

***Abstract*—For supporting different applications requirements the network programmability brings flexibility, agility and rapid introduction of new capabilities. Significant benefits like Quality of Service QoS can be achieved by using programmable traffic management at data plane. To achieve the QoS in Term of minimum queuing delay we used the P4 which is a domain specific language for programming network switches. we implement our novel active queue management (NAQM) scheme by manipulating the bits at data plane and we can see in results that our proposed scheme is better in term of less less queuing delay.**

***Index Terms*—Quality of Service, Data plane, P4, Active Queue management.**

I. Introduction

Centralization of the network's intelligence in Software Defined Networking (SDN) is an advantage for applications that do not have strict real-time requirements and depend on global network state. However, when the service uses local state information, e.g., to support QoS, the same level of flexibility must be supported at the data plane. Making stateful data plane algorithms programmable, complementing existing programmable forwarding plane solutions, can be beneficial in terms of meeting QoS requirements, reduce the control load on the SDN controller(s) and network overhead, and enhance network flexibility by enabling customized traffic management. Programmable data plane solutions such as P4 and supported architectures, provide an excellent way to define the packet forwarding behavior of network devices. However, most programmable devices still typically have non-programmable traffic managers. The P4 open source community has recently started working towards defining a P4 programmable traffic manager, encompassing functionality such as packet scheduling, shaping, policing etc. However, traffic management is still not programmable for P4 targets. Towards providing fully programmable and customized data planes, we present our initial effort to complement the suite of NAQM schemes using the P4 domain specific language for programming network switches (such as in [1]). Our approach builds on the premise that an SDN data path is a very fast and efficient engine to perform low level primitive operations at wire speed. We implement a modern NAQM algorithm.

II. Background

P4 is a declarative language for programming protocol-independent packet processors, with constructs optimized for packet forwarding functions. Using P4, developers can program data plane packet pipelines, on a variety of targets (e.g., ASICs, FPGAs etc.) based on a match/action architecture. While the P4 language is target independent, a P4 compiler translates P4 programs into the instruction set of the hardware of the packet processor. The current specification of the language (P4-16), introduced the concept of the P4 architecture that defines the P4-programmable blocks of a target and their data plane interfaces. The standard P4-16 architecture called Portable Switch Architecture (PSA) was published in November 2018 [2]. The abstract forwarding model of any P4 device starts with a programmable parser, which allows custom defined headers. Next multiple match+action stages, which can be in parallel or in series, may modify headers or determine egress ports. The P414 forwarding model has a pipeline, divided into an ingress and egress part. Between those parts, a fixed function block for queuing packets is implemented. We use the open source software switch BMV2, since it constitutes the reference software implementation of the PSA. However, at the moment only a partial implementation is available. Therefore we adopt the v1model, a description written in P4-16 of the old fixed architecture implied by P4-14, as it is fully supported by BMV2 [3].

III. Related Work

Sharma et.al. have used the possibilities of P4 programmable data planes to establish fairness between multiple-flows by per flow metering and a approximating fair queueing algorithm [4]. A very similar fairness problem was addressed by another work which implemented a equity motivated queueing algorithm for P4 data planes [5]. Sivaraman et.al. analyzed different AQM algorithm and found out that there is “No Silver Bullet” which fits best for all use cases [6]. They have shown that it is possible to implement CoDel with FPGAs, attached to a fixed function forwarding ASIC. Their key insight was, that the data plane must be flexible in order to be open for different AQM algorithms. From their point of view, that could be achieved best by including small FPGAs on switches.

IV. SYSTEM DESIGN

The BMV2 switch buffers packets between the ingress and egress pipeline using an egress queue per port. Using the v1model architecture, a P4 program can access in the egress pipeline the queuing delay for a packet and related queuing information, as part of the standard packet metadata. Therefore the PI2 NAQM in our case is integrated into the P4 egress pipeline as denoted in Fig. 1. The NAQM can be deployed on any Linux based system without the need of specialized hardware or proprietary software. The implementation serves as a proof of concept for deploying a custom AQM using P4.

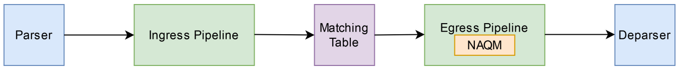


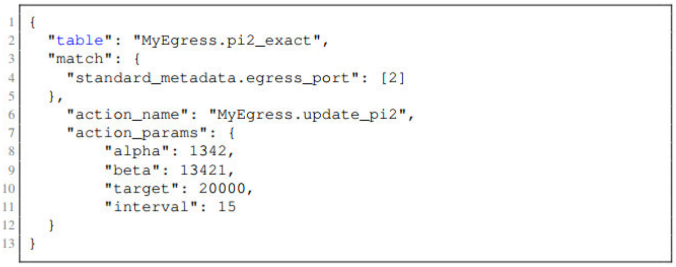
Fig. 1. NAQM in V1 model

A. Ingress Control Flow:

Match-action tables are the mechanism for performing packet processing. The ingress pipeline uses one match action table to implement basic L3-forwarding.

B. Egress Control Flow:

A second match-action table is introduced that applies the NAQM, complemented with the PI2 configuration parameters. The switch upon matching the port number executes the update-PI2 action. Configuration parameters include the queuing delay target in µs, PI2 gain factors (and) and the PI update interval T in exponent µs. A sample static rule used by the control plane to populate the table is provided in fig. 2. The PI gain factors are set to = 0.3125Hz and = 3.125Hz according to the stability analysis for RTTs up to 100 ms and an update time of around 33ms (typically 1/3th of the max RTT). Fixed calculations and conversions on parameters are performed by the controller to avoid repetitive per packet processing. So the actual input values for the P4 program are scaled to fit in a 32-bit integer value. The delay target is set to 20,000 µ. In fig. 3. flowchart for our scheme can be seen which illustrate the working of our scheme and overall architecture of P4 switch.



```
{
  "table": "MyEgress.pi2_exact",
  "match": {
    "standard_metadata.egress_port": [2]
  },
    "action_name": "MyEgress.update_pi2",
    "action_params": {
        "alpha": 1342,
        "beta": 13421,
        "target": 20000,
        "interval": 15
  }
}
```

Fig. 2. PI_exact Rule

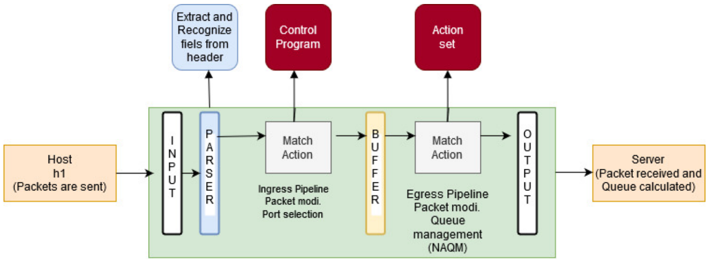

Extract and Recognize fiels from header
Control Program
Action set
Host h1 (Packets are sent)
INPUT
PARSER
Match Action
Ingress Pipeline Packet modi. Port selection
BUFFER
Match Action
Egress Pipeline Packet modi. Queue management (NAQM)
OUTPUT
Server (Packet received and Queue calculated)

Fig. 3. P4 switch architecture and Flowchart

C. Discussion:

The current P4 abstraction model poses several challenges for implementing an NAQM scheme. The absence of floating point types required careful mapping into integer types, while the lack of support for complex arithmetic functions can be tackled with the use of smart bit manipulations or external approximation functions supported by the hardware. Authors in [7] employed the P4 longest prefix match table feature to map approximations for the square root. Our implementation does not require such resources which otherwise would constrain the forwarding functionality.

phase over an interval of 250 sec (average values over 1 second intervals).

V. Performance Evaluation

In the current section we illustrate the functionality of our P4- programmable AQM scheme using some simple test cases.

A. Simulation Environment:

Our simulation setup consists of 3 nodes with at least two 1Gbps interfaces as shown in Fig. 3. The configuration parameters for the PI2 AQM algorithm are provided in Listing 1. We compare PI2 with a existing queue that requires a buffer of 200 packets, assuming a reference link of 20Mbps and a delay target of 20ms. For the PI2 AQM we set the buffer size high enough (10,000 packets) to make sure that the only congestion control is performed by the AQM. The target delay can be modified without impacting other control parameters (we use 5ms and 20ms).

The P4 switch can limit the rate of packets emitted from the egress pipeline, emulating different link capacities. We set the emitting rate of packets upto 300 Mbs for best result. We use this configuration option to set the rate limit. A base RTT is established by setting a 5ms delay on the interfaces of the client and server machines. A single greedy TCP flow is running and the network queuing delay is shown during its steady state phase over an interval of 250 sec (average values over 1 second intervals).

VI. Results

We verified our NAQM implementation using an extensive set of tests to make sure it lines up with the results. However to demonstrate the effectiveness of the implementation, we present in Fig. 4 the queuing delay for the P4-PI2 NAQM,

TABLE I
NETWORK PARAMETER USED IN SIMULATION ANALYSIS

Parameters	Values
Simulation Time	250 Seconds
Noes	3
P4 Switch	1
Server	1
Host	1
Bandwidth	10 Mbs, 30 Mbs
Header Size	20 bytes
MTU	1500
Buffer size	10000
Packet Type	TCP packet

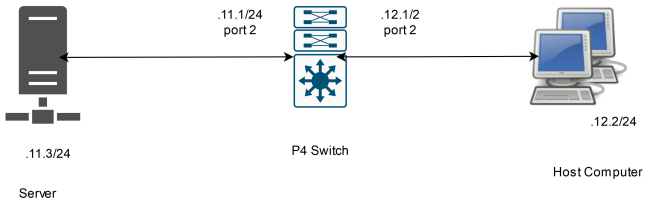


Fig. 4. Simulation Setup

compared to the Existing scheme for varying network conditions. The initial bottleneck link capacity is set to 30 Mbps. Fig. 4.(i) confirms as expected that a TD buffer size of 200 packets limits the delay to 20ms. It also shows that a PI2 AQM can easily be reconfigured by the controller with different delay targets (here 20ms and 5ms) via a simple update in the corresponding entry at the table. we also configured PI2 for 5 ms queuing delay also this the reason we are gaining two different queuing delay for our scheme one for 20 ms and second for 5 ms. The existing taildrop queuing scheme is only giving 20 ms queuing delay in ideal situation when the link capity is maximum.

When the link capacity is reduced to 10Mbps in our simulation environment as in the plot of Fig. 5 the queue will be served much slower, resulting for the TD case in a much higher delay (up to 200ms). The PI2 controller on the other hand, keeps the delay at the target levels. Moreover, there is no need to update the AQM settings when the throughput of the link changes and this is reason our poroposed scheme is performing ideal in this situation and giving same 20 ms and 5 ms delay. The packet drop ratio and packet delivery ratio can be calculated by our scheme. We also introduced new header and registers in our scheme for calculating the number of drop packets and number of delivered packets.

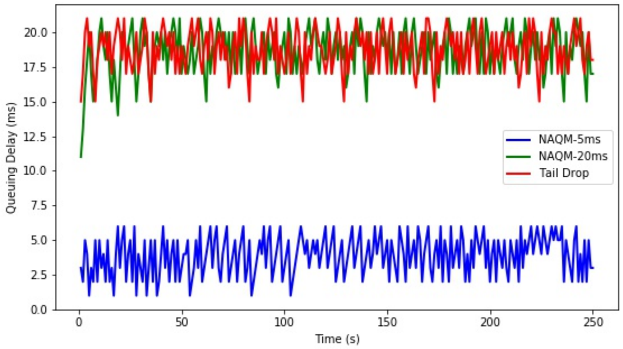


Fig. 5. Queuing Delay with 30 Mbs

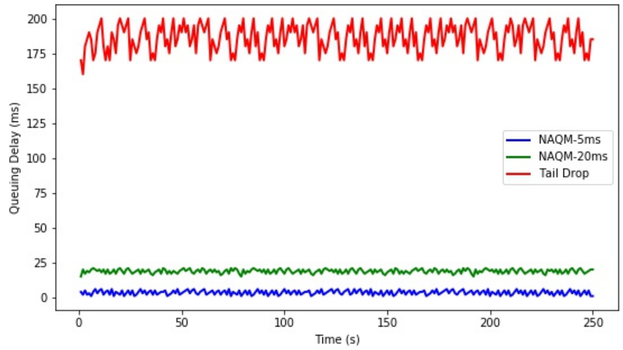


Fig. 6. Queuing Delay with 10 Mbs

VII. CONCLUSION AND FUTURE WORK

In this paper a novel queuing delay scheme is proposed and with the help experiments it can be seen in graph that our proposed scheme give the better results in term of queuing delay as compared to existing scheme in various network condition. All relevant code and information available on Github repository to reproduce the the work (**https://github.com/Meerkhan12/Reproducible-research**). Due to its isolation features, network slicing is a great opportunity to further customize per-slice network behavior. Next to NAQMs, we believe additional per-slice protocol customization and real-time network-service interactions are required enablers to deal with future network challenges. This will require an agile and flexible programmable data-plane beyond the current forwarding and traffic management scope.

REFERENCES

[1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. SIGCOMM Comput. Commun. Rev. 44, 3 (July 2014), 87–95.

[2] Antonin Bas. [n. d.]. The P4 behavioral model version 2. https://github.com/ p4lang/behavioral-model [Online; Retrieved 2019]

[3] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. 2016. PI 2: A Linearized AQM for both Classic and Scalable TCP. In Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. ACM, 105–119.

[4] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in USENIX Symposium on Networked Systems Design and Implementation, 2018.

[5] C. Cascone, N. Bonelli, L. Bianchi, A. Capone, and B. Sansò, "Towards approximate fair bandwidth sharing via dynamic priority queuing," in Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on. IEEE, 2017, pp. 1–6.

[6] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "Nosilver bullet: extending sdn to the data plane," in Proceedings of the Twelfth ACM Workshop on Hot Topics in networks. ACM, 2013, p. 19.

[7] Ralf Kundel, Jeremias Blendin, Tobias Viernickel, Boris Koldehofe, and Ralf Steinmetz. 2018. P4-CoDel: Active Queue Management in Programmable Data Planes.In Proceedings of the IEEE 2018 Conference on Network Functions Virtualization and Sofwtare Defined Networks. IEEE, 27–29.

[8] Dimitris J Bertsimas and Leslie David Servi. 1992. Deducing queueing fromtransactional data: the queue inference engine, revisited.Operations Research40,3-supplement-2 (1992), S217–S228.

[9] F. Schwarzkopf, S. Veith, and M. Menth, "Performance analysis of codel and pie for saturated tcp sources," in Teletraffic Congress (ITC 28), 2016 28th International, vol. 1. IEEE, 2016, pp. 175–183.

[10] Vladimir Gurevich for Barefoot Networks. 2017. Programmable Data Plane atTerabit Speeds. https://p4.org/assets/p4-d2-2017-programmable-data-plane-at-terabit-speeds.pdf . (May 2017).

[11] B. Vinnakota, "Netronome and p4: A brief history and a roadmap," https://www.netronome.com/blog/ netronome-and-p4-a-brief-history-and-a-roadmap/, accessed: 2018-07-.

[12] K. Nichols, V. Jacobson, A. McGregor, and A. Iyengar, "Controlled Delay Active Queue Management," Internet Engineering Task Force, Request for Comments 8289, 2018.