Congestion Policing Queues - A New Approach To Managing Bandwidth Sharing At Bottlenecks

David P. Wagner
Institute of Communication Networks and Computer Engineering, University of Stuttgart, Germany

***Abstract*—Managing bandwidth sharing at bottlenecks is a challenge as old as packet switched networks. When equal senders compete for bandwidth of a bottleneck, it is desirable not only to enforce an instantaneous sharing of the scarce resource but also to prevent permanently active customers from suppressing less active customers. Moreover, it is desirable to incentivize shifting load to non-congested networks or times. Today there is no cheap, efficient and effective mechanism available to achieve these goals. It has been argued that policing based on congestion as perceived by the transport layer can achieve these goals. In this paper we present the concept of Congestion Policing Queues (CPQ), based on a very lightweight dequeuing and scheduling because all customers share one queue. CPQs can police congestion if deployed at bottlenecks relevant to the customers' traffic. We developed three base policers that differ in the level of integration with the Active Queue Management (AQM) of the shared queue. By simulations of three scenarios we evaluate the robustness of the achieved resource sharing and performance in partial deployments for multi-bottleneck situations.**

I. Motivation

Network operators aim to satisfy the requirements of as many customers as possible. Since any information about requirements retrieved from the customers is prone to be selfish, the best option is enforcing equal sharing between all (active) customers (for each class of service) when and where resources are scarce, i.e. at bottlenecks. Therefore, today many operators design their networks in order to pinpoint the bandwidth bottleneck to certain nodes, e.g. the Broadband Network Gateway (BNG), and deploy technologies such as Deficit Round Robin (DRR) [13] on these nodes. By that, instantaneous equal bandwidth sharing is achieved, i.e. all N active customers obtain $\frac{1}{N}$ of the total bandwidth. Often, such schedulers are configured hierarchically, e.g. per service class, [7] and partly use classification based on Deep Packet Inspection (DPI). But this approach has substantial disadvantages. Managing one one queue per service class per customer and having the scheduler operating on that many queues is expensive. These mechanisms also ignore time: if a customer has been inactive for a long time, one would suggest this customer should get precedence over permanently active customers for some time. Moreover, such system do not incentivize distributing resource usage both in time and regarding resources, which would be desirable.

Policing each customer's congestion at the customer's point of attachment conceptually avoids these shortcomings [9]. Unfortunately, complete information about the congestion caused by a customer's traffic is not available to today's network operators. Nevertheless, for many transmissions the relevant bottleneck lies within their Internet Service Provider (ISP)'s network and not at peering points, as e.g. the broadband reports of public agencies such as the Federal Communications Commission (FCC) indirectly show.

In this paper we present the concept of Congestion Policing Queues (CPQs). It aims to contain the congestion any single customer creates by a per-customer policing based on locally acquired congestion information. It eliminates the need for complex hierarchical queuing and scheduling but allows using a single shared queue for all customers. The concept of CPQs works in scenarios where traffic passes one or more bottlenecks. Therefore, CPQs will be beneficial in particular to cope with varying bottlenecks in the aggregation networks which could become commonplace at peak hours with future high-speed access links, e.g. Passive Optical Networks (PONs).

This paper is structured as follows: Section II gives an overview of related work. Section III presents three bottleneck congestion policing mechanisms, that are evaluated in Section V. The paper closes with conclusions and an outlook in Section VI.

II. Related Work

CPQs aim to enforce a certain resource sharing among an unknown and changing number of customers. Therefore, we shortly describe not only congestion policing but also other queue management and scheduling concepts.

Congestion policing assumes that congestion is the cost that one customer can inflict on other customers and therefore should be limited [11]. This accepted congestion rate may be a just an internal technical policy of the network operator or may also be explicitly contracted with the customers.

A. ConEx Policing

Congestion Exposure (ConEx) [1] is a technical protocol proposal being developed by the IETF to make congestion measurable on the path of a flow. The basic concept is that senders actively expose the congestion they detect to the network by in-band signaling. It depends on the receiver of the flow relaying information about experienced congestion back to the sender as is the case for Transmission Control Protocol (TCP). ConEx only concerns congestion visible as packet loss or Explicit Congestion Notification (ECN) markings. ConEx is also intended for congestion policing [4] and some simulations are documented [14]. Today there exists no final protocol standard and consequently there is no deployment.

978-3-901882-48-7

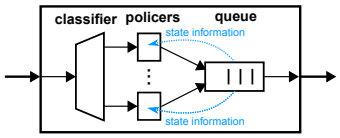


Fig. 1: Generic overview of a Congestion Policing Queue

B. Queue management targeting Fairness

Several Active Queue Management (AQM) mechanisms have been developed that target fairness, many targeting special deployment environments. The most prominent generic AQM targeting fairness is Fair Queuing (FQ) [6], which is also used combined with specific AQMs for the sub-queues, such as FQ-CoDel (using Controlled Delay (CoDel) [12]). FQ always enforces per-flow fairness and has no notion of customers. So these algorithms actually aim for a different goal.

If per-customer equal sharing shall be achieved, ISPs often explicitly configure classification and rate-limited queues and DRR scheduling on central network nodes so that these become the only bottleneck. This approach reliably achieves robustness against differing aggressiveness of customers, but does not take into account the varying activity of customers. Such system also cannot provide incentives for shifting traffic to less congested networks or times: Even if a customer deliberately sends less aggressive, e.g. by using so-called scavenger protocols as Microsoft Windows does for its updates, such configuration will still allocate the configured bandwidth.

III. Bottleneck Congestion Policing

A. Generic Structure of a Congestion Policing Queue

On a high level, an implementation of a CPQ consists of three components as depicted in Figure 1:

- a classifier mapping incoming traffic to customers
- congestion policers for all customers filling this queue
- one queue, possibly with an AQM.

The per-customer congestion policers, are independent from the classification mechanism as well as of the queue. Most importantly it can be combined with any mechanism deployed today, e.g. in BNGs, without modification.

B. Generic Structure of a Congestion Policer

On an abstract level, any congestion policer defines four core functions: congestion allowance, congestion assessment, allowance consumption and drop decision. By configuring the congestion allowance the operator can adjust the resource sharing in times of congestion between its customers according to its policies, e.g. depending on the contract of the customers. In contrast, the design of the other components define the behavior of a congestion policer, its performance and robustness.

We adopt the token bucket concept to translate the constant money per time the customers pay to the instantaneous variable congestion price [5], that Kelly proved would optimize the global Internet [11]. So the central element of a congestion policer are one or more congestion token buckets as depicted in Figure 2. They have also been used by others ([5] [14]) for

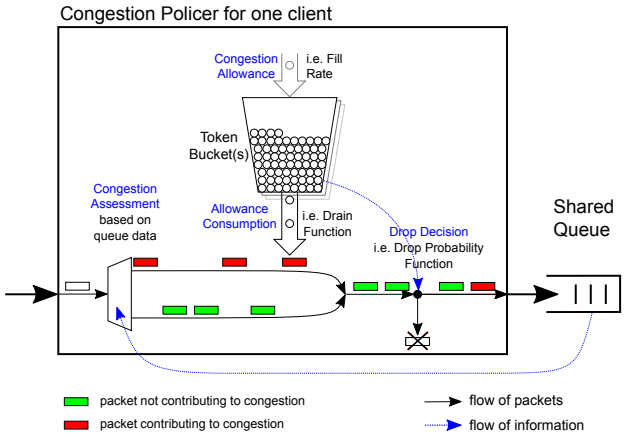


Fig. 2: A Congestion Policer of a CPQ for one customer

congestion policing. The token buckets have a specified size, are filled with a certain rate and are drained only for packets contributing to congestion. The fill rate defines the tolerated sustained congestion rate of that customer.

1) Congestion Allowance: The size(s) and the fill rate(s) of the token bucket(s) are configured by the ISP. We only use one bucket since we examine basic properties of CPQs.

2) Congestion Assessment: Essentially, this function assesses if and how much a packet contributes to congestion. This can be realized very differently: one option is to count dropped / marked packets (similar to the input ConEx would provide), others assign a real number between zero and one for every packet. In any case the policer's understanding of congestion should correspond to the signals the queue provides to the sender to create the desired incentives to shift traffic to less congested paths or times.

3) Allowance Consumption: If a customer's packet contributes to congestion, the customer's congestion credit in the token bucket is reduced by $size(packet) * congestion\_contribution(packet)$. If packet drops are not the only source of congestion information, a difficult question is if packets dropped by the AQM should be accounted for: It can be argued that a packet drop is a clear sign for congestion so tokens should be consumed. Nevertheless, dropped packets do not contribute to congestion just because they are dropped and not transmitted, so it makes sense not to drain tokens.

4) Drop Decision: The drop decision is a drop probability function that maps the bucket fills to a drop probability and forwards or drops packets accordingly. It may depend on conditions, e.g. so that packets are dropped only during congestion periods.

IV. Local Congestion Policing Algorithms

We implemented seven different policing algorithms which we classified into three types, depending on the level of integration with the accompanying AQM.

A. Independent Congestion Policing

Independent Congestion Policing (ICP) algorithms are independent of the queue's state and just receive information

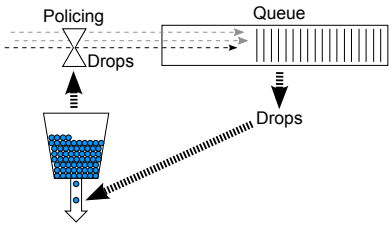


Fig. 3: Control flow between policer and AQM for ICP

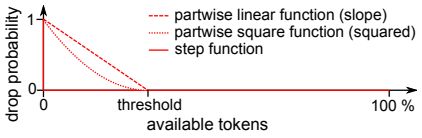


Fig. 4: Forwarding decision functions implemented for ICP

on packet drops or ECN marks. So they only use congestion signals that are also visible to the receiver, resembling the input ConEx provides. While ConEx covers the whole path (with one Round Trip Time (RTT) delay), a CPQ can only use the drops / marks at the local queue. Figure 3 shows the general concept.

Due to lack of context information, any dropped or marked packet consumes tokens corresponding to its size. Since [14] showed that the step function is a suboptimal choice for ConEx policing we implemented and evaluated three drop probability functions depicted Figure 4: a step function, a piecewise linear and a piecewise square function. The step function is expected to result in on-off behavior for too aggressive customers: Once the bucket fill level has fallen too low, all packets requiring more tokens will be dropped. Since packets of bigger transmissions in today's Internet are all of similar size (∼1500 Bytes), the policer will resume forwarding packets once the bucket has a sufficient fill level. If the customer will continue causing too much congestion, this process will repeat.

ICP is attractive since the policer can be implemented as independent device. If existing routers' firmware could be altered so that the AQM does not drop packets but forwards them to the policer, subsequent deployment is possible for ICP.

B. AQM-Fed Congestion Policing

AQMs such as Random Early Detection (RED) [8] have been introduced in order to avoid global synchronization of TCP flows, RED being the only AQM that is widely available in deployed equipment. RED drops packets based on a drop probability depending on the moving average of the queue size $\widetilde{q}$. The drop probability function is zero for $\widetilde{q}$ smaller than a minimum threshold min_thresh and then increases linearly to a maximum probability max_p at a maximum threshold max_thresh. It equals one for greater $\widetilde{q}$. So RED as probabilistic AQM assigns a drop probability $drop\_prob$ to each incoming packet, which can be interpreted as the congestion currently perceived by the queue.

The basic idea of AQM-Fed Congestion Policing (AFCP) [3] by Briscoe is to use the drop probability of an AQM instead of the actual drops or markings, in particular also for packets

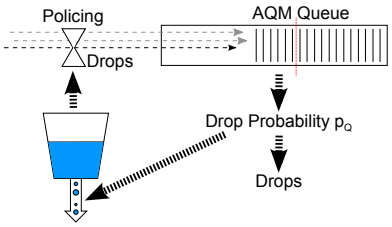


Fig. 5: Control flow between policer and AQM in AFCP

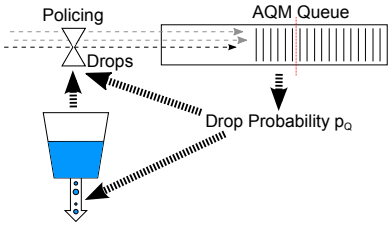


Fig. 6: Control flow between policer and AQM in AICP

that are not dropped. Figure 5 abstractly shows the flow of information between the AQM and a selected customer's policer. We designed AFCP to adopt that understanding of congestion of the accompanying AQM without modification for both the allowance consumption and the forwarding condition: Any incoming packet will only be forwarded to the queue if there are still enough tokens available, i.e. its size scaled by the AQM's current drop probability:

$$drain = size(pkt) * AQM.drop\_prob$$

and any forwarded packet will consume $drain$ tokens. We investigated two variants of this drain algorithm according to the two views on dropped packets discussed in III-B3: AFCP-a drains tokens for every packet let pass by the CPQ, while AFCP-b does not drain tokens for packets that pass the policer but are dropped by the AQM.

In order to access to the AQM state, AFCP requires integrating the policing function with the AQM in one device and efficient implementation of calculations with the drop probability.

C. AQM-Integrated Congestion Policing

Since AFCP does not alter the AQM itself, even customers who don't use more than their fair share will stochastically get congestion signals. The goal of AQM-Integrated Congestion Policing (AICP) is that customers with a full bucket have a much lower drop probability than customers who mostly consume their allowance rate, i.e. have almost empty buckets. As Figure 6 shows, for AICP there are no independent AQM drops. The drop decision is defined by the policer for the customer accountable for that packet, taking into account both the queue state and the state of the policer. There is further research needed on how to trade off prioritization of customers, link utilization and queuing delay. We developed and implemented two basic variants.

AICP-a is based on shifting the slope of the RED drop probability function depending on the fill level of the customer's token bucket. Therefore the drop probability for a packet of

a specific customer is computed by the RED algorithm, but both thresholds used by the RED algorithm are shifted by the fill ratio of the token bucket, e.g. for min_thresh:

$$min\_thresh = AQM.min\_thresh * (1 + \frac{current\_fill}{bucket\_size})$$

Because this variant uses in average larger thresholds, in particular after idle times, it tends to allow larger queues and queuing delays, which might be not acceptable for some use cases.

AICP-b does not touch the thresholds but manipulates the drop probability. We decided to inlcude the RED's drop probability red_prob but to reduce its influence by squaring it. On top, we add a drop probability p_prob depending on the fill of the token bucket:

$$drop\_prob(cstmr) = red\_prob * red\_prob + p\_prob(cstmr)$$

In our implementation we chose p_drop_prob as slope linearly increasing from zero at a threshold bucket fill p_ratio_thresh (we use 0.25) to a maximum probability p_max (we use 0.5) for an empty bucket. So for customers having more than 75% of their tokens consumed, the drop probability is derived as

$$p\_prob(cstmr) = p\_max * (1 - \frac{fillRatio(cstmr)}{p\_ratio\_thresh})$$

V. Simulative Evalutaion

We use IKR SimLib [2], an event-driven simulation framework, which allows integrating latest operating system kernels using full virtualization approach [15]. This allows using TCP congestion control of real deployed operating systems. We choose to use Linux version 3.10.9 and TCP cubic since many senders in the Internet are Linux machines and cubic is the default congestion control since end of 2006. All simulations ran for 5020 seconds, divided into 20 seconds startup phase (ignored in statistical evaluation) and ten 500 second measurement intervals.

A. Overview

The selected three scenarios aim to cover the crucial challenges for CPQs: robustness of resource sharing and deployability. More precisely, we examine the resource sharing when facing differing aggressiveness of the customers' congestion control or differing activity patterns of the customers in a single-bottleneck scenario as typical for today. Furthermore, we examine the effect of partial deployment in multi-bottleneck scenarios. A lightweight, well-performing capacity sharing mechanism would allow ISPs to limit their investments in aggregation networks. So we evaluate three scenarios: aggressiveness scenario, activity scenario and mixed deployment scenario.

In all three scenarios, we model the extreme case where just two customers compete for the scarce bottleneck bandwidth. For all scenarios, we use the seven different Policing and AQM mechanisms presented in IV. After some calibration experiments, we decided to use the set of congestion allowance configurations shown in Table I for all experiments. Since we target to capture a stable state for the persistently sending customers, the bucket sizes are chosen rather small to reach a stable state fast. The bucket fill rate has to be aligned to the way the algorithm accounts congestion to a customer, so we identified one parameter for drop-based and one for probability-based algorithms.

TABLE I: Token bucket configuration used in the simulations

	ICP	AFCP	AICP
Bucket size [Bytes]	350000	225000	225000
Token fill rate [Bytes/s]	550	300	300

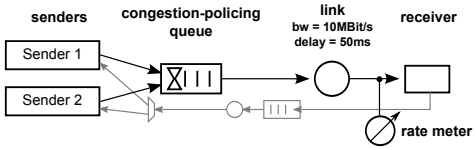


Fig. 7: Simulation topology used in the Aggressiveness and Activity Scenario

B. Aggressiveness Scenario

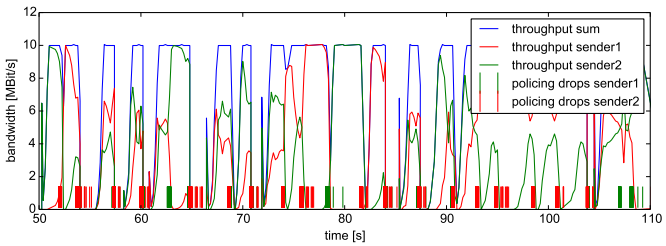
We evaluated the robustness regarding aggressive senders using the scenario depicted in Figure 7. Sender 1 uses 5 concurrent TCP cubic flows that have data available at any time (greedy flows), while sender 2 just uses one greedy flow. In average, agnostic queues, drop tail as well as RED, would result in a per-flow fair resource sharing, i.e. the sender 1 would acquire five times the bandwidth of sender 2.

1) Observations: As expected, the heavy allowance exceedance results for all three ICP variants in on-off behavior, as visible in Figure 8a showing a trace for ICP using a slope drop probability. The policing drops occur in bursts, followed by a time without policing. During significant times the link is by far not fully utilized. For AFCP-a, shown in Figure 8b, the policing drops are also clustered, but the distance between clusters is bigger. The bandwidth sharing seems much more equal than five to one, the utilization of the bottleneck is high. In Figure 8c the effects of AICP shifting the regulation of bottleneck access from the AQM to the policer are obvious: The frequency of policing drops in Figure 8b is remarkably higher compared to AFCP.

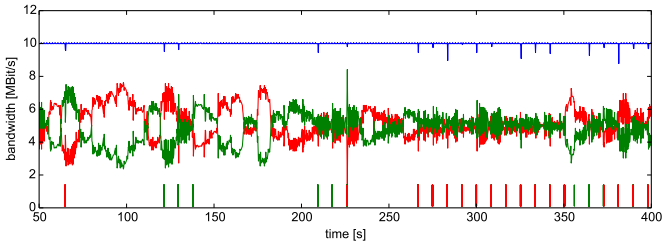
2) Statistical evaluation: For the Aggressiveness Scenario, the important metrics are Jain's fairness index [10] and the bottleneck link utilization, given in Table II. All algorithms achieve higher fairness than a RED queue, but without the effort of managing several queues as for DRR. The achieved bottleneck utilization for ICP is not acceptable. ICP may only be useful if the purpose is deterring customers from overly causing congestion rather than policing traffic.

TABLE II: Results for aggressiveness scenario

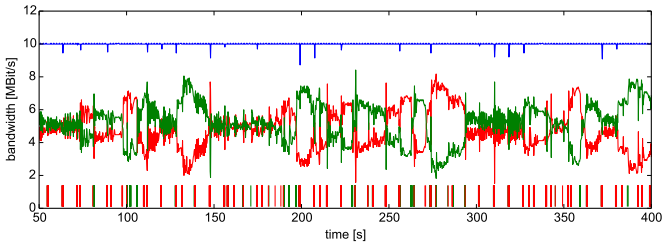
	Reference		ICP			AFCP		AICP	
	RED	DRR	Step	Slope	Sqrd	a	b	a	b
Jain's Fairness	0.575	1.0	0.996	0.990	0.997	1.0	0.951	0.994	0.999
Bottleneck Utilization	1.0	1.0	0.790	0.783	0.785	0.999	1.0	1.0	1.0



(a) Trace for using ICP with a slope drop probability function



(b) Trace for using AFCP-a



(c) Trace for using AICP-a

Fig. 8: Traces for aggressiveness scenario

C. Activity Scenario

In this scenario we use the same topology shown in Figure 7 and customer 1 maintains one permanent greedy TCP connection In contrast, the customer 2 just has sporadic and short transmissions, which follow a statistical distribution both in length and inter arrival times. We chose a normal distribution (mean = 50 s, standard deviation = 5) for the inter arrival times and a binomial distribution for the transmission length (mean = 2 E6 and variance = 5 E5). The key metric is the mean finish time of the short transmissions, although the bottleneck utilization should be high as well.

Table III summarizes the statistical results for the activity scenario. Regarding finish times only the mean is of statistical relevance, nevertheless we chose to include minimum and maximum values to indicate the spread of results. The variants of ICP achieve the fastest finish times for the sporadic transmissions. This benefit is only possible because the link is temporarily empty, i.e. it is paid for by a very low link utilization. The results for AFCP show a small advantage for variant a, but no fundamental differences. Whereas the results for AICP show a big difference between the two variants. AICP-a achieves reliably low average finish times. Variant b performs much worse, showing about 50% longer finish times in average. This can be explained by AICP-a reducing the drop thresholds for the permanent customer and by that leaving space in the queue for customers becoming active. Nevertheless, this approach is disputable since with our

TABLE III: Results for Activity Scenario

		ICP			AFCP-a	AFCP-b	AICP-a	AICP-b
		Step	Slope	Squared				
Finish Times	Min	2.05	02.05	2.05	2.73	2.71	2.65	3.12
	Mean	3.99	3.92	3.61	5.29	5.35	4.42	6.45
	Max	8.10	7.17	8.19	8.07	8.78	4.82	14.14
Bottleneck Utilization		0.729	0.710	0.723	0.992	0.991	0.987	0.990

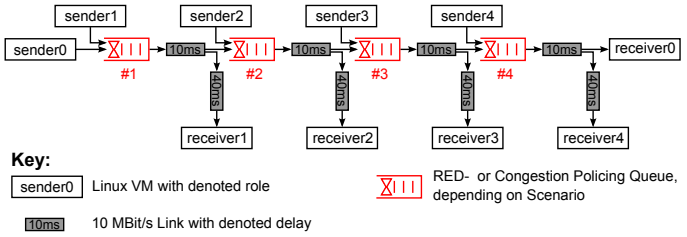


Fig. 9: Topology used in Mixed Deployment Scenarios

configuration it increases thresholds for customers with full buckets, thus resulting in bigger queue size and larger queuing delays. The balance between the three metrics, utilization, precedence for newly active customers and low delay requires more research.

D. Mixed Deployment Scenarios

We consider a greedy flow of customer 0 that passes four congested links in the topology depicted in Figure 9. The competing flows only pass one bottleneck, but link delays are defined as depicted in Figure 9 so all connections experience a similar RTT. By this, we limit the impact of the inherent RTT-unfairness of current TCP congestion controls. In this scenario, various sub-sets of queues are configured as CPQs, whereas the remaining queues are configured as RED queues.

This experiment targets to prove that even if there are more than one bottleneck on a path and CPQs are partially deployed, there is benefit and robustness to expect. Therefore, we compared the results with a deployment of RED queues only and looked on bottleneck utilization and fairness between flows passing a different number of bottlenecks. In Table IV the exact results are given.

We evaluated the average utilization of the four links shared by two flows each (the horizontal links in Figure 9), see Table IVa. ICP results in significant bottleneck bandwidth being not used, so wasting scarce resources, AFCP and AICP result in almost full utilization ($\geq 99.5\%$).

Most interestingly, we analyzed the resource sharing between traffic passing just one and traffic passing four queues (i.e. flow 0 vs. the average of flows 1–4). All implemented policing algorithms do improve fairness, but the degree depends on the algorithm and the amount of deployed CPQs. If there is just one CPQ deployed, the effect is much lower than if three or all four queues are configured as CPQs. In these cases, ICP policing results in higher fairness than AFCP (at expense of wasting scarce resources, see above). In contrast, AICP policing is able to achieve a Jain's Fairness Index of above 0.95 for both variants whenever there are at least three of the four queues CPQs.

TABLE IV: Results for the Mixed Deployment Scenario

(a) Average utilization of bottleneck links

CPQs at bottlenecks	ICP			AFCP-a	AFCP-b	AICP-a	AICP-b
	Step	Slope	Squared				
none	0.999						
#1	0.953	0.949	0.949	0.999	0.999	0.998	0.998
#1 – #3	0.854	0.841	0.848	0.999	0.998	0.996	0.997
#3	0.954	0.947	0.952	0.999	0.999	0.999	0.998
#2 – #4	0.860	0.848	0.847	0.998	0.998	0.996	0.998
all	0.827	0.798	0.816	0.998	0.998	0.996	0.996

(b) Jain's Fairness of the traffic passing one vs. the traffic passing four queues

CPQs at bottlenecks	ICP			AFCP-a	AFCP-b	AICP-a	AICP-b
	Step	Slope	Squared				
none	0.693						
#1	0.717	0.716	0.712	0.695	0.698	0.732	0.743
#1 – #3	0.762	0.771	0.767	0.703	0.705	0.942	0.952
#4	0.711	0.718	0.710	0.700	0.697	0.744	0.753
#2 – #4	0.760	0.767	0.766	0.709	0.718	0.953	0.953
all	0.795	0.820	0.817	0.719	0.717	0.978	0.991

This is a very interesting finding, since it shows that CPQs using AICP are able to compensate the negative effect on throughput when passing several bottlenecks without reducing utilization. The AICP algorithm achieves this global effect by just using local information, without any signaling.

E. Discussion

All ICP algorithms fail to utilize the scarce resource for many conditions. Our findings show that there are much better options if the AQM's information can be accessed. We therefore deem ICP not suitable for deployment.

Regarding AFCP the results show advantages for AFCP-a, which drains tokens for dropped packets. AFCP-a is very well able to enforce equal sharing independent from the aggressiveness of the senders, comparable to DRR. Moreover, in average AFCP-a reduces finish times for sporadic senders, however not as much as AICP. AFCP does not change the tendency to put flows which pass several congested queues at a disadvantage. This can be an advantage or an disadvantage, depending on the goals of the network operator.

With AICP policers, the number of traversed bottlenecks is much less important than with other policers or even without policing. This allows deliberate routing through several bottlenecks while not putting the respective customers at a disadvantage. This might be an important benefit if aggregation networks need to be operated at their capacity limit at peak hours and in consequence many links become bottlenecks from time to time.

VI. Conclusion

In this paper we presented the concept of a Congestion Policing Queue, an element combining queuing and per-customer congestion policing. We analyzed the general structure of the policing element and designed algorithms that we differentiated to three classes:

- ICP which is independent of the queue and its AQM
- AFCP which uses state information of the AQM
- AICP which integrates the AQM / supersedes the AQM.

We evaluated CPQ with these policer algorithms in simulations using real Linux TCP cubic traffic regarding robustness against differing aggressiveness of the customers' congestion control and differing activity patterns. Furthermore, we evaluated the effects of mixed deployments. Our results show that AFCP and AICP policers achieve high utilization and mostly also the desired bandwidth sharing at the bottleneck. Moreover, AICP policers are able to compensate the negative effect on throughput when passing several bottlenecks without reducing utilization.

Summarizing, CPQs can enforce equal resource sharing as DRR would, but significantly improve congestion fairness for the customers and at the same time increase the operator's flexibility in operation. Although further research in algorithms is needed, our results indicate that any good policer algorithm for CPQs will be fed by or integrated with the AQM.

VII. Acknowledgements

The foundation of this work is the idea of Bob Briscoe who pointed out that congestion policing can be performed on local congestion information at a bottleneck.

This work has been funded by the German Federal Ministry of Education and Research (BMBF) in the SASER project (grant identifier 16BP12202).

References

[1] Congestion exposure (conex). http://datatracker.ietf.org/wg/conex/, April 2014. retrieved at 2014-05-14.

[2] Ikr simulation and emulation library. http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/, April 2014. retrieved at 2014-05-14.

[3] B. Briscoe. nice traffic management without new protocols. http://www.bobbriscoe.net/presents/1210isoc/1210isoc-briscoe.pdf, October 2012. retrieved at 2014-05-14.

[4] B. Briscoe. Network Performance Isolation using Congestion Policing. Internet Draft draft-briscoe-conex-policing-01, Internet Engineering Task Force, Aug. 2014. Work in progress.

[5] B. Briscoe et al. Policing congestion response in an internetwork using re-feedback. *Proc. ACM SIGCOMM 05, Computer Communication Review*, (TR-CXR9-2004-001):277–288, 2005.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19(4):1–12, Aug. 1989.

[7] DSL Forum. Migration to ethernet-based DSL aggregation, 2006.

[8] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.

[9] A. Jacquet et al. Policing freedom to use the internet resource pool. In *Proceedings of the 2008 ACM CoNEXT Conference*, pages 71:1–71:6, New York, NY, USA, 2008. ACM.

[10] R. Jain et al. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *CoRR*, cs.NI/9809099, 1998.

[11] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 1997.

[12] K. Nichols and V. Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012.

[13] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, 4(3):375–385, Jun 1996.

[14] V. Singh (Editor) et al. Final report on resource control, including implementation report on prototype and evaluation of algorithms, December 2010.

[15] T. Werthmann et al. VMSimInt: a network simulation tool supporting integration of arbitrary kernels and application. In *SIMUTools '14: Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*. ACM, March 2014.