Take your own share of the PIE

Sándor Laki, Gergő Gombos
Eötvös Loránd University
Budapest, Hungary
{lakis,ggombos}@inf.elte.hu

Szilveszter Nádas, Zoltán Turányi
Ericsson Research
Budapest, Hungary
{szilveszter.nadas,zoltan.turanyi}@ericsson.com

ABSTRACT

In this paper, we propose the PVPIE Active Queue Management (AQM) method that combines the packet scheduling and dropping algorithms of PIE AQM and the packet marking-based resource sharing of the Per Packet Value (PPV) concept. The algorithm calculates dropping probabilities needed for keeping the queueing delay at a predefined level using the PIE algorithm. Then instead of applying this drop probability directly on incoming packets it translates the dropping probability to a Congestion Threshold Value (CTV) filter and drops (or marks) all incoming packets with Packet Value smaller than the threshold. The translation is based on statistics collected about Packet Values of incoming packets. Our evaluation based on simulations shows that PVPIE AQM combines the benefits of PIE and PPV concepts, keeping a target queueing delay and implementing policy-based resource sharing at the same time. The motivation for the proposed algorithm is simplicity and ease of deployment in real networks, since the schedulers of the original PPV approach need to perform drops from the middle of the queue. Such drops may be costly and may not be supported by current hardware.

CCS CONCEPTS

• **Networks → Packet scheduling**;

KEYWORDS

PIE; PPV; AQM; resource sharing

ACM Reference format:
Sándor Laki, Gergő Gombos and Szilveszter Nádas, Zoltán Turányi. 2017. Take your own share of the PIE. In *Proceedings of ANRW '17, Prague, Czech Republic, July 15, 2017,* 6 pages.
DOI: 10.1145/3106328.3106333

1 INTRODUCTION

Quality of Service (QoS) and in particular, resource sharing are important parts of the network research area [11], and QoS is still listed as a key issue for 5G standardization [1]. An ideal resource sharing technique has to be lightweight, allow rich resource sharing policies. It has to be simple to implement and it must be able to fulfill predefined delay requirements. In the past ten years, various packet marking-based resource sharing approaches have been proposed from Core Stateless Fair Queueing [12] and Rainbow Fair Queueing [2] to Per Packet Value [8], sharing the idea that the packets are labeled at edge nodes of the network by applying predefined operator policies and then the resource nodes inside the network can solely schedule or drop packets according to the packet labels. These approaches provide lightweight solutions to control bandwidth sharing among flows even when per flow queuing is not possible. Considering the PPV concept, the practical schedulers described so far require dropping from the middle of the queue (drop smallest Packet Value (PV) first). This might be impractical in some network nodes, e.g., when the hardware design only supports drops upon packet arrival and not later.

In the past decade, Active Queue Management (AQM) schemes such as RED [4], PIE [10], CoDel [9] and PI^2 [3] have been developed. PIE (Proportional Integral controller Enhanced) [10], in particular, proposes a lightweight controller-based scheme to adjust the packet drop probability needed for keeping the queuing delay at a target level. Incoming packets are then dropped or ECN (Explicit Congestion Notification) marked with this probability. PIE's feedback loop was developed for classical TCP flows like NewReno or CUBIC [5] and thus a flow with different congestion control behavior might get an unfair share or can even dominate all other flows. In addition, it is not possible to define rich resource sharing policies among flows as the PPV framework allows, because the drop probability is the same for all incoming packets.

In this paper, we extend the packet marking-based bandwidth sharing control of PPV with a lightweight proactive mechanism similar to PIE [10] that reacts to congestion situations earlier than when buffers are filled up and ensures the bandwidth share defined by the marking policy between flows. The proposed AQM method called Packet Value-aware PIE (PVPIE) works in two stages: 1) the algorithm of PIE is applied to determine the expected drop probability needed to keep a target buffering delay; 2) the drop probability is translated to a Packet Value threshold that is then used to filter out incoming packets with Packet Value less than this threshold. The goal of PVPIE AQM is to keep the queuing delay at the target level and in parallel ensure the predefined resource sharing policies at any congestion level.

The remaining part of the paper is organized as follows: In Section 2 we briefly overview how the PPV concept solves the resource sharing problem. Section 3 describes our PVPIE AQM scheme in more detail. The proposed solution is examined and proved by simulations in Section 4. In Section 5 we summarize our results and identify future research directions.

ANRW '17, Prague, Czech Republic

978-1-4503-5108-9/17/07…$15.00
DOI: 10.1145/3106328.3106333

2 PER PACKET VALUE-BASED RESOURCE SHARING

Different applications may have different throughput or delay requirements and thus sharing resources between flows at shared bottlenecks in the network has an utmost importance for ensuring high QoS and eventually good QoE. As mentioned previously, a good QoS solution should support a wide range of flexible resource sharing policies and in parallel its implementation should be lightweight.

The PPV concept [8] extends the idea of packet marking-based resource sharing solutions like [2, 12] by marking each packet with a continuous value called Packet Value (PV) representing the gain of the network operator when the packet is delivered. Since the Packet Value is continuous, arithmetic operations between PVs are also possible (e.g. the gain can be divided or reduced by the transmission cost at a radio gateway node, etc.). The network aims at maximizing the total profit of the operator by maximizing the total aggregate PV delivered.

The system model of PPV is split into two phases: 1) Packet marking at network edge; 2) Packet scheduling and dropping based on the Packet Value at resource nodes in the middle of the network.

First, packets are marked at the edge of the network by using the resource sharing policy of the operator. Note that marking may require flow-level, application-level or even user-specific information to determine the policy to be applied. Packets of the same flow are marked by Packet Values following a PV distribution reflecting the applied operator policy. Operator policies are described by Throughput-Value Functions (TVFs) (marked by $V(.)$), that basically defines the PV distribution of a flow for any sending rate. Specifically, for any throughput value b, the traffic up to b shall receive a PV of $V(b)$ or higher. Accordingly, at high congestion only packets with high values are transmitted, more precisely packets with values above a given Congestion Threshold Value (CTV). Note that the amount of high and low value packets determines the resource share between various flows, resulting in that at high congestion, flows with larger share of high Packet Values receive more throughput. It can also be observed that the larger the space of Packet Values is, the finer the granularity of resource sharing policies can be reached.

Figure 1 depicts example TVFs used in our simulation scenarios presented in Section 4. They describe conditional weighted resource sharing between three classes: Gold, Silver and Background, representing premium, normal and low priority access to resources. The intersection of the TVFs with horizontal lines representing different congestion levels (i.e. CTVs) defines the desired throughput of the classes at the given congestion level. Until Background flows reach 100 kbps, Silver flows get 4 times the throughput of Background ones (I.). Similarly, until Silver flows reach 1 Mbps, Gold flows get twice the throughput of Silver ones (I.). Until the throughput of the Gold flows can reach 4 Mbps, Silver flows are limited to 1 Mbps and Background flows to 100 kbps (II). Above that Silver flows get 10 times the throughput of Background ones and Gold flows get 4 times the throughput of Silver ones (III).

Second, resource nodes in the middle of the network schedule and drop packets without maintaining flow-states, solely relying on the carried Packet Values. Each such node aims at maximizing the

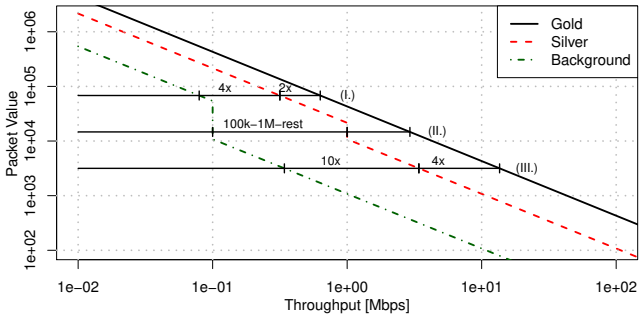

Figure 1: TVF examples (log-log scale).

total amount of values transmitted over the shared bottleneck. To this end, the authors of [8] propose a simple scheduling algorithm that relies on a modified FIFO queue where dropping from the middle of the buffer is also enabled. Accordingly, each time when a packet arrives at a full queue, the algorithm first checks if its PV is less than the minimum PV in the buffer. If this is the case, the incoming packet is the less important and thus it is dropped, otherwise packets with the lowest PV are dropped from the queue to make sufficient space to enqueue the incoming packet. This simple mechanism ensures that the node always drops the least important packet. However, in practice dropping from the middle of the buffer may be costly and are not supported by current chipsets.

Note that in a stationary situation the most value can be delivered over a shared bottleneck by transmitting all packets above the Congestion Threshold Value and dropping the ones below. Assuming that CTV is determined accurately, transmitting any packets below the threshold can only be done at the expanse of a packet above, decreasing the total value delivered. As congestion increases the CTV also rises, while if the load is decreasing, the threshold is also reduced, enabling less important packets to be transmitted. Note that the CTV is generally not stationary, reflecting multiple factors from the available capacity and the amount of offered traffic to the observed Packet Value distribution.

3 PACKET VALUE-AWARE PIE (PVPIE) AQM

The proposed Packet Value-aware PIE (PVPIE) AQM method combines the resource sharing concept of PPV method and the AQM scheme of PIE. It first applies the mechanism of the original PIE AQM for controlling the current drop probability (p). However, instead of dropping (or marking with an ECN flag) the incoming packets uniformly at random with probability p, it sets the congestion threshold value V as a packet filter to both ensure the expected drop rate and take into account that the importance of packets may be different. Accordingly, dropping packets with high PVs is less likely than the ones with small PVs.

A general overview of PVPIE AQM method is depicted in Figure 2. The algorithm first calculates the expected drop probability needed for ensuring the target queueing delay (τ), using a PI (Proportional-Integral) controller like in the original PIE method [10]. The controller has two parameters α for the proportional and β for the integral term and the dropping probability at time

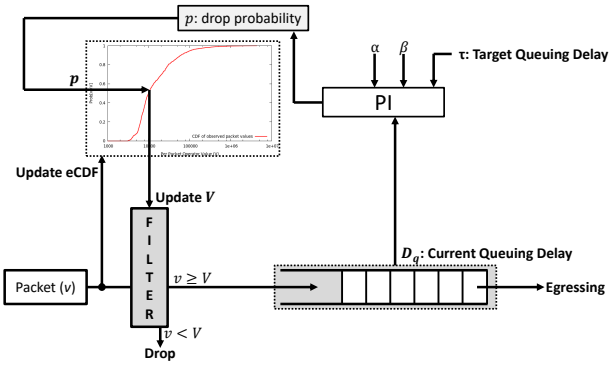

Figure 2: An overview of the PVPIE AQM scheme

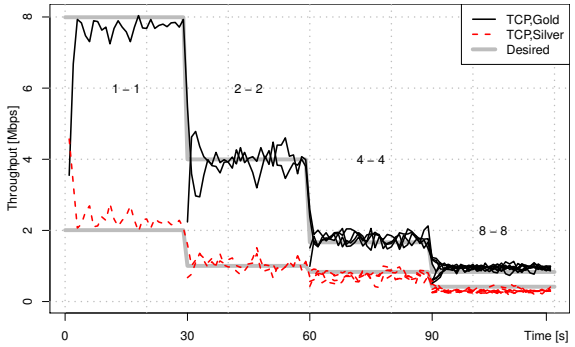

Figure 3: Scenario 1a, throughput.

t is calculated from the current and the target queuing delays as follows:

$$p(t) = p(t-T) + \alpha(D_q(t) - \tau) + \beta(D_q(t) - D_q(t-T)), \quad (1)$$

where T denotes the time elapsed between two updates of the probability value, $D_q(t)$ is the actual queueing delay while $D_q(t-T)$ represents the queueing delay at the time of previous update. Note that the formula is the same as in [10]. After p is determined, the Packet Value threshold V at time t is calculated from the observed PV distribution of incoming packets, according to the following formula:

$$V(t) = eCDF^{-1}_{[t-\gamma T, t)}(p(t)), \quad (2)$$

where $eCDF^{-1}_{[t-\gamma T, t)}(.)$ is the percent-point function (also known as the inverse cumulative distribution function) derived from the Packet Value distribution observed during the time window $[t-\gamma T, t)$ where parameter $\gamma \geq 1$ expresses the length of time window in period time T. Note that $[t-\gamma T, t)$ is referred to as ECDF Window in the following part of the paper. Practically, the eCDF is continuously updated during run-time, reflecting an estimated Packet Value distribution at any time. When the number of packets received in the time window is smaller than $1/p$, the filter (V) is set to 0, thus all packets pass. The controller updates the filter value periodically in every T ms. Every incoming packet with packet value v less than the threshold V is dropped or marked with an ECN flag. If a packet is already buffered, it cannot be dropped by the resource node anymore and is served according to FIFO scheduling. Note that the PI controller used in the first step for determining the drop probability requires parameter-tuning. To find the appropriate values of α and β, the methodology used by PIE [10] can be applied. We also note that the adaptive algorithm of PIE for setting α and β parameters has not been used in this paper.

4 EVALUATION

The PVPIE AQM scheme described in Section 3 has been implemented and evaluated in the NS-3.25 simulator [6]. The algorithm parameters are taken from [10]: $\alpha = 0.125$, $\beta = 1.25$ and the update period is chosen as $T = 32$ *ms*. In all the examined simulation scenarios, we consider a single downlink bottleneck where the same bottleneck buffer is shared by a number of *flows*. We also assume that there is no uplink bottleneck. The term *flow* refers to a piece of traffic to which an operator policy expressed by a specific TVF is applied. Note that according to this flexible definition a wide range of practical traffic aggregates are possible: e.g., a *flow* can refer to all traffic of a given user; it can represent the traffic generated by a single application; or it can be a single TCP or UDP connection. During the simulations, both TCP and UDP traffic are applied. For TCP, TCP CUBIC [5] implementation of NS-3 NSC [7] is used to to generate continuous downloads. UDP flows are not congestion controlled and their constant sending rate is set to 60% of the bottleneck capacity. In the scenarios, each TCP *flow* represents the traffic of an individual user, consisting of 1 or 5 TCP connections/flow, denoted by $N_{tcp.c}$. Markers apply one of the TVFs depicted on Figure 1, that is, Gold, Silver or Background representing the operator policy applied to the flow. We implemented the hierarchical token bucket marker described in [8], which encodes PV logarithmically to an 8-bit field using the formula $30 \cdot \log_{10}(V(b))$. In most cases, a moderate round-trip propagation delay (40 ms) is used and the delay target is set accordingly ($\tau = 40$ *ms*). Parameters used in the different simulator scenarios are summarized in Table 1.

Scenario 1: Varying number of Gold and Silver TCP flows.

In this scenario we increase the number of TCP flows over a fixed bottleneck. In the beginning there are 1 Gold and 1 Silver TCP flows and their number is doubled every 30 s. We plotted the throughput of the flows over a 10 Mbps bottleneck on Figure 3. It is the reproduction of Figure 4 in [8], but using the PVPIE algorithm and the results are comparable to the reference. Figures 4 and 7 show the throughput of the flows over a 100 Mbps bottleneck for $N_{tcp.c} = 1$ and 5 respectively. It can be seen that for all simulations the actual bandwidth shares approach the desired shares accurately. For the 100 Mbps, $N_{tcp.c} = 1$ case (Figure 4), the fluctuations are due to the limited control on high speed TCP flows by dropping, as packet drop events in this case are rare and cannot completely enforce the desired bandwidth share. It is visible that already for the $N_{tcp.c} = 5$ case (Figure 7) it is improved.

We detail the internals of the algorithm for the 100 Mbps cases. Figures 5 and 8 show the Packet Value filter determined by the PVPIE algorithm. We indicate the ideal filter value as "Desired". We observed that in the $N_{tcp.c} = 5$ case, the actual filter is much closer to the ideal than in the $N_{tcp.c} = 1$ case. This is due to the smaller effect of a single drop event on the total flow throughput. It is visible that in the 1-1 and 2-2 flow cases the filter is rarely set, especially in

Scenario	1a	1b	1c	2	3a	3b	4a	4b
Bottleneck [Mbps]	10	100		50	10, 50, 100, 50, 10^{30}		10	
Number of TCP flows (Gold-Silver)	1-1, 2-2, 4-4, 8-8^{30}			0-0, 1-1, 2-2, 4-4^{30}	1-1	10-10	5-0	
Number of UDP flows	0			3 (Background)	0		2 (Silver)	
Number of TCP connections/flow ($N_{tcp.c}$)	1		5	1	5	1	1	5
Target Delay [ms] (τ)	40						20	
round-trip propagation delay [ms]	40						100	
ECDF window (γ)	$1 \cdot T$						$1 \cdot T$	$10 \cdot T$
Figures	3	4,5,6	7,8,9	10	11	12	13	14,15,16

Table 1: Parameters of the evaluated simulation scenarios

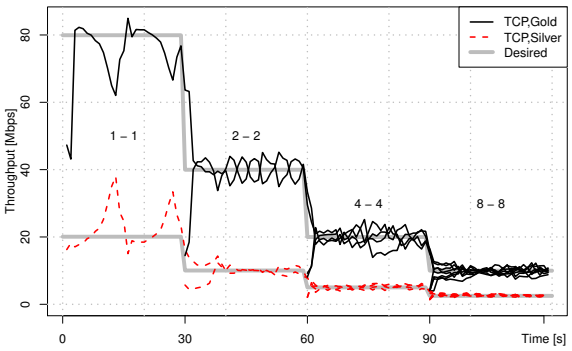

Figure 4: Scenario 1b, Flow throughput

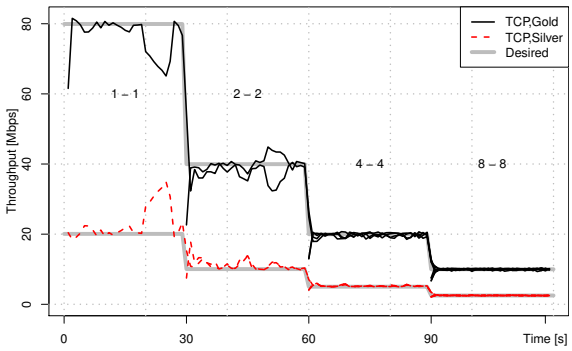

Figure 7: Scenario 1c, Flow throughput

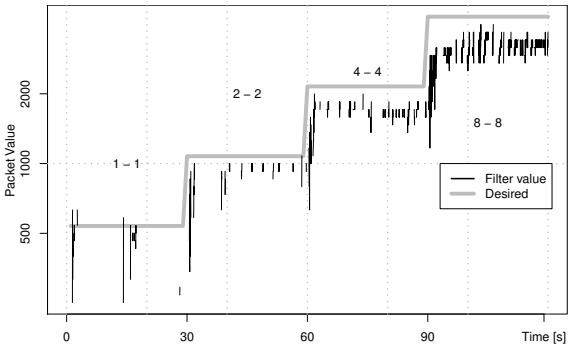

Figure 5: Scenario 1b, Filter threshold

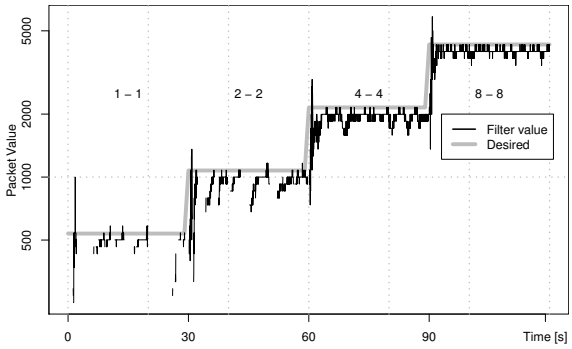

Figure 8: Scenario 1c, Filter threshold

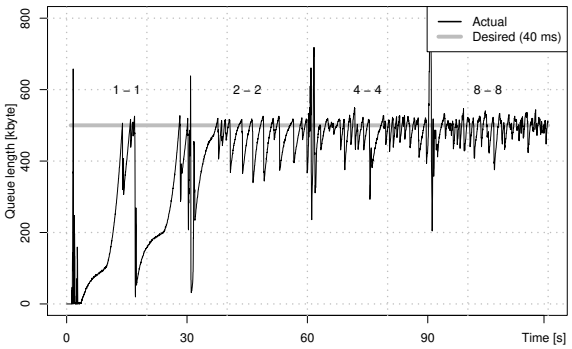

Figure 6: Scenario 1b, Queue length

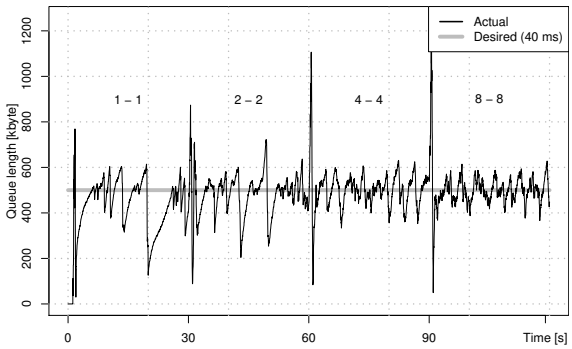

Figure 9: Scenario 1c, Queue length

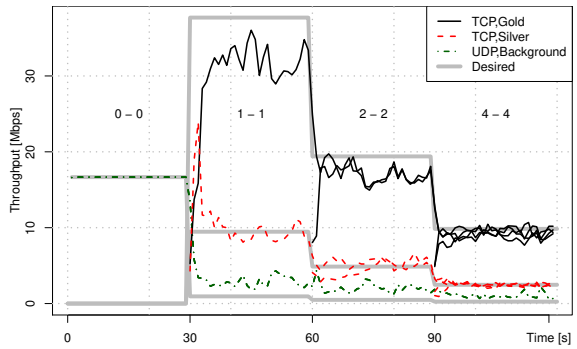

Figure 10: Scenario 2, Flow throughput

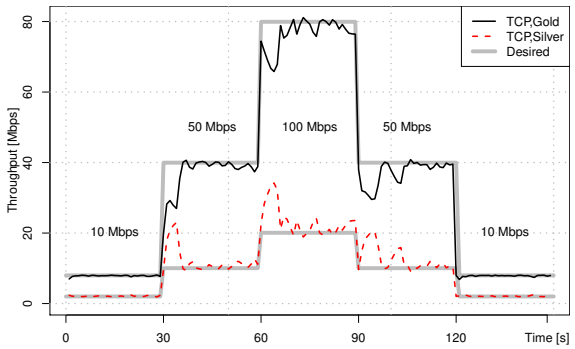

Figure 11: Scenario 3a, Flow throughput

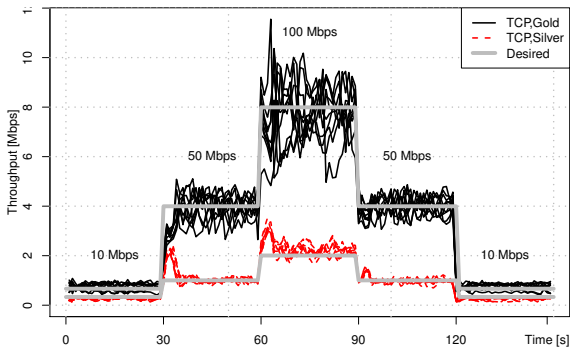

Figure 12: Scenario 3b, Flow throughput

the $N_{tcp.c} = 1$ case. Figures 6 and 9 show the length of the queue compared to the 40 ms target. One can observe that in both cases the target delay requirement is mostly satisfied and deviations can only be seen at transient points when the number of flows has been increased. Naturally, if $N_{tcp.c} = 5$, the overshot at transients is higher. For example, at 90 s on Figure 9, 40 TCP connections arrive simultaneously, instead of 8 connections on Figure 6.

Scenario 2: The effect of Background UDP flows on TCP traffic. Figure 10 shows the effect of aggressive Background UDP traffic on TCP flows, expressing a similar scenario to Figure 7 in [8]. There are 3 Background UDP flows of 30 Mbps during the whole measurement. Those share the available capacity equally in the beginning. At 30 s Gold and Silver TCP flows ($N_{tcp.c} = 1$) are added and it can be seen how these flows get a throughput close to their desired share even though the aggressive flows remain in the system. The number of Gold and Silver flows is then increased to 2-2 and 4-4 at 60 and 90 s. One can observe that the actual share is getting closer to the ideal as the number of flows increases. The reason behind this phenomenon is that individual TCP connections are vulnerable to packet drops. The impact of such packet drops caused by the aggressive UDP flows can especially seen between 30 and 60 s when there are only two individual TCP connections (1 Gold and 1 Silver).

Scenario 3: Dynamic bottleneck. Gold and Silver TCP flows. In this scenario we show the performance of the algorithm when the bottleneck capacity is changed every 30 sec to 10, 50, 100, 50 and 10 Mbps. Figure 11 depicts 1 Gold and 1 Silver TCP flows with $N_{tcp.c} = 5$, while Figure 12 depicts 10-10 flows with $N_{tcp.c} = 1$. The transient periods at capacity increase are due to time the TCP congestion control requires to reach the desired share after the capacity change. The higher fluctuations for Gold flows in the 10-10 flow case are due to $N_{tcp.c} = 1$. In this case a single packet drop event have high impact on the throughput of the Gold flows.

Scenario 4: Comparison to PIE results. Figure 13 depicts a scenario with 5 Gold TCP and 2 Silver UDP flows ($N_{tcp.c} = 1$). The bottleneck is 10 Mbps, the delay reference is 20 ms and the RTT is 100 ms. This scenario is the same as Figure 4c in [10][1], however the resource sharing in [10] is not shown[2]. It can be seen that the resource sharing is quite far from the Desired (0.83 Mbps for the Silver UDP and 1.67 Mbps for the Gold TCP), but the UDP flows still cannot overwhelm the TCP flows.

We explored a wider range of parameter settings to approach the desired bandwidth share. We changed the eCDF window (γ) up to $10 \cdot T$ and increased $N_{tcp.c}$ to 5. Figure 13 represents the worst case in this spectrum, while Figure 14 is the best with $\gamma = 10 \cdot T$ and $N_{tcp.c} = 5$. Figure 15 depict the filter value for this best case. The larger statistics interval and the larger number of TCP connections/flow results in a highly stable filter value. The filter was not reaching the ideal CTV and had high oscillations in the worst case. We evaluated two more cases[3]. One is with $\gamma = T$, $N_{tcp.c} = 5$, the other with $\gamma = 10 \cdot T$, $N_{tcp.c} = 1$. The bandwidth share and the filter value stability was better than for Scenario 4a in both cases, but worse than for Scenario 4b. We also experimented with $\gamma = 10 \cdot T$ for the previous scenarios, but that resulted in worse transient behavior. It is for future research to stabilize the filter further while keeping the transient behavior fast enough.

Finally, we plotted the queue length for the best case on Figure 16. It can be seen that the target is well met after an initial transient, similarly to our reference case (Figure 4c in [10]). The queue length was similarly good even in the worst case.

[30] It changes in every 30 sec.

[1] Of course, a resource sharing target is not defined in [10], while it is defined by TVFs for PVPIE.

[2] PIE cannot differentiate between TCP and UDP flows as PVPIE, thus we believe that the TCP throughput was close to 0, due to the very high loss rate in that case.

[3] Due to space limitation we do not show detailed figures for these cases.

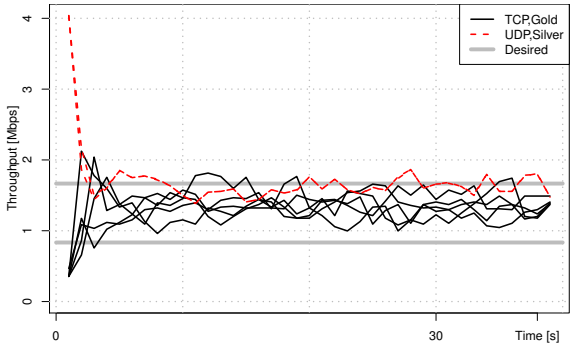

Figure 13: Scenario 4a, Flow throughput

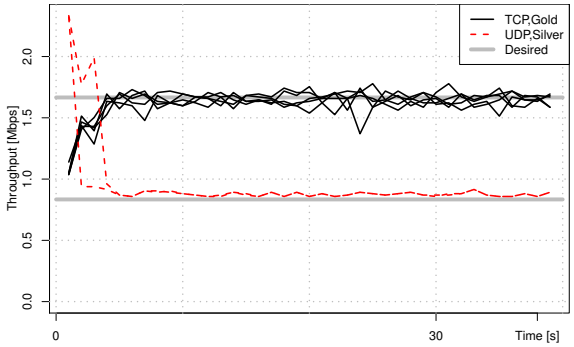

Figure 14: Scenario 4b, Flow throughput

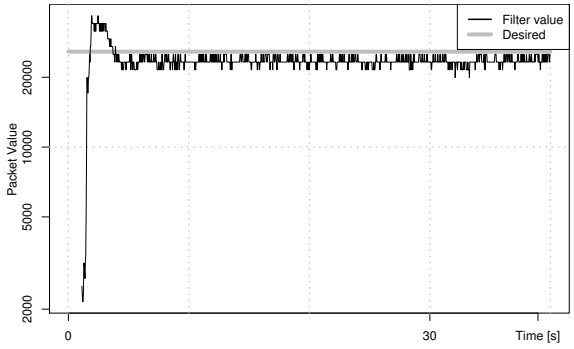

Figure 15: Scenario 4b, Filter threshold

5 CONCLUSIONS

We have created an AQM algorithm which can govern resource sharing based on the combination of PIE scheduler and the Per Packet Value concept. The algorithm calculates dropping probabilities using the PIE concept. Then instead of applying this drop probability directly on incoming packets, it translates that dropping probability to a Congestion Threshold Value filter and drops all incoming packets with Packet Value smaller than this threshold. The translation is performed by approximating the Packet Value distribution of incoming packets and selecting the Packet Value as filter threshold at the probability determined by the PIE algorithm. This method is more practical to implement than the one used in the original PPV concept paper as it decides about packet dropping at packet arrival, instead of dropping from the middle of the queue.

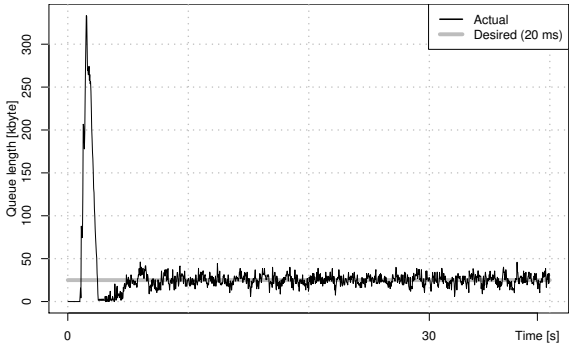

Figure 16: Scenario 4b, Queue length

We evaluated the AQM algorithm by NS-3 simulations comparing the results to both PIE and PPV baseline papers. We have shown that the algorithm can realize the desired resource share while keeping the queuing delay reasonable.

Our future work is to further fine tune and simplify the algorithm. As mentioned previously, parameter tuning may improve the efficiency of PVPIE at both transient and stationary phases, reducing the reaction time to changes of network conditions and providing more stable filter threshold values in stationary periods. We also aim at simplifying the conversion from dropping probability to Congestion Threshold Value filter, targeting more practical Packet Value-aware AQM algorithms with less memory and computational footprint.

ACKNOWLEDGMENTS

S. Laki and G. Gombos thank the support of Ericsson Hungary Ltd. given through the ELTE CNL cooperation.

REFERENCES

[1] Study on architecture for next generation system. 3GPP TR 23.799, February 2016.
[2] Z. Cao, E. Zegura, and Z. Wang. Rainbow fair queueing: theory and applications. *Computer Networks*, 47(3):367 – 392, 2005.
[3] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe. Pi2: A linearized aqm for both classic and scalable tcp. In *CoNEXT '16*, pages 105–119, New York, NY, USA, 2016. ACM.
[4] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, Aug. 1993.
[5] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.
[6] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. Network simulations with the ns-3 simulator. In *Sigcomm (Demo)*, volume 14, 2008.
[7] S. Jansen and A. McGregor. Simulation with real world network stacks. In *Simulation Conference, 2005 Proceedings of the Winter*, pages 10–pp. IEEE, Dec 2005.
[8] S. Nádas, Z. R. Turányi, and S. Rácz. Per packet value: A practical concept for network resource sharing. In *IEEE Globecom 2016*, 2016.
[9] K. Nichols and V. Jacobson. Controlling queue delay. *Commun. ACM*, 55(7):42–50, July 2012.
[10] 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 *IEEE HPSR*, pages 148–155, July 2013.
[11] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe. Open research issues in internet congestion control. Technical report, IRTF RFC 6077, February 2011.
[12] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Trans. Netw.*, 11(1):33–46, Feb. 2003.