Lightweight Fairness Solutions for XCP and TCP Cohabitation

Dino M. López Pacheco[1,2], Laurent Lefèvre[2], and Congduc Pham[3]

[1] CONACyT
[2] INRIA RESO / Université de Lyon / ENSL, France,
[3] LIUPPA Laboratory, Université de Pau, France
{dmlopezp,lefevre.laurent}@ens-lyon.fr,
congduc.pham@univ-pau.fr

Abstract. XCP is a promising router-assisted protocol due to its high performance and intra-protocol fairness in fully XCP networks. However, XCP is not inter-operable with current E2E protocols resulting in a poor performance of XCP flows when the resources are shared with TCP-like flows. In this paper, we propose a lightweight fairness solution, that can benefit the cohabitation between XCP and TCP in large *bandwidth x delay* product networks and long-life flows scenarios, as it is shown in our simulation results.

Key words: TCP, End-to-End protocols, XCP, ERN protocols, inter-protocol fairness, inter-operability.

1 Introduction

End-to-End (E2E) protocols are currently the most deployed protocols to control congestion in the networks and provide fair share of resources between users. One of the reasons because E2E protocols are widely deployed is that most of their mechanisms are generally implemented only in the senders side. Thus, E2E protocols are totally independent from the network infrastructure.

However, E2E protocols have limited performance. For instance, TCP [14], [3], which is currently the most used congestion control protocol, produces frequently dropped packets and is unable to provide fairness when users have different delay [1]. In addition, TCP have poor performance in networks with large capacity and delay [15]. Others proposed E2E protocols especially designed to get high performance in large *bandwidth x delay* product (BDP) networks, like High Speed TCP [15] or Scalable TCP [6] suffers intra-protocol and inter-protocols unfairness [16].

Since E2E protocols have revealed unable to solve the problems of congestion and fairness, a new family of congestion control protocols has been proposed. This new protocols use the assistance from routers to provide an accurate information about the network conditions to the senders, that is why they are known like routers-assisted protocols or Explicit Rate Notification (ERN) protocols.

XCP [5], which is one of the most famous of ERN protocols, is a very promising protocol since XCP shows very stable behavior, high performance and intra-protocol fairness. However, XCP (as others ERN protocols) (*i*) requires the collaboration of all the routers on the data path, which is almost impossible to achieve in an incremental deployment scenario of XCP and (*ii*) does not have any mechanism to fairly share the resources with E2E protocols (e.g. TCP).

Concerning the interoperability problems of XCP with heterogeneous network equipments, we have already proposed a solution to this problem in [9] (the *XCP-i* module). On the other hand, in [8] we presented a short description about a solution to ensure the XCP-TCP fairness, that we implemented and tested on the XCP protocol, but that can be easily applied to any other ERN protocol. In that occasion, we dedicated specially the space to show graphically that our XCP-TCP fairness solutions is able to ensure a max-min fairness solution between XCP and TCP in a wide range of scenarios. Now, in this article, our main interest is to present a detailed description of our lightweight XCP-TCP fairness solution as well as a widely discussion about a few simulation results obtained by mean of the *ns*2 simulator [12]. We profit also to present some arguments to reinforce the "lightweight" character of our XCP-TCP fairness solution.

This paper is organized as follows: Section 2 provides a quick description of the XCP protocol and Section 3 describes the XCP-TCP unfairness problem. Section 4 presents a detailed description of our XCP-TCP fairness solution, while in Section 5 we validate our solution by simulation. Section 6 presents an improvements of our XCP-TCP fairness mechanism to make it lightweight in terms of CPU and memory. Finally, in Section 7 we provide concluding remarks.

2 The XCP Protocol

XCP [5] (eXplicit Control Protocol) uses router-assistance to accurately inform the sender of the network conditions. In XCP, data packets carry a congestion header, filled in by the source, that contains the sender's current congestion window size (`H_cwnd`), the estimated RTT (`H_rtt`) and a feedback field (`H_feedback`). The `H_feedback` field is the only one which could be modified at every XCP router based on the value of the two previous fields. Basically, the `H_feedback` represents the amount by which the sender's congestion window size is increased (positive feedback) or decreased (negative feedback).

The core mechanism of XCP resides in XCP routers. For every incoming packet, an XCP router needs to access the IP header, to get the packet size, the packet type, and the congestion control protocol used. When the congestion control protocol used is XCP, and the incoming packet is not an ACK, then, the XCP router must get the *H_cwnd*, *H_rtt* and *H_feedback* values. With the informations collected, XCP routers will execute two control laws in order to calculate a per packet feedback:

1. The Efficiency Controller (EC), that maximizes the link utilization while minimizing losses of packets.

2. The Fairness Controller (FC), that assigns resources fairly between XCP flows.

To maximize the link utilization, the Efficiency Controller computes a value named feedback, denoted by the symbol ϕ, that reflects the available bandwidth:

$$\phi = \alpha.rtt.(O - I) - \beta.Q \tag{1}$$

where α and β are constant whit values 0.4 and 0.226 respectively. rtt is the average RTT of all the packets crossing the router, O the output link capacity, I the input traffic rate seen by the XCP router during a control interval and Q, the persistent queue size. Note that in the feedback equation, ϕ decreases as the input traffic rate increases, and the available bandwidth decreases.

Later on, the Fairness Controller translates the computed general feedback, ϕ, in a feedback per packet, that will assign the same amount of bandwidth to each flow. The feedback per packet, computed by the FC, will replace the *H_feedback* value carried in the data packet if the computed feedback is smaller than the one carried in the header. Thus, for every data packet arriving to the XCP receiver, (i) the receiver will copy the $H\_feedback$ value into the ACK packets to let the sender update its congestion window size, if we are using the XCP standard protocol, or (ii) if we are using XCP-r (as described in [10]), the receiver will calculate the new congestion window size with the $H\_feedback$, and that size will be sent to the sender back. The way to update the congestion window either in the sender or in the receiver is

$$cwnd = max(cwnd + H\_feedback, packetsize) \tag{2}$$

3 Fairness Issues Between XCP and TCP

In order to maximize link utilization and avoid congestion, the Efficiency Controller computes the available bandwidth S (the *spare bandwidth*) as $S = O - I$. However, the spare bandwidth equation does not have any mechanism to differentiate between traffic generated by XCP and non-XCP flows. Hence, the input traffic rate could be generated by any TCP flow, forcing XCP flows to take only the remaining available bandwidth. Thus, XCP routers are not able to decrease the sending rate of non-XCP flows, like TCP flows.

Using the topology shown in Figure 1, we analyzed the behavior of one XCP flow when competing with two TCP flows by mean of a simulation. The results can be found in Figure 2, where we plotted the throughput of the XCP flow (continuous line) and the throughput of both TCP flows (dashed lines).

As we can see in Figure 2, the XCP flow starting at second 0 takes all the available bandwidth while it does not share the resources with concurrent TCP flows. However, at second 10 when two TCP flows appear in the scenario, the performance of the XCP flow is strongly degraded. In fact, after second 19, the XCP flow has almost disappeared.

In Figure 2, between seconds 15 and 19, it seems that the XCP flow tries to get some bandwidth but without success. The explanation of this phenomenon is

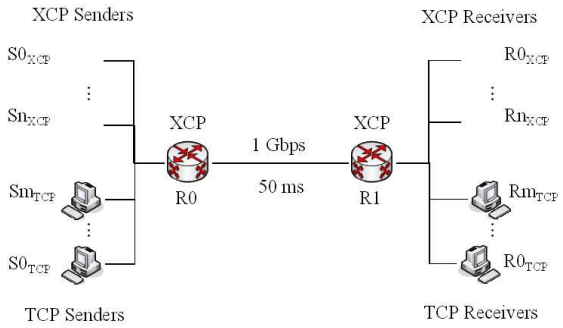


Fig. 1. Topology: n XCP and m TCP flows sharing the bottleneck.

as follows: Since the congestion window size of both TCP flows are not yet large enough to keep a continuous transference of packets, during some milliseconds the XCP router placed in the bottleneck detects a positive difference between the output link capacity and the input traffic rate. This very small remaining bandwidth is assigned to the XCP sender, which has still the opportunity of transferring some data packets.

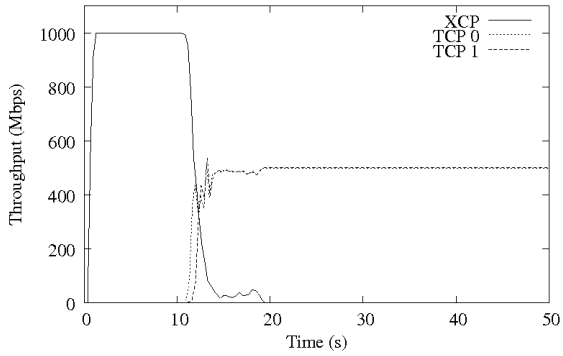


Fig. 2. XCP flow dealing with TCP concurrent flows.

4 Proposition for an Inter-Protocol XCP-TCP Fairness

4.1 Definition of XCP-TCP Fairness & Goals of Our Fairness Mechanism

To consider that TCP is fair with XCP, XCP must get a bandwidth equivalent to the ratio between the output link capacity O, and the sum of the number of

XCP and TCP flows, multiplied by the number of XCP flows. Equation 3 shows this relationship.

$$BW_{XCP} = \frac{O}{N+M} * N \tag{3}$$

In Equation 3, BW_{XCP} is the bandwidth needed by XCP when the bottleneck is shared by XCP and TCP flows, N is the number of active XCP flows and M the number of active TCP flows. Thus, we can estimate the limit in terms of bandwidth for TCP, BW_{TCP}, as a function of BW_{XCP}:

$$BW_{TCP} = O - BW_{XCP} \tag{4}$$

4.2 Estimating the Resources Needed by XCP

In order to fairly share the resources between XCP and TCP, from Equation 3, we can deduct that it is necessary to calculate the number of XCP and TCP active flows. Calculating the number of active flows crossing a router is not easy. In this article we describe two mechanisms to estimate the number of active flows that we consider as the more importants: the Bloom filter algorithm [2] and the SRED's zombie estimator [13].

The Bloom filter algorithm [2], as it has been implemented by NRED [7], works as follows:

When a packet arrives, the NRED's Bloom filter algorithm hashes the source-destination pair of the packet into a bin. A bin is a 1 bit memory to mark 0 or 1. The estimator maintains $P \times L$ bins. The bins are organized in L levels, each of which contains P bins. The estimator also uses L independent hash functions, each of which is associated with one level of bins, that maps a flow into one of the P bins in that level. For every arriving packet, the L hash functions can be executed in parallel. At the beginning of measurement interval, all bins and a counter N_{act} are set to zero. When a packet arrives at the router, the L hashed bins for the source-destination IP address pair of the packet are set to 1 and N_{act} increases in one unit if at least one of the L hashed bins is zero before hashing. However, it could arrive that for packets belonging to two different flows are hashed into the same bins of L levels causing a "misclassification". The authors claim that the probability of having a "misclassification" decreases as the number of L hash functions increases.

The problem of the Bloom filter algorithm is based on the fact that to avoid "misclassification", an important number of hash functions is needed, and since every hash function is executed one time for every incoming packet, this algorithm could be expensive in terms of CPU requirements. In addition, even if the hash functions can be performed in parallel, those hash functions must synchronize in order to update N_{act}, becoming this algorithm expensive in terms of time.

On the other hand, the *zombie estimator* is an algorithm used in SRED [13]. The zombie estimator keeps a table called *zombie*, able to keep up to 1000 ID flows (the ID flow could be src_addr:src_port::dest_addr:dest_port). The zombie

table is filled in at the beginning with the ID flows belonging to the first 1000 incoming packets.

When the zombie table has been filled in, every arriving packet to the router is examined. First, the ID flow of the packet is taken, and compared with an ID flow taken randomly from the zombie table. If the IDs are equal, an event *hit* is declared. Otherwise, an event *mis* is declared. When a *mis* is detected, with a probability of 25%, the old stored ID in the zombie table will be replaced by the new flow ID. Later on, in both cases *hit* or *mis*, a variable $P(t)$, that keeps the probability to make a *hit* is updated as follows:

$$P(t) = (1 - \delta)P(t - 1) + \delta.Hit(t) \tag{5}$$

where δ reflects the probability to get a packet from the zombie table with the same ID of the incoming packet ($\delta = 0.25 * (1/1000) = 0.00025$). It should be noted that α from the P(t) original equation has been replaced by δ to avoid confusions with the α XCP parameter. $Hit(t)$ is a constant with value 1 in case of *hit*, or 0 otherwise.

If the total of active flows in a router is N, then the probability to get a *hit* is $1/N$. Since the probability to get a *hit* is already contained in $P(t)$, hence $1/P(t)$ represent the estimation of the number of active flow seen by a router at time t.

The operations executed by the zombie estimator do not require a big utilization of CPU, since this estimator only needs one comparison, to generate two random numbers, and in some cases one writing, for every incoming packet. However, one of the weakness of the zombie estimator lies on the fact that it does not calculate the exact number of active flows, but only tries to estimate this number.

We believe that with current technologies it is very difficult to know the exact number of active flows during a given period (we need faster CPU and bigger memory than currently available in routers). We believe also that having an estimation of the number of active flows is enough to ensure a max-min fairness between XCP and TCP. For this reason, we have used the *zombie estimator* to compute the number of active TCP and XCP flows.

Note that even though a zombie table with 1000 slots could not accurately estimate a high number of flows (e.g., 10,000 active flows), a zombie table with such a size can give us a close idea about the real behavior of the estimation method. In a real environment we can use a larger zombie table, to accurately estimate higher numbers of active flows, without introducing significantly CPU operations.

The way to implement the zombie estimator in the XCP routers is very similar to the way proposed in SRED. However, in our case we will have one variable $P(t)_{XCP}$ that will keep the probability to get an XCP_{hit}, and another one $P(t)_{TCP}$ that will keep the probability to get a TCP_{hit}. Thus, by applying the described zombie estimator, we can have an idea about the resources needed by the XCP flows BW_{XCP}. We have proposed to estimate the resources needed by XCP and TCP at every control interval of the XCP router.

4.3 Ensuring the XCP-TCP Fairness

In order to ensure the fairness between XCP and TCP, it is necessary to compute also the real amount of resources taken by XCP. Thus, by comparing both the needed and the real amount of resources taken by XCP, we will be able to make a decision to improve the fairness.

Computing the Real Amount of Resources Taken by XCP. As we said in section 2, for every incoming packet, XCP routers must compute a per packet feedback by computing an input traffic rate (*in_traffic_rate*) at the end of every control interval (*ctrl_int*). This corresponds to the average RTT signaled in every incoming data packet. The *in_traffic_rate* variable is calculated by dividing the total amount of data received during a control interval (*in_traffic*), by the duration of such control interval. *in_traffic* is calculated by adding the packet size of every incoming (data or ACK) packet.

Since the XCP routers already implement a procedure to compute the input traffic rate, to estimate the XCP input traffic rate requires only minor changes in the XCP mechanism. Thus, all that we need to do is to add a new variable, *xcp_in_traffic*, that will store the sum of the size of every XCP packet, and divide this variable by the duration of the control interval *ctrl_int*, to get the XCP input traffic rate *xcp_in_traffic_rate*.

Ensuring Fairness Between XCP and TCP Flows. Once calculated the XCP input traffic rate, *xcp_in_traffic_rate*, and the needed XCP bandwidth, BW_{XCP}, we can make decisions to provide fairness between XCP and TCP. In order to limit the TCP throughput and get fairness, with a p_{drop} probability, our XCP-TCP fairness will determine if an incoming TCP packet should be discarded. The p_{drop}, which is initialized with a value of 0.001, is updated at every control interval as follows:

if $xcp\_in\_traffic\_rate > BW_{XCP}$, the probability p_{drop} is updated as $p_{drop} = p_{drop} * D_{drop}$, where $0.99 < D_{drop} < 1$. If $xcp\_in\_traffic\_rate < BW_{XCP}$, the probability p_{drop} is updated as $p_{drop} = p_{drop} * I_{drop}$, where $1.01 > I_{drop} > 1$. When $xcp\_in\_traffic\_rate = BW_{XCP}$, then p_{drop} conserves its last value.

4.4 Specificities About our XCP-TCP Fairness Mechanism

It is very important to emphasize that the XCP routers will execute the XCP-TCP fairness mechanism only (*i*) when both XCP and TCP protocols have been detected during a control interval, and (*ii*) if the total input traffic rate exceeds a threshold γ.

We will not execute our XCP-TCP fairness mechanism when the input traffic rate is smaller than γ, since it means that the current router is not the bottleneck. Concerning the γ parameter, we advise to set γ to a value slightly smaller than the output link capacity (e.g., 97% of the output link capacity), since even in a bottleneck router, the computed rate during a control interval is not always

equal to the output link capacity, due to the burstiness nature of the senders. Thus, during a certain control interval we could compute an input traffic rate slightly smaller than the output link capacity, and in the next control interval, see the router dropping packets.

Finally, when the total input traffic rate becomes smaller than our γ threshold, the p_{drop} variable will be reinitialized to the original value (0.001).

5 Validating our XCP-TCP Fairness Solution

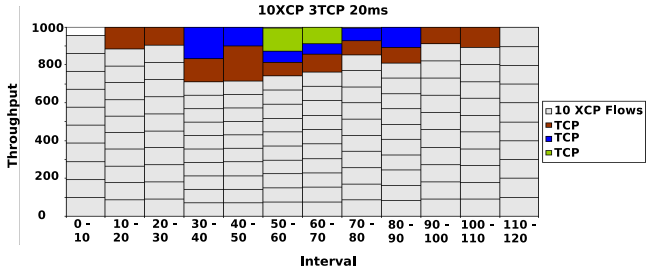
In order to test our XCP-TCP fairness, we implemented our propositions in the ns2 XCP modules provided by D. Katabi (`http://www.ana.lcs.mit.edu/dina/XCP/`). In our simulations, we will focus on 2 different scenario: national small distance Grid (such as the French Grid5000 [4] infrastructure) with an average 20 ms RTT and larger scale Grids with 100 ms RTT. The topology used to test our XCP-TCP fairness solutions will be the one shown in Figure 1. In our experiments we have used $D_{drop} = 0.9999$ and $I_{drop} = 1.0001$.

In this set of experiments (the results are shown in Figure 3) we gradually incorporated 3 TCP flows after second 10 (each flows is incorporated with a 20-seconds delay), among 10 XCP flows (all XCP flows are started at second 0), with an $RTT = 20ms$ (case 1) and an $RTT = 100ms$ (case 2). We believe that these scenarios represent well what will happen in a real environment where thousand or hundreds of active flows share the network. At the same time, using a limited number of flows in our simulations (3 TCP and 10 XCP flows) can help us to understand the behavior of our XCP-TCP fairness solution, since we can easily observe and compare the evolution of every XCP and TCP flow in a dynamic network.

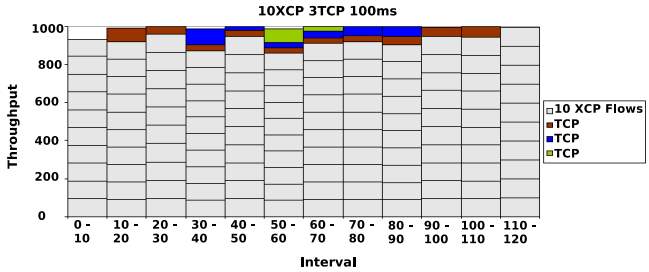
The simulation results of the first case are shown in Figure 3(a) ($RTT = 20ms$), where we can see that every time a new TCP flow starts, it gets more bandwidth than needed (seconds 10, 30, 50). However, after Slow Start finishes, our XCP-TCP mechanism succeeds in ensuring fairness between XCP and TCP, as we can observe between seconds 20 and 30 and between seconds 50 and 110. Figure 3(a) proves also that our mechanism ensures fairness even though every TCP flow comes in/out asynchronously to the network.

In the case where the $RTT = 20ms$, the worst fairness level is found maybe between seconds 30 and 50, where 2 TCP flows share close to 300Mbps while the remaining 10 XCP flows share around 700Mbps ($BW_{TCP} \approx 170.66Mbps$ and $BW_{XCP} \approx 853.34Mbps$). On the other hand, one of best fairness level is found between second 60 and 70, where 3 TCP flows share approximately 250Mbps while the remaining 10 XCP flows share around 750Mbps ($BW_{TCP} \approx 236.30Mbps$ and $BW_{XCP} \approx 787.70Mbps$). Those results are far from the one shown in Figure 2.

When the $RTT = 100ms$, at the beginning every TCP flow takes more resources than needed, producing the execution of the XCP-TCP fairness mechanism. After finished the Slow-Start phase, due to our fairness mechanism, the amount of bandwidth taken by TCP is smaller than the maximum allowed band-



(a) 20 ms RTT



(b) 100 ms RTT

Fig. 3. 3 TCP flows appear among 10 XCP flows

width. Even if our mechanism does not penalize TCP flows when the TCP throughput is too low, since $p_{drop} \rightarrow 0$, the time needed by TCP to get enough resources is very large (see seconds 60 to 110 in Figure 3(b)). It is important to note that the time needed for TCP to get the resources decreases as the RTT decreases and the number of TCP flows increases.

In the case where the $RTT = 100ms$, the worst fairness level is found maybe between seconds 60 and 70, where 3 TCP flows share close to 100Mbps while the remaining 10 XCP flows share around 900Mbps ($BW_{TCP} \approx 236.30Mbps$ and $BW_{XCP} \approx 787.70$). On the other hand, one of best fairness level is found between second 100 and 110, where 1 TCP flow gets approximately 100Mbps while the remaining 10 XCP flows share around 900Mbps ($BW_{TCP} \approx 93Mbps$ and $BW_{XCP} \approx 931Mbps$). Still, those results are far from the one shown in Figure 2.

6 Limitations and Optimizations

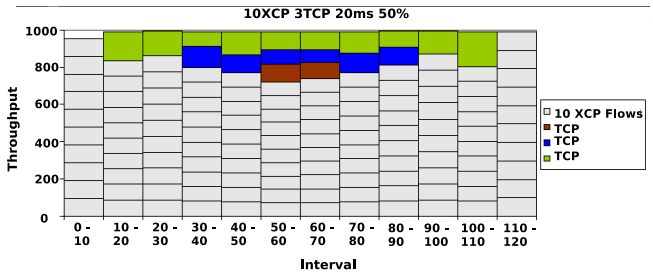
Even if the *zombie estimator* only executes a few operations for every incoming packet, those operations, added to the ones needed by the XCP algorithm, could increase significantly the processing time of packets.

For this reason, we have proposed to compute the number of XCP and TCP flows, taking as base only a percent of the total incoming packets. This modification may have an impact very important in the routers. For instance, in a

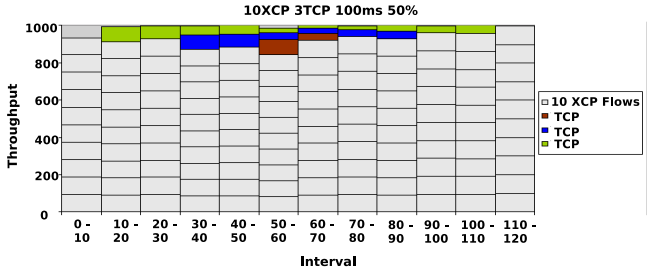
router processing 833,333 packets/s (10 Gbps whether every packet is composed by 1500B), to estimate the number of flows such a router would approximately generate 1,666,666 random numbers, access the zombie table 833,333 times, execute 833,333 comparisons, between others operations like write the ID flow when a *mis* is declared, etc. Taking into account the operations listed here, our 10Gbps routers should execute 3,333,332 operations/s. Thus, a reduction of 30% in the number of inspected packets should represent a decrease of approximately 1,000,000 operations/s in the router.

In order to estimate the active flow number without checking every incoming packet, a few changes in the zombie estimator are necessary. As shown early, the *hit* probability equation is given by $P(t) = (1 - \delta)P(t - 1) + \delta.Hit(t)$, where α reflects the probability to get a packet from the zombie table with the same ID of the incoming packet.

Thus, we propose to check only 50% of the total incoming packets. However, verifying only 50% of the total incoming packet and updating the zombie table with a low probability (25%) increases significantly the probability of missing flows in our zombie table. On the other hand, to increase the updating probability of the zombie table should decrease the problem of missing flows. For these reasons, in our work we have decided to use a probability of 50% to update the zombie table in case of *mis*. Therefore $\delta = 0.00025$.



(a) 20 ms RTT



(b) 100 ms RTT

Fig. 4. 3 TCP flows appear among 10 XCP flows - inspecting 50% of packets

With the modification introduced in our fairness mechanism (that let routers monitor only 50% of incoming packets to estimate the number of flows), we re-executed the experiment shown in the Section 5, where 3 TCP flows are incorporated gradually among 10 active XCP flows. Figure 4 shows the results of our new set of experiments.

As we can see in Figure 4, our fairness mechanism success in getting fairness between XCP and TCP flows inspecting only 50% of the total incoming packets. If we compare (*i*) Figure 4(a) ($RTT = 20ms$ and 50% of inspected packets) with Figure 3(a) ($RTT = 20ms$ and 100% of inspected packets), and (*ii*) Figure 4(b) ($RTT = 100ms$ and 50% of inspected packets) with Figure 3(b) ($RTT = 20ms$ and 100% of inspected packets); we can see that the results are very similar. Thus, we show that we can reduce significantly the operations executed by our XCP-TCP fairness solution, while keeping the same fairness level shown in the simulations of Section 5 (where we inspected 100% of packets).

7 Conclusion

In this article we have shown that when XCP and TCP share the bandwidth of the bottleneck link, TCP flows grab as many resources as needed, in damage of XCP flows. This unfairness problem affected most of the ERN protocols, like JetMax [17], TCP MaxNet [11], etc.

Therefore, in this article we presented a solution that ensures the XCP-TCP fairness, which is independent to the XCP protocol, easy to be applied to others ERN protocol and lightweight in terms of CPU and memory usage. Our XCP-TCP fairness mechanism lies on the execution of two main mechanism. The first mechanism estimates the resources needed by the XCP and non-XCP flows. The second mechanism limit the TCP throughput by dropping some of its packets with a probability p_{drop}.

The experiments presented along this article proved that our algorithms successfully "allocate" and "deallocate" bandwidth dynamically to XCP, in order to ensure the fairness between XCP and TCP. However, the level of fairness provided between XCP and TCP is strongly influenced by factors that escape to the control of our XCP-TCP fairness mechanisms. For instance, after that our XCP-TCP fairness solution drops packets to limit the TCP throughput and get fairness, TCP flows could finish with a throughput lower than the one desired. The capacity of TCP to re grab the lost bandwidth will depend on the RTT and the number of TCP flows.

In order to decrease the unfairness between XCP and TCP due to the RTT, one solution can be to "evaluate" the TCP aggressiveness when TCP increase their throughput. Thus, the drop probability could be updated in a more intelligent way.

Finally, we want to remark that with our XCP-TCP fairness solution is a new step to stimulate the deployment of the Explicit Rate Notification protocols in heterogeneous long-distance high speed networks, to improve the performance of long life flows.

References

1. A. Akella, S. Seshan, S. Shenker, and I. Stoica. Exploring Congestion Control, 2002.
2. Burton H. Bloom. Space/Time Trade-Offs In Hash Coding With Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.
3. R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.
4. Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.
5. D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *ACM SIGCOMM*, 2002.
6. Tom Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, 2003.
7. Jung-Shian Li and Yong-Shun Su. Random Early Detection With Flow Number Estimation and Queue Length Feedback Control. *Journal of Systems Architecture*, 52(6):359–372, 2006.
8. Dino M. Lopez Pacheco, Laurent Lefevre, and Cong-Duc Pham. Fairness Issues When Transferring Large Volume of Data on High Speed Networks With Router-Assisted Transport Protocols. In *High Speed Networks Workshop 2007, in conjunction with IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.
9. Dino M. Lopez Pacheco, Cong-Duc Pham, and Laurent Lefevre. XCP-i : eXplicit Control Protocol for Heterogeneous Inter-Networking of High-Speed Networks. In *Globecom 2006*, San Francisco, California, USA, November 2006.
10. Dino M Lopez-Pacheco and Congduc Pham. Robust Transport Protocol for Dynamic High-Speed Networks: Enhancing the XCP Approach. In *Proceedings of IEEE International Conference on Networks*, volume 1, pages 404–409, Kuala Lumpur, Malaysia, November 2005.
11. Bartek Wydrowskilete Martin Suchara, Ryan Witt. TCP MaxNet: Implementation and Experiments on the WAN in Lab. In *IEEE International Conference on Networks*, November 2005.
12. ns2. The Network Simulator. In *http://www.isi.edu/nsnam/ns/index.html*, 2007.
13. Teunis J. Ott, T. V. Lakshman, and Larry H. Wong. SRED: Stabilized RED. In *INFOCOM*, pages 1346–1355, 1999.
14. J. Postel. Transmission Control Protocol. RFC 793 (Standard), sep 1981. Updated by RFC 3168.
15. S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.
16. Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *INFOCOM*, 2004.
17. Derek Leonard Yueping Zhang and Dmitri Loguinov. JetMax: Scalable Max-Min Congestion Control for High-Speed Heterogeneous Networks. In *INFOCOM*, April 2006.