Theoretical and Applied Informatics
ISSN 1896–5334
Vol.24 (2012), no. 3
pp. 205–225
DOI: 10.2478/v10179-012-0013-4

On the optimization of the inter-flow fairness in the Internet

BŁAŻEJ ADAMCZYK, ANDRZEJ CHYDZIŃSKI

Institute of Informatics, Silesian University of Technology
Akademicka 16, 44-100 Gliwice, Poland.
{Blazej.Adamczyk,Andrzej.Chydzinski}@polsl.pl

Received 1 August 2012, Revised 3 September 2012, Accepted 1 October 2012.

Abstract: All known active queue management algorithms invented to provide fair bandwidth allocation between TCP flows are designed to cooperate with the classic TCP congestion control (New Reno). However, some new congestion control schemes are becoming more and more popular nowadays (e.g. the Cubic algorithm). Therefore, the following question arises: will these fair queue management algorithms work well in the presence of a new congestion control scheme? To answer this questions, we present a comprehensive study of the performance of seven fair queue management algorithms in the presence of seven TCP variants. In particular, the fairness index, queue size and throughput were measured in scenarios with diversified RTTs, traffic patterns and congestion levels. Not only do the results allow us to answer the aforementioned question, but also to formulate recommendation on how to provide the best cross-layer fairness optimization in the Internet.

Keywords: fairness, active queue management, congestion control

1. Introduction

It is widely recognized that the common combination of the Internet congestion control mechanism with the router queueing algorithm, that is the New Reno mechanism with the drop-tail FIFO queue, does not provide a fair division of the bandwidth at bottleneck links. In particular, connections with long RTTs do not get their fair shares of the throughput. Moreover, the aforementioned mechanisms are vulnerable to attacks of misbehaving flows that do not obey common congestion control rules.

The presented unfairness problem has been addressed from two distinct perspectives. On one hand, more fair variants of the TCP congestion control algorithms have been proposed. For instance, the Hybla TCP [1] achieves the fair bandwidth allocation by means of overcompensation of the congestion window for connections with long RTTs.

The Cubic TCP [32] improves the inter-flow fairness by using the real time (rather than the RTT cycles) for the evolution of its congestion window.

On the other hand, there has been a lot of researches on how to provide the fairness at the network level. Several fairness-driven active queue management (AQM) algorithms were proposed (see [3] for an excellent survey of them). They use a wide range of techniques to decide which flow is using more bandwidth than its fair share is and drop more packets belonging to this detected flow. In some algorithms (e.g. CHOKe [4]) there is no explicit detection of flows consuming too much bandwidth – the balance between the bandwidth consumption is maintained automatically due to ingenious queue management algorithm.

Now, it must be stressed that most of the previous studies were focused on one type of solution only, i.e. either on the network layer (AQM) or at the transport layer (TCP congestion control). When a new AQM was proposed, it was tested with the classic congestion control (New Reno or, rarely, Sack). Similarly, after a new TCP variant was invented, it was evaluated with the classic queue management (drop-tail FIFO queue). In this paper we present a study of the interactions between the TCP congestion control mechanisms and the active queue management algorithms. To the best of our knowledge, this is the first study of this type. In particular, we demonstrate the influence of the newest Cubic TCP on the fairness provided by several AQMs, designed with New Reno congestion control in mind. Moreover, basing on the large number of simulation results we formulate recommendation on the best way to optimize inter-flow fairness using both, network and transport layer solutions.

To achieve these goals, we built a simulation setup with diversified propagation delays and traffic types. Then, we applied seven AQMs and seven TCP variants in this setup and checked the resulting performance in three different congestion levels (this made 147 distinct simulation scenarios in total). In each simulation scenario we observed the inter-flow fairness index (Jain's index), the per-flow throughput, the aggregated throughput and the bottleneck queue size. Among many TCP variants described in the literature, we chose seven that are actually implemented in modern operating systems and commonly used: New Reno, Sack, Fack, Vegas, Westwood, H-TCP and Cubic. As for the fair AQMs, we chose all the algorithms that have publicly available code, namely CHOKe, SFB, CARE, RED, FRED, SRED and DT.

As regards the related work, the closest to our paper is [5]. The authors propose a network rate management protocol (RMP) which, among other things[1], computes the target rates for a new TCP sliding-window-based congestion control algorithm. Therefore, each TCP flow can adapt its window size to achieve the RMP suggested fair bandwidth share. The authors tested the new solution by means of simulation of a network comprising 74 core links and up to 768 flows, each using its own access link. They also

[1]The RMP protocol has also QoS-oriented functionalities and has been used in several QoS architectures – see [6,7,8] for more details.

carried out an actual implementation in the Linux kernel and performed experiments in a WAN testbed network with six routers and long haul links running UDP flows and TCP flows of different types. The obtained simulation and experimental results are very good.

The main difference in comparison with our paper is that [5] proposes a new comprehensive solution with new protocols at the network and transport layer, designed to cooperate with each other. We on the other hand use the well-known algorithms (many of them already widely used), and try to find a combination that solves the unfairness problem. Naturally, a clean-slate solution enables obtaining better results, but it is also harder in actual deployment.

Other two papers that are close to our study are [9] and [10]. In both of them interactions between AQM algorithms and different TCP variants are studied as well. However, only the general-purpose AQM algorithms are used there. In this paper we deal with different type of active queue management schemes – only the algorithms supporting the inter-flow fairness are considered. Moreover, we evaluate the cooperation between the AQM and TCP from this perspective (i.e. with fairness in mind).

All the conclusions drawn in this paper are based on simulation results. We did not attempt mathematical analysis of the interactions between the aforementioned TCP variants and AQM algorithms due to the following reason. The AQM and TCP mechanisms studied herein have typically a complex structure which is hard to mimic using solvable mathematical models. Even the widely disputed RED algorithm has no good and solved models. Some attempts to analyze RED, e.g. [11,12,13], are based on significant simplifications[2] and therefore their applicability is limited. Furthermore, some AQMs studied herein are much more complex than RED. For instance, CHOKe has a built-in RED as a part of its functionality. Now, the situation considered herein is even worse than in a pure AQM study. To obtain analytical results we would have to combine the AQM models with TCP models (also far from perfect), and solve the resulting combined models. This seems to be far beyond the reach.

The remaining part of the paper is structured as follows. In Section 2, the fairness-driven AQMs are presented. Section 3 describes the TCP congestion control algorithms used in the paper. In Section 4, the details of the scenarios used in simulation experiments are given. Then, in Section 5, the simulation results are presented and discussed. Finally, in Section 6 the conclusions are gathered.

2. The queue management algorithms

The active queue management schemes can be classified in several ways. As the network performance can be described by various characteristics (e.g. throughput, queue

[2]For instance, non of the aforementioned RED models take into account the moving average of the queue size, which is crucial for the proper operation of a real RED router.

size, fairness), different AQM algorithms are focused on optimizing one or more of these characteristics. In this paper we deal with fair AQM schemes, that is with schemes that focus on providing the best possible inter-flow fairness.

The fair AQMs can be further divided into three categories. Some schemes, like Fair RED (FRED) [14], Longest Queue Drop (LQD) [15], or Balanced RED (BRED) [16], consume lots of memory, as they require to store all per-flow information. Therefore their scalability is limited. Stabilized RED (SRED) [17], Capture-Recapture (CARE) [18], and Blacklisting unresponsive flows (BLACK) [19], are examples of AQM mechanisms estimating the number of active flows instead of storing full per-connection information. Naturally, this approach is more efficient and decreases the memory consumption by far. Algorithms belonging to the last group are less resource consuming – they don't even require to estimate the number of active flows. Examples of such schemes are Random Early Detection (RED) [20], Stochastic Fair Blue (SFB) [21], and CHOKe [4].

2.1. Drop tail queue

Drop Tail (DT) is the only algorithm used herein, which is not in fact an active queue management scheme. It appears in this paper because it is still the most popular algorithm implemented by router vendors. The operation of DT algorithm is very simple: if the queue is full, incoming packets are dropped, otherwise packets are accepted. All packets are treated with the same priority and there is no distinction between packets coming from distinct flows.

2.2. Random Early Detection

This is the widely known AQM that is able not only to reduce the queue sizes and desynchronize TCP flows but also it deals surprisingly well with the fairness issue. The idea behind Random Early Detection (RED) [20] is to calculate the average queue length each time a packet arrives and decide randomly whether to drop the packet. The larger the average queue length is, the bigger the probability of dropping the incoming packet. RED introduces two additional parameters: maximum and minimum threshold. If the average queue length is less than minimum threshold the packet will be enqueued with probability 1. Similarly, if the average queue length is bigger than maximum threshold, the packet will be dropped with probability 1.

As far as fairness is concerned, such approach is much better than simple tail drop procedure because it does not discriminate short lasting connections. It drops packets randomly almost all the time which results in bigger probability of dropping packets from the more active connections.

2.3. Fair RED

FRED [14] is a modification of RED algorithm created to further improve the fairness among different flows. For each flow it stores the queue length ($qlen$) as well as the number of times the flow does not respond to congestion ($strike$). Such information is sufficient enough to decide whether to drop a new incoming packet belonging to a given flow. Two additional parameters are available: min_q and max_q. By using min_q it is possible to specify the minimum buffer space equally, what helps low-speed flows. max_q on the other hand is responsible for limiting very fast flows. Finally, to manage unresponsiveness, flows with high $strike$ value cannot exceed the average per-flow queue length.

FRED is the only algorithm examined in this article that requires information on the queue size assigned to every flow.

2.4. Stabilized RED

Another modification of RED algorithm is the Stabilized RED [17]. In contrast to FRED, it does not require full per-flow state information, so its memory usage is lower. The idea is to estimate the number of active flows by maintaining a data structure called $zombielist$. Such structure contains information about M recently arrived packets with their $Count$ as well as $timestamps$. When the $zombielist$ is full, new packets are compared with a randomly chosen entry from the $zombielist$. If they belong to the same flow, the $Count$ variable is increased. Otherwise, the randomly chosen entry is replaced with certain probability p ("hit") and obviously the $zombielist$ is left unchanged with probability of $1 - p$ ("no hit"). This information allows to estimate the number of active flows as well as identify mis-behaving flows without maintaining all per-flow information. Using this information the probability of dropping incoming packet p_{zap} is calculated as follows:

$$p_{zap} = p_{sred}(q) \times min\left(1, \frac{1}{(256 \times P(t))^2}\right) \times \left(1 + \frac{Hit(t)}{P(t)}\right),$$

where:

$$p_{sred}(q) = \begin{cases} p_{max} & \text{if } \frac{1}{3}B \le q < B, \\ \frac{1}{4} \times p_{max} & \text{if } \frac{1}{6}B \le q < \frac{1}{3}B, \\ 0 & \text{if } 0 \le q < \frac{1}{6}B, \end{cases}$$

$$Hit(t) = \begin{cases} 0 & \text{if no hit,} \\ 1 & \text{if hit,} \end{cases}$$

$P(t)$ – the hit frequency at the time of t-th packet,
q – the current SRED queue length,
B – the buffer size.

SRED seems to be really good modification of RED scheme. The drawback of this approach is that it is assumed that all flows have the same intensity. However, the authors of SRED showed several points where the algorithm could possibly be improved.

2.5. Capture-Recapture

CARE algorithm [18] is another example of AQM which estimates the number of active flows. It uses M_h CR Model (see [22]) in conjunction with Jackknife estimator to compute the fair share and the actual sending rates. After these values are calculated, CARE can drop packets from flows which occupy more than the fair share value.

Because CARE algorithm performs a lot of calculations, it is not a good choice for fast network nodes. Moreover, to achieve best results CARE has to capture large amount of data. Having in mind that there are no methods employed for freeing up memory, the space complexity may also become a serious problem. The advantage of CARE is that it uses an original (taken directly from the biological science) idea for the estimation process. It is also possible that some optimizations may reduce its time and space complexity.

2.6. CHOKe

CHOKe algorithm [4] is also an extension of Random Early Detection scheme. It is basing on a simple but ingenious idea. Algorithm 1 presents the way of operation of CHOKe AQM. The most important part (not present in RED algorithm) is implemented in lines 5 to 8.

It can be easily noticed that unresponsive flows will usually have more packets in a standard FIFO queue than the others. Thus the probability of choosing a packet belonging to a misbehaving flow is greater. This is why the newly arrived packet is being compared with a random packet from the queue. If they both belong to the same flow they are dropped to penalize the unresponsive flow.

The drawback of the basic version of the algorithm is that its quality deteriorates with growing number of flows. To handle multiple flows, the authors propose choosing several drop candidates for one arriving packets, depending on how large the average queue size is. This makes the algorithm more powerful with not to much computation overhead.

2.7. Stochastic Fair Blue

SFB [21] main goal is to distinguish and penalize unresponsive flows. It uses a set of hash tables for identifying each flow. When a packet arrives, its flow identifier is hashed by all hash functions incrementing the appropriate element in all hash tables. The higher the element of some table is, the higher its dropping probability p_m gets. The

Algorithm 1 CHOKe

```
 1: Arriving packet $q$
 2: if $avgq \leq min_{th}$ then
 3:     Enqueue packet $q$
 4: else
 5:     Draw a random packet $q_{random}$ from queue
 6:     if $Flow(q) = Flow(q_{random})$ then
 7:         Drop both packets
 8:     else
 9:         if $avgq \leq max_{th}$ then
10:             Enqueue packet $q$ with probability $p$
11:         else
12:             Drop packet $q$
13:         end if
14:     end if
15: end if
```

final value of dropping probability for an incoming packet is the minimum value of p_m corresponding to that packet from all tables.

SFB can deal quite well with the fair bandwidth division, but again, its performance depends strongly on the number of flows, number of hash tables and hash function generation scheme. Generally, it will work properly for a small number of flows. With an increasing number of flows the probability of mis-identifying a responsive flow increases as well. However, with good selection of parameters, SFB may come with very satisfactory results.

3. TCP congestion control algorithms

The base TCP standard described in RFC793 does not provide any congestion control mechanism. To avoid the congestion collapse, [23], it was necessary to introduce modifications so that the transmission rate would be adjusted to the network condition. Beginning with the TCP Tahoe algorithm [24] which was the first solution employing well-known modes like *Slow Start, Congestion Avoidance* and *Fast Retransmit*, the TCP congestion control mechanism has evolved throughout the years and many different solutions have been proposed. In this article, we focus on seven TCP variants, namely: New Reno, Sack, Fack, Vegas, Westwood, H-TCP and Cubic. This set consists of algorithms that are actually implemented in modern operating systems and commonly used.

3.1. TCP New Reno and Sack

These are classic and most popular variants of the TCP protocol. Their properties and performance are well known and were described in a large number of papers and books (e.g. [25,26,27]).

3.2. TCP Vegas

TCP Vegas was proposed in [28] as an alternative to TCP Reno. The main difference is that Vegas uses the changing value of Round-Trip Time as an indication of congestion instead of a packet drop (as TCP Reno and its descendants do).

3.3. TCP Westwood

TCP Westwood [29], was proposed as a sender side modification to the TCP New Reno algorithm. It enables better bandwidth utilization by allowing faster recovery in large pipes. The sender maintains the available bandwidth estimate (*BWE*) and basing on this estimate sets the congestion window and slow start parameters.

3.4. TCP Fack

After defining the selective ACKs, another variant of TCP congestion control mechanism, called *Forward Acknowledgments*, has been proposed [30]. This approach uses the information about delivered packets to detect losses and to control transmission rate accordingly, without employing Reno's fast recovery.

To control the transmission rate Fack maintains three parameters: R – number of retransmitted packets, F – the forward-most received packet and H – the highest packet sent. These parameters can be used to approximate number of packets sent which are not yet delivered as $H - F + R$. Finally, this value is used by Fack algorithm to decide how much data can still be send.

Experiments have proved that Fack outperforms Reno and New Reno in recovery from errors. This is the reason why Fack algorithm is being widely used all over the world and has been included to the Linux kernel beginning with version 2.1.92. The Fack algorithm was the default Linux TCP until the kernel 2.6.19.

3.5. H-TCP

One of several high speed TCP variants is H-TCP [31]. It has been implemented in Linux 2.6 kernels as an optional module. In contrast to BIC or HS-TCP, after a packet loss, it increases its aggressiveness with time. This allows to utilize high-speed links and additionally increases the fairness somewhat. The latter effect is connected with the fact that the rate increase does not depend on window size what makes new flows achieve their fair bandwidth share faster.

3.6. TCP Cubic

Since the kernel 2.6.19, the Cubic algorithm [32] is the default Linux TCP variant. It is an extension of TCP BIC [33] which is supposed to improve fairness and simplify the BIC window control. As most high-speed implementations of TCP, Cubic modifies the well-known Tahoe window function with a function which utilizes the bandwidth faster. It uses a cubic function of time elapsed since last packet loss:

$$W_{cubic} = C(t-K)^3 + W_{max},$$

with

C – scaling factor,

t – real time elapsed since last window reduction,

W_{max} – window size before last reduction,

$K = \sqrt[3]{W_{max}\beta/C}$,

β – constant multiplication decrease factor.

This function was chosen because of its stability and scalability. Basing on real time instead of RTT makes the algorithm independent and TCP-friendly for both short and long RTT paths.

4. Experimental setup

All experiments described in this study were performed using ns-2 simulator ver. 2.34, on a dumb-bell topology with a single bottleneck link between two routers (see Fig. 1). The propagation delays N1-RA, . . . , RB-N6 are taken from the TCP evaluation suite [34]. The resulting delays of nine possible transmission paths, N1-N4, . . . , N3-N6, are supposed to mimic delay distribution in the Internet. Both routers A and B use the AQM scheme being investigated. All links in the topology have the capacity of 100Mb/s.

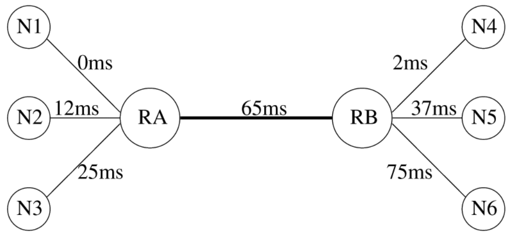

Fig. 1. Experimental topology with one-way propagation delays

Because we used AQMs, relatively small buffers (of the size of 10% of the Bandwidth-delay product) were used.

Three congestion scenarios, depending on the number of flows, were simulated:

- Uncongested link scenario: 10 flows,
- Mild congestion scenario: 100 flows,
- Heavy congestion scenario: 1000 flows.

As it was said, there are nine paths, N1-N4, N1-N5, …, N3-N6, in the presented topology. Now, all the simulated connections were uniformly distributed among them. It means that in the heavy congestion scenario, there were 111 flows on every path (one remaining flow was assigned randomly to one of the paths). Similarly, in the mild congestion scenario, there were 11 flows on every path, etc.

To simulate the real Internet traffic, different types of data transfers were simulated, including traffic characteristics induced by the application level (HTTP-like short flows and FTP-like long flows). Namely, in each simulation 90% of all flows were TCP flows sending 1500B packets. Furthermore, 25% of these were short-lived flows which were starting randomly according to the Poisson process with rate of 0.125, 1.25 and 12.5 in the heavy congestion, mild congestion and uncongested network scenario, respectively. Every such short flow was transferring a small file of Pareto-distributed size (with average avg=50kB and shape α=1.3). The rest of 1500-byte-packet flows were long-lived flows which transmitted continuously during the whole simulation time. Remaining 10% of all flows were TCP flows sending smaller, 536B, packets.

To make the simulation realistic, it is important to introduce the reverse traffic. Therefore, 10% of the total bandwidth available was used for the UDP (CBR) traffic in the backward direction.

The simulations lasted for 100s but the first 30s were used for stabilization. All measurements were taken from the 30-th second of the simulation.

All the parameters of the AQMs and TCP variants used were set for the default, recommended by their authors values. We did not manipulate these parameters at all. Naturally, it is possible to obtain better results in some scenarios by a proper tuning of some AQM and TCP parameters. However, such tuning often causes deterioration of the performance in other scenarios.

5. Results

Now the results can be presented and discussed. Firstly, the Jain's fairness index is presented in Tab. 1 (exceptionally good results are printed in green bold font, exceptionally bad results are printed in red italic font). We see at once that the best results were obtained using Cubic algorithm. Only a bit worse results are produced using H-TCP.

AQM algo-rithm	**Flows count**	**TCP New Reno**	**TCP Sack**	**TCP Fack**	**TCP Vegas**	**TCP H-TCP**	**TCP West-wood**	**TCP Cubic**
DT	10	0.88	0.82	0.85	*0.80*	0.88	*0.79*	**0.91**
	100	0.83	0.83	0.84	0.84	**0.88**	*0.71*	**0.88**
	1000	**0.84**	**0.84**	0.80	0.80	0.79	*0.76*	0.80
RED	10	0.83	0.87	0.84	*0.67*	**0.96**	*0.81*	**0.98**
	100	0.93	0.92	0.89	0.93	**0.98**	*0.87*	0.95
	1000	0.92	0.93	0.92	0.92	0.92	0.91	0.92
SRED	10	*0.75*	*0.75*	0.80	0.79	**0.93**	0.79	**0.92**
	100	0.89	0.87	0.89	0.87	0.92	0.86	0.92
	1000	0.88	0.89	0.86	0.87	0.89	0.87	0.86
FRED	10	0.81	0.84	**0.89**	0.83	0.82	*0.75*	0.84
	100	0.85	0.80	0.82	0.82	**0.86**	*0.72*	**0.86**
	1000	0.87	0.87	0.86	0.86	0.85	0.83	0.84
CHOKe	10	*0.81*	0.89	*0.82*	*0.79*	0.87	0.89	**0.96**
	100	0.89	0.88	0.86	0.89	**0.94**	*0.84*	**0.94**
	1000	0.94	0.93	0.91	0.93	0.93	*0.88*	0.93
SFB	10	0.88	0.86	0.82	0.86	**0.88**	*0.65*	**0.90**
	100	**0.85**	**0.83**	**0.83**	**0.83**	0.81	*0.54*	0.80
	1000	0.88	0.88	0.86	0.87	0.86	0.86	0.86
CARE	10	0.89	0.92	0.92	0.92	0.95	*0.86*	**0.96**
	100	0.94	0.93	0.94	0.94	0.93	*0.88*	0.94
	1000	0.84	0.84	0.81	0.83	0.80	0.79	0.82

Table 1: Fairness index (Jain's index). In green bold font – results at least four percent better than the average of the row. In red italic font – results at least four percent worse than the average of the row.

The advantage of these two TCPs is more visible in low and moderate congestion scenarios. The improvement offered by Cubic and H-TCP is significant. Even an increase of the fairness index by 5 percent is makes a difference, while we can observe an increase by as much as 31 percent in some cases (e.g. RED+Vegas vs RED+Cubic in the low congestion scenario).

The fairness can be also studied using detailed per-flow characteristics. Namely, in Figures 2-7 the throughput of every distinct flow is presented in six selected scenarios. In order to demonstrate the dependence on the RTT, the flows are sorted with respect to the path propagation delay, which varies from 0.067s to 0.165s. Naturally, the flatter each graph is, the fairer bandwidth division it represents. In Figs. 2 and 3 we can compare the fairness of the RED+New Reno configuration with the RED+Cubic configuration. We see at once, that the RED+Cubic case is fairer. What is more, in the RED+Cubic case the dependence of the per-flow throughput on the RTT is rather weak, while in the RED+New Reno we can see a strong, almost linear dependence. Similarly, in the mild congestion scenario (e.g. Figs. 4 and 5 for the CHOKe algorithm), applying Cubic

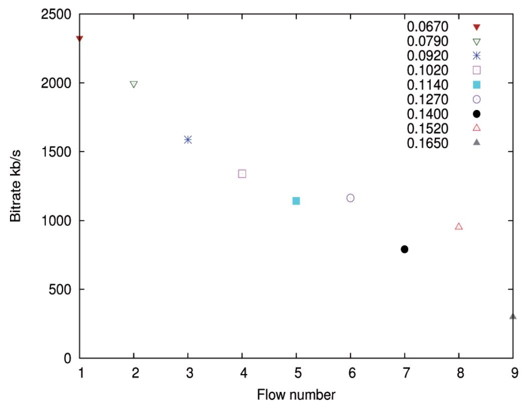

Fig. 2. The per-flow throughput in the RED+New Reno configuration in the uncongested scenario. The number assigned to each flow denote the propagation delay (in seconds) of the path that the flow used (in other words, it is RTT/2)

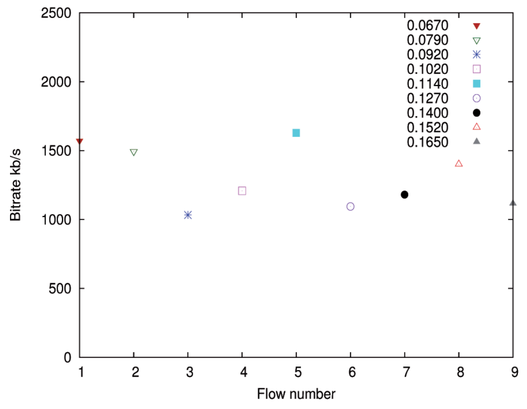

Fig. 3. The per-flow throughput in the RED+Cubic configuration in the uncongested scenario

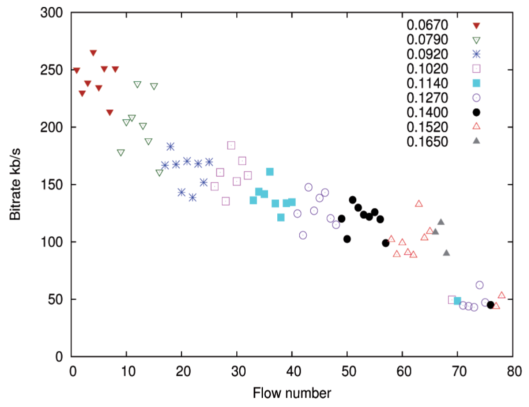

Fig. 4. The per-flow throughput in the CHOKe+Fack configuration in the mild congestion scenario

improves the fairness and reduces the dependence on the RTT. However, in the heavy congestion scenario such an improvement is not observed (compare Figs. 6 and 7 for the FRED queueing).

Finally, note that from Tab. 1 it follows that the three most fair configurations are RED+H-TCP, RED+Cubic and CHOKe+Cubic.

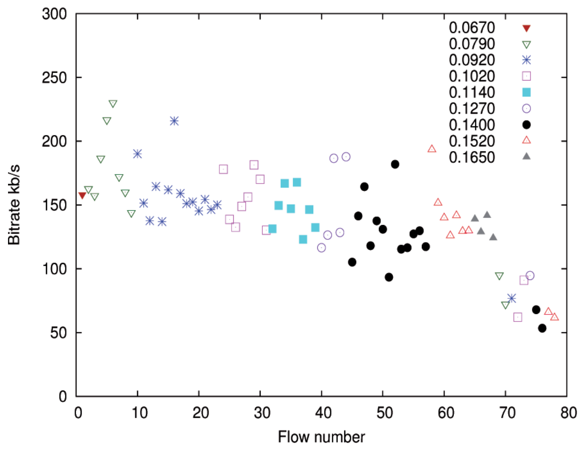

Fig. 5. The per-flow throughput in the CHOKe+Cubic configuration in the mild congestion scenario

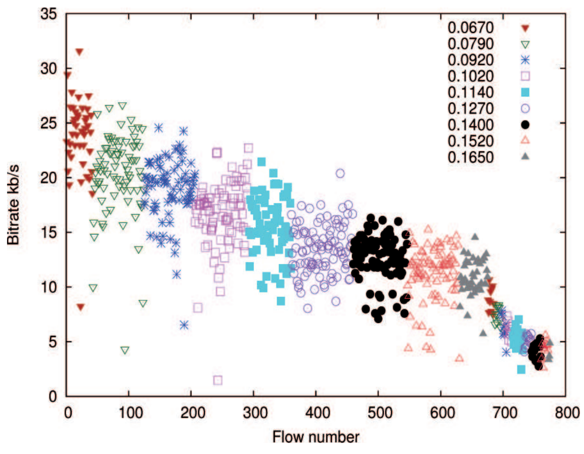

Fig. 6. The per-flow throughput in the FRED+Sack configuration in the heavy congestion scenario

Now we should check other than the fairness index characteristics. In Tab. 2 the total throughput achieved at the bottleneck link is shown. Generally speaking, configurations with H-TCP, Westwood and Cubic perform significantly better than others. For low and mild congestion levels, this improvement can sometimes be dramatic, compared with New Reno (e.g. the throughput can be doubled, like in the SRED+Cubic configuration

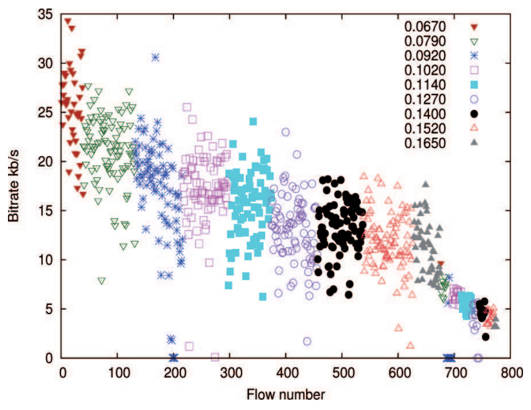

Fig. 7. The per-flow throughput in the FRED+Cubic configuration in the heavy congestion scenario

vs SRED+New Reno configuration). This effect can be further observed in Figs. 8 and 9. We see that by applying Cubic not only we improve the fairness (flatter graph) but also the total throughput (the points are higher on the graph).

Taking only the total throughput into account, we see that the best configurations are: CHOKe+Westwood, RED+Westwood and CHOKe+Cubic. If we compare this with three most fair configurations, i.e. RED+H-TCP, RED+Cubic and CHOKe+Cubic, we see that the only configuration that provides fair bandwidth share and high bandwidth utilization at the same time is CHOKe+Cubic. Therefore it can be recommended as the best configuration among all considered in this paper.

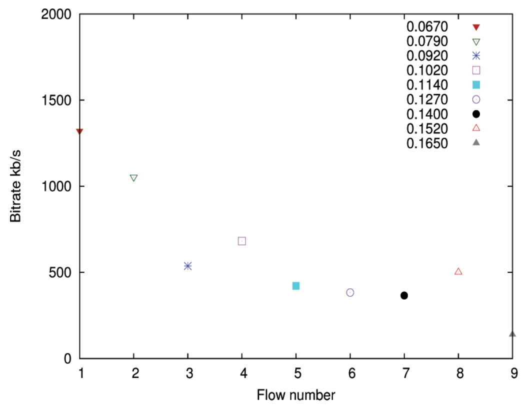

Fig. 8. The per-flow throughput in the SRED+Sack configuration in the uncongested scenario

This result is most likely caused by the fact that both algorithms, Cubic and CHOKe have built-in mechanisms supporting fairness and high bandwidth utilization at the same time. Firstly, due to usage by Cubic of the real time instead of send-ACK cycles, the

AQM algo-rithm	Flows count	TCP New Reno	TCP Sack	TCP Fack	TCP Vegas	TCP H-TCP	TCP West-wood	TCP Cubic
DT	10	*37.23*	*52.55*	*46.48*	*51.97*	**96.59**	**73.03**	**86.43**
	100	93.22	96.15	96.45	96.78	94.73	99.98	99.48
	1000	99.98	99.96	99.97	99.97	99.99	99.99	99.98
RED	10	98.00	98.58	97.80	99.88	*90.96*	99.97	99.50
	100	98.18	98.28	98.95	99.92	96.10	99.95	*92.78*
	1000	99.97	99.87	99.71	99.15	99.78	99.48	99.95
SRED	10	*42.34*	*48.52*	*52.38*	65.32	**70.05**	**95.26**	**86.30**
	100	*85.25*	86.70	*85.68*	86.66	92.87	**99.85**	**93.81**
	1000	99.97	99.91	99.31	99.64	99.90	99.81	99.97
FRED	10	*17.40*	*21.30*	*20.79*	*28.42*	**55.26**	**72.01**	**54.68**
	100	*76.25*	*79.05*	*80.37*	85.89	**92.28**	**97.41**	**92.59**
	1000	99.38	99.16	99.07	99.08	99.04	98.83	99.46
CHOKe	10	91.17	*86.96*	**96.79**	93.14	*83.33*	**99.92**	**97.66**
	100	98.28	98.53	98.97	99.66	96.25	99.96	99.76
	1000	99.95	99.95	99.79	99.99	99.99	99.98	99.97
SFB	10	*9.18*	*10.20*	*10.91*	*12.41*	**29.27**	**47.28**	**22.06**
	100	*49.50*	*48.89*	*51.22*	*52.08*	**69.66**	**74.70**	**69.45**
	1000	99.56	98.35	96.60	96.79	98.07	97.34	99.06
CARE	10	*39.92*	*47.34*	*49.75*	*47.11*	**63.53**	**79.00**	**64.35**
	100	*84.89*	88.91	89.46	89.91	93.58	**99.97**	**96.63**
	1000	99.98	99.97	99.97	100.00	99.98	99.97	99.98

Table 2: Throughput [Mb/s]. In green bold font – results at least four percent better than the average of the row. In red italic font – results at least four percent worse than the average of the row.

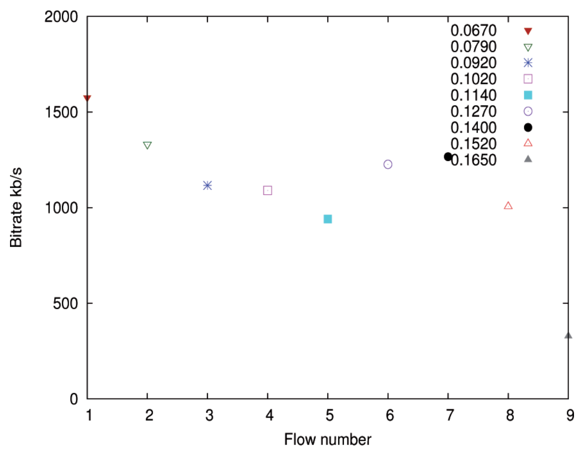

Fig. 9. The per-flow throughput in the SRED+Cubic configuration in the uncongested scenario

AQM algorithm	Flows count	TCP New Reno	TCP Sack	TCP Fack	TCP Vegas	TCP H-TCP	TCP Westwood	TCP Cubic
DT	10	1.5	2.6	2.1	1.6	7.8	30.1	10.4
	100	25.2	31.9	32.4	30.0	39.1	74.3	50.9
	1000	82.7	81.1	79.0	78.8	80.9	80.1	82.7
RED	10	54.8	58.7	54.7	36.2	107.1	76.9	76.5
	100	97.3	99.1	97.2	98.2	103.7	98.6	98.1
	1000	102.0	98.0	99.8	99.3	98.5	97.6	98.7
SRED	10	1.8	1.8	2.5	2.7	27.8	26.9	14.0
	100	10.3	10.8	10.6	8.1	25.9	37.0	15.3
	1000	45.4	40.1	28.3	30.8	46.1	34.8	32.1
FRED	10	0.1	0.2	0.2	0.3	1.8	3.0	1.4
	100	5.7	6.5	7.1	6.7	12.3	16.6	10.4
	1000	26.4	26.0	25.1	24.9	25.6	24.9	25.6
CHOKe	10	28.7	22.7	32.7	21.9	23.6	73.7	48.7
	100	72.0	70.0	75.4	79.5	69.7	85.9	83.0
	1000	101.0	98.7	91.3	98.9	99.8	99.3	100.0
SFB	10	0.03	0.03	0.04	0.03	0.24	0.52	0.1
	100	0.7	0.8	0.9	0.8	1.9	2.4	1.7
	1000	16.4	12.4	9.1	9.2	10.7	9.6	11.7
CARE	10	1.8	2.1	2.7	1.2	7.4	14.7	5.6
	100	14.9	19.2	20.7	14.0	38.7	77.0	36.6
	1000	93.9	92.2	89.8	90.4	91.3	90.7	93.8

Table 3. Mean queue size [pkts].

algorithm is fairer than other TCPs. In the Reno algorithm and its descendants, the congestion window of a connection with long RTT increases very slowly, which causes bandwidth underutilization. This problem is removed in the Cubic algorithm, as the congestion window grows with the same speed for connections with different RTTs. This positive effect is then strengthen by the CHOKe mechanism presented in lines 5-8 of Algorithm 1, which further reduces any remaining unfairness. Secondly, Cubic has the well-known ability for high bandwidth utilization. This is connected with the shape of the cubic function used for congestion window expansion after a packet loss. The cubic window function consists of a quick phase of recovery from the window reduction, a stabilization phase, and a quick phase of probing for new available bandwidth. CHOKe on the other hand has built-in RED algorithm, which supports high bandwidth utilization by desynchronization of TCP sources and preventing consecutive window reductions due to buffer overflows.

As for the queue size at the bottleneck link, the average results are given in Tab. 3, while sample queue length processes for the CHOKe+Sack and CHOKe+Cubic are depicted in Figs. 10 and 11, respectively. In Tab. 3 we can see that the configurations with

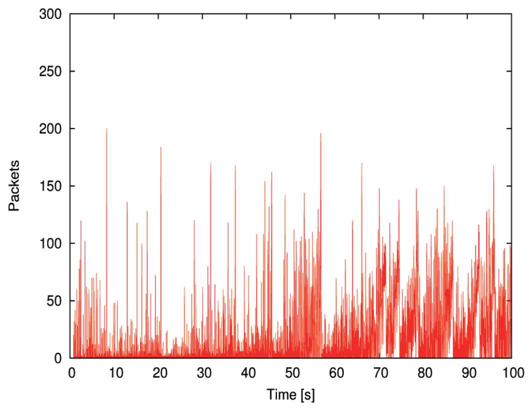

Fig. 10. The queue size evolution in the CHOK+Sack configuration, uncongested scenario

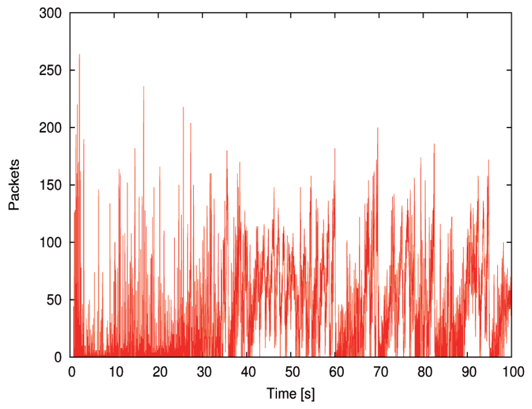

Fig. 11. The queue size evolution in the CHOK+Cubic configuration, uncongested scenario

Cubic algorithm induce usually higher, than in New Reno case, queue sizes (but typically shorter than in Westwood case). However, in all cases, this change is not important from the practical point of view. For instance, the increase of the queue size from 28 to 48 packets (CHOKe+New Reno vs CHOKe+Cubic) is not significant, because both values represent low queue sizes anyway. Moreover, when the number of flows grows, we observe that the queue size is more or less the same for every AQM used.

6. Conclusions

In this paper we presented a study on the impact of seven TCP variants on the performance of fairness-driven queue management algorithms.

The good news is that almost all studied AQMs perform well in the presence of the newest Cubic TCP, which is becoming popular in the Internet. In fact, although not designed purposely to cooperate with it, the AQMs work much better when used with Cubic than when used with other TCPs. In particular, the fairness index and the overall throughput are improved at the cost of minor increase of the average queue size. Moreover, applying Cubic reduces the dependence of the per-flow throughput on the flow's RTT.

Taking into account the fairness index and achieved throughput the best cooperation between network and transport layer was obtained in the CHOKe+Cubic configuration.

Acknowledgements

This work was supported in part by the Polish Ministry of Scientific Research and Information Technology under Grant No. N N516 381134.

References

1. C. Caini, R. Firrincieli: *TCP Hybla: a TCP Enhancement for Heterogeneous Networks*. International Journal of Satellite Communications and Networking, Volume: 22, Number: 5, pp. 547-566, 2004.
2. I. Rhee, L. Xu: CUBIC: *A New TCP-Friendly High-Speed TCP Variant*, SIGOPS Operating Systems Review vol. 42 (5), pp. 64-74, July 2008.
3. G. Chatranon, M.A. Labrador, S. Banerjee: *A survey of TCP-friendly router-based AQM schemes*, Computer Communications 27, pp. 1424–1440, 2004.
4. R. Pan, B. Prabhakar, K. Psounis: *CHOKe — A Stateless Active Queue Management Scheme For Approximating Fair Bandwidth Allocation*, Proceedings of IEEE INFOCOM 2000, vol. 2, pp. 942-951.
5. Z. Rosberg, J. Matthews, M. Zukerman: *A network rate management protocol with TCP congestion control and fairness for all*, Computer Networks, vol. 54, issue 9, pp. 1358-1374, 2010.
6. Z. Rosberg: *Control Plane for End-to-End QoS Guarantee: A Theory and Its Application*, Proc. IWQoS'08, pp. 269–278, 2008.
7. Z. Rosberg, C. Russell, V. Sivaraman: *Rate and End-to-End Delay Control for Multicast and Unicast Flows*, IEEE International Conference on Communications, pp. 1-6, 2009.
8. Z. Rosberg, F. Sabrina, S. Dealy, J. Matthews, C. Russell: *Rate and delay controlled core networks: an experimental demonstration*, in: Proceedings of IWQoS'09, Charelston, S. Caraolina, 2009.
9. A. Chydzinski, A. Brachman: *Performance of AQM Routers in the Presence of New TCP Variants*, Proc. of International Conference on Advances in Future Internet, pp. 88–93, Venice, July 2010.

10. Mohammad Abu Obaida, Md. Sanaullah Miah, Md. Abu Horaira: *Random Early Discard (RED-AQM) Performance Analysis in Terms of TCP Variants and Network Parameters: Instability in High-Bandwidth-Delay Network*, International Journal of Computer Applications, Vol. 27, No.8, pp.40-44, August 2011.
11. T. Bonald, M. May, J.C. Bolot: *Analytic evaluation of RED performance*, Proc. 19'th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1415–1424, 2002.
12. W. Hao, Y. Wei: *An extended GIX/M/1/N queueing model for evaluating the performance of AQM algorithms with aggregate traffic*, vol. 3619 of Lecture Notes in Computer Science, pp. 395–404, 2005.
13. A. Chydzinski, L. Chrost: *Analysis of AQM queues with queue-size based packet dropping*, International Journal of Applied Mathematics and Computer Science, Vol. 21, No. 3, pp. 567–577, 2011.
14. D. Lin, R. Morris: *Dynamics of Random Early Detection*, Proceedings of SIGCOMM, pp. 127-137, 1997.
15. B. Suter, T. Lakshman, D. Stiliadis, A. Choudhury: *Design Considerations for Supporting TCP with Per-Flow Queuing*, Proceedings of IEEE INFOCOM, pp. 299-306, 1998.
16. F. Anjum, L. Tassiulas: *Fair Bandwidth Sharing Among Adaptive and Non-Adaptive Flows in the Internet*, Proceedings of IEEE INFOCOM, pp. 1412-1420, 1999.
17. T. Ott, T. Lakshman, L. Wong(Eds.): *SRED Stabilized RED*, IEEE INFOCOM '99 vol. 3, pp. 1346-1355, 1999.
18. M. Chan, M. Hamdi: *An Active Queue Management Scheme Based on a Capture–Recaptured Model*, IEEE Journal on Selected Areas in Communication 21 (4), pp. 572-583, 2003.
19. G. Chatranon, M.A. Labrador, S. Banerjee: *BLACK: Detection and Preferential Dropping of High Bandwidth Unresponsive Flows*, Proceedings of IEEE ICC vol. 1, pp. 664-668, 2003.
20. S. Floyd, V. Jacobson: *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking vol. 1 (4), pp. 397-413, 1993.
21. W. Feng, D. Kandlur, D. Saha, K. Shin: *Stochastic Fair Blue A Queue Management Algorithm for Enforcing Fairness*, Proceedings of IEEE INFOCOM 2001, vol. 3, pp. 1520-1529.
22. G.C. White, D.R. Anderson, K.P. Burnham, D.L. Otis: *Capture–Recapture and Removal Methods for Sampling Closed Populations*, Los Alamos National Laboratory LA-8787-NERP, pp. 235, 1982.
23. C.A. Kentand, J.C. Mogul: *Fragmentation considered harmful*, Proceedings of the ACM workshop on Frontiers in computer communications technology (SIGCOMM), Stowe, Vermont, pp. 390-401, August 1987.
24. V. Jacobson: *Congestion avoidance and control*, ACM SIGCOMM vol.25 (1), pp. 157-187, 1995.

25. S. Floyd, T. Henderson: *The New-Reno Modification to TCP's Fast Recovery Algorithm*, RFC 2582, Apr 1999.
26. M. Mathis, J. Mahdavi, S. Floyd, A. Romanov: *TCP selective acknowledgment options*, RFC 2018, 1996.
27. A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock: *Host-to-Host Congestion Control for TCP*, IEEE Communications surveys & tutorials vol.12 (3), pp. 304-342, 2010.
28. L.S. Brakmo, S.W. O'Malley, L.L. Peterson: *TCP Vegas: New techniques for congestion detection and avoidance*, ACM SIGCOMM Computer Communication Review, vol. 24, pp. 24-35, 1994.
29. C. Casetti, M. Geria, S.S. Lee, S. Mascolo, M. Sanadidi: *TCP with faster recovery*, MILCOM 2000. 21st Century Military Communications Conference Proceedings, pp. 320-324, 2000.
30. M. Mathis, J. Mahdavi: *Forward acknowledgment: refining TCP congestion control*, Proc. conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), NewYork, USA, pp. 281-191, 1996.
31. D. Leith and R. Shorten: *H-TCP: TCP for high-speed and long-distance networks*, Proc. of the 2nd Workshop on Protocols for Fast Long Distance Networks, 2004.
32. S. Ha, I. Rhee, L. Xu: *CUBIC: a new TCP-friendly high-speed TCP variant*. ACM SIGOPS Operating Systems Review, vol. 42, issue 5, pp. 64-74, 2008.
33. L. Xu, K. Harfoush, I. Rhee: *Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks*, Proceedings of IEEE INFOCOM, vol. 4, pp. 2514-2524, 2004.
34. L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, I. Rhee: *Towards a Common TCP Evaluation Suite*, Proc. International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Manchester, 2008.

O poprawie sprawiedliwości podziału pasma pomiędzy przepływy w Internecie

Streszczenie

Wszystkie znane z literatury algorytmy aktywnego zarządzania kolejkami (AQM) ukierunkowane na zapewnienie sprawiedliwego podziału pasma pomiędzy przepływy w Internecie były opracowywane z myślą o współpracy z (klasycznym dziś) algorytmem kontroli zatłoczenia TCP, tzn. New Reno. W ostatnich latach można zauważyć w Internecie istotne zwiększanie się udziału nowych algorytmów kontroli zatłoczenia TCP (jak np. algorytmu Cubic). Dlatego też pojawia się naturalne pytanie: czy algorytmy sprawiedliwego podziału pasma zaprojektowane dla New Reno będą równie dobrze działać w obecności tych nowych wariantów TCP?

Aby uzyskać odpowiedź na to pytanie, przeprowadzone zostały szeroko zakrojone studia symulacyjne. Studia te uwzględniały siedem najważniejszych algorytmów AQM

do sprawiedliwego podziału pasma, siedem wersji TCP (w tym najnowocześniejsze warianty), różne scenariusze zatłoczenia sieci oraz czasy RTT połączeń TCP. Uzyskane wyniki pozwoliły nie tylko udzielić odpowiedzi na sformułowane powyżej pytanie, ale także wskazać, które wersje TCP i AQM najlepiej realizują ideę optymalizacji międzywarstwowej w celu zapewnienia sprawiedliwego podziału pasma pomiędzy przepływy w Internecie.