YeAH-TCP: Yet Another Highspeed TCP

Andrea Baiocchi, Angelo P. Castellani and Francesco Vacirca

INFOCOM Department - University of Roma ”Sapienza”, Via Eudossiana 18, 00184 Roma, Italy
e-mail: {baiocchi,castellani,vacirca}@infocom.uniroma1.it

***Abstract*— In recent years, several new TCP congestion control algorithms have been proposed to improve TCP performance over very fast, long-distance networks. High bandwidth delay products require more aggressive window adaptation rules, yet maintaining the ability of controlling router buffer congestion. We define a relatively simple experimental scenario to compare most current high speed TCP proposals under many metrics: efficiency, internal fairness, friendliness to Reno, induced network stress, robustness to random losses. Based on the gained insight, we define Yet Another High-speed TCP, as a heuristic attempt to strike a balance among different opposite requirements.**

***Index Terms*— High Bandwidth-Delay Product Network, TCP.**

I. Introduction

TCP has been defined and refined during the 80’s. Its strength and amazing flexibility stems from its longevity and capacity to accomplish its task even while the network evolved from a 64 kbps backbone to a multi-Gbps core network, with extensive use of wideband wireless access, to say the least. Achieved performance are not optimal, and concern has arisen in the scientific community as to the re-definition of TCP for use in large Bandwidth-Delay Product (BDP) networks, as provided by optical network core over geographic distance even for terrestrial networks[1]. Recent works devoted to this topic are addressed in Section II.

The aim of this work is to report an extensive experimental measurement of most current high speed TCP proposals, evaluated under a number of performance metrics. We consider efficiency in bandwidth exploitation, average packet delay, internal and RTT fairness, friendliness to Reno, robustness to random losses[2]. We set up a single bottleneck test-bed, that can include cross traffic and adjustable RTT and random packet loss; this is a trade-off between controllability and significance of the experimental results. We do not claim ours are definitive results, yet they are consistent and lead to sufficient insight that we felt worth defining a new heuristic for high speed TCP, which we named as Yet Another High-speed (YeAH) TCP.

The paper is organized as follows: Section II reviews recent literature on new proposals for TCP in high BDP networks and gives the motivations for our work. In Section III, the description of YeAH-TCP algorithm is provided. The experimental testbed is described in Section IV; Section V reports measurement results. The main conclusions are drawn in Section VI.

This work has been partially supported by the Italian Research Ministry under the PRIN FAMOUS grant.

[1]A well known instance of large BDP links is satellite.

[2]This is not so unrealistic, in view of the essentially bufferless architecture of DWDM based optical packet switches.

II. Related Works

In the recent literature, different strategies have been explored to address the problem of TCP in high BDP networks; these can be classified into four different categories:

- Loss-based
- Delay-based
- Mixed loss-delay-based
- Explicit congestion notification

Congestion control algorithms that consider packet loss as an implicit indication of congestion by the network belong to the first category. All proposals in this category (STCP [1], HSTCP [2], H-TCP [3], BIC [4] and CUBIC [5]) modify the increase and decrease rule of Reno congestion control to be more aggressive when they work in high BDP networks.

Other proposals (second category) consider delay as an indication for network congestion. The most important delay-based congestion control protocol for high BDP network is FAST TCP [6]; it employs an alternative congestion control algorithm using both queuing delays and packet losses as indications of congestion in the network. Under normal working conditions, the congestion window is updated every RTT and depends on the estimation of the average RTT.

In the third category, we find some approaches based on a mix between delay-based and loss-based congestion indications. TCP Africa [7] is a dual state algorithm; the congestion window is updated differently in the two operation modes. Specifically the algorithm switches between the “slow” mode state in which the congestion window is updated according to Reno algorithm, and the “fast” mode state in which the congestion window is updated according to HSTCP increase rule. Switching between states is governed by the number of queued packets in the bottleneck buffer, inferred through a delay-based approach. As the authors highlight, TCP Africa is aggressive when the pipe is not full and it behaves like Reno when the full link utilization is achieved.

Another approach similar to TCP Africa is the one proposed in [8]. Compound TCP borrows from Africa TCP the idea to be aggressive only when the capacity of the bottleneck link is underutilized, by using a different approach: the algorithm keeps two different variables, the standard congestion window $cwnd$ and the delay window $dwnd$; the congestion window is updated according to the Reno scheme and the number of outstanding packets is the sum of the congestion window and the delay window. The purpose of the delay window is to enable Compound TCP to be more aggressive when the delay variation is low. This behavior is achieved by enlarging and shrinking the delay window according to the round trip time estimation.

In the last category, there are those solutions (e.g XCP [9]) that require explicit signal from the network elements to inferr the congestion of the network. In the remainder of this work, algorithms belonging to this category are not considered since their development requires the cooperation of router and hence a modification of today Internet. Besides, we do not consider cross-layer solutions involving e.g. AQM; we assume congestion control relies only on end-to-end mechanisms.

As shown in this section, several proposals exist to overcome the problem of TCP in high BDP network. However, recent discussions on the end2end mailing list [10] and several experimental and simulative works ([11], [12] [13]) reveal that there is no agreement on the best congestion control paradigm for high BDP networks. A protocol, whose performance are optimal in a particular scenario, may perform unsatisfactorily in other scenarios. Moreover, different testbeds lead to different results due to minimal differences in the protocol implementation or in the network scenario design. Besides it is not clear, which are the metrics that should be considered to evaluate a new congestion control algorithm. A big effort in this direction has been carried out by IETF in [14] to standardize the methods and the metrics for congestion control evaluation.

In our opinion, the new proposals for TCP in high BDP networks are not evaluated correctly since it is often forgotten that one of the main characteristic should be the capability of the protocol to avoid congestion in the network and not only the capability of the protocol to achieve the full link utilization. I.e., an important issue that is not payed enough consideration in most performance evaluation papers is if the proposed algorithm is optimal from a congestion controller point of view. If we consider a single STCP flow in a single bottleneck scenario, it is able to achieve the full link utilization in few round trip times, since its increasing rule is aggressive. This leads to multiple losses, whenever the bottleneck link buffer has been filled up, that can be rapidly recovered by an efficient loss recovery procedure, such as SACK TCP. In opposition, standard TCP is slow in reaching the steady state behavior in large BDP network since it increases its congestion window by one packet per RTT, but from the point of view of a loss-based congestion controller it is optimal, since it probes the network with one packet more per round trip time, which is the minimum increment rate adapted to the delay of the feedback signal. Nowadays, the large diffusion of TCP congestion control preserves network health for legacy Reno traffic and new-generation application with real-time or interactive requirements. Instead the lack of congestion control design in new transport protocol can cause network instability and non-negligible degradations.

In this context, the purpose of our work is twofold. On one hand, we propose *yet* another congestion control paradigm that is able to fully exploit the capacity of high BDP links without loosing its congestion control capabilities. On the other hand, results obtained with *yet* another experimental testbed, can be used by other researchers to gain a deeper insight in the evaluation of other existing proposals.

In the experimental evaluation section, the number of congestion controllers has been creamed off to make the obtained results easily readable; we compare CUBIC, HSTCP, H-TCP, Africa, Compound TCP and our proposal, namely YeAH-TCP. FAST TCP has not been considered since the protocol code is not publicly available; STCP has not been considered since it has been widely shown that the protocol is highly RTT unfair (see for example [11]). Since CUBIC TCP is the new candidate algorithm for Linux TCP default setting and it can be considered the evolution of BIC, BIC results are not shown.

III. YeAH: ALGORITHM DESIGN

In the design of YeAH-TCP we considered simultaneously different goals to be achieved:

1) Network capacity should be exploited efficiently. This is the most obvious goal, which can be achieved by modifying the congestion window update rules; as described later, YeAH TCP can exploit anyone of the increment rules of other proposals (e.g. STCP, H-TCP, etc.).
2) The stress induced to the network should be less or equal than that induced by Reno TCP. Most of the high speed TCPs induce congestion events frequently at the bottleneck router and the number of packet drops in a single congestion event are significantly higher as compared to standard Reno congestion control, degrading the performance achieved by other traffic sharing the path. The degradation is not limited to increased drop probability, but also heavy queuing delays and delay jitter.
3) TCP friendliness with Reno traffic. Another characteristic to be addressed is the behavior of the proposed algorithm when competing with Reno TCP. A “politically” acceptable algorithm should be able to compete fairly with Reno flows, avoiding starvation of competing flows, and simultaneously to exploit the link capacity.
4) The algorithm should be internally and RTT fair.
5) Performance should not be substantially impaired by lossy links. It would be reasonable to have a congestion controller that is able to fully exploit the link also when the packet loss probability due to medium impairments is not null (e.g. large bandwidth wireless access link).
6) Small link buffers should not prevent high performance. It is not feasible to design buffer size equal to the bandwidth-delay product in high BDP links as required by standard Reno congestion control [17]. This goal can be achieved by adopting a decrease policy in case of packet loss similar to the Westwood algorithm [16].

YeAH-TCP attempts to address all the aforementioned issues. It envisages two different modus operandi: “Fast” and “Slow” modes, like Africa TCP. During the “Fast” mode, YeAH-TCP increments the congestion window according to an aggressive rule (we chose STCP rule, since it is very simple to implement). In the “Slow” mode, it acts as Reno TCP.

The state is decided according to the estimated number of packets in the bottleneck queue. Let RTT_{base} be the minimum RTT measured by the sender (i.e. an estimate of the propagation delay) and RTT_{min} the minimum RTT estimated in the current data window of $cwnd$ packets. The current estimated queuing delay is $RTT_{queue} = RTT_{min} - RTT_{base}$. From RTT_{queue} is possible to infer the number of packets enqueued by the flow as:

$$Q = RTT_{queue} \cdot G = RTT_{queue} \cdot \left(\frac{cwnd}{RTT_{min}} \right) \qquad (1)$$

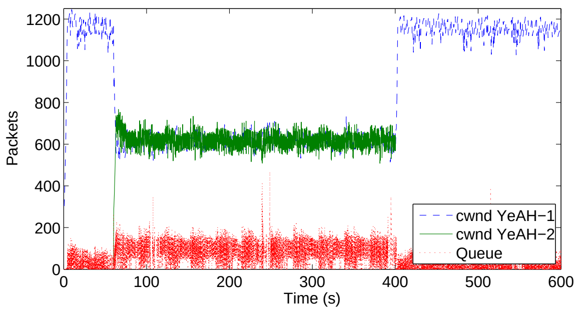
where G is the goodput. We can also evaluate the ratio between the queuing RTT and the propagation delay $L = RTT_{queue}/RTT_{base}$, that indicates the network congestion level. Note that RTT_{min} is updated once per window of data.

If $Q < Q_{max}$ and $L < 1/\varphi$, the algorithm is in the "Fast" mode, otherwise it is in the "Slow" mode. Q_{max} and φ are two tunable parameters; Q_{max} is the maximum number of packets a single flow is allowed to keep into the buffers and $1/\varphi$ is the maximum level of buffer congestion with respect to BDP. During the "Slow" mode, a precautionary decongestion algorithm is implemented[3]: whenever $Q > Q_{max}$, the congestion window is diminished by Q. Since RTT_{min} is computed once per RTT, the decongestion granularity is one RTT.

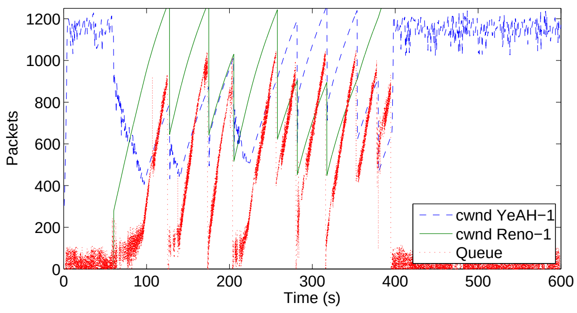
Considering the case that a single YeAH-TCP competes for the bottleneck link, Q is an estimate of the excess amount of packets to the minimum congestion window size required to exploit the available bandwidth. This amount of packets can be removed from the actual congestion window without degrading the goodput. When the number of competing flows increases, every flow attempts to fill the buffer by the same number of packets (at maximum Q_{max}) independently of the perceived RTT, achieving the internal RTT fairness. Moreover the precautionary decongestion prevents the bottleneck queue from building up too much, reducing queuing delays and diminishing packet losses due to buffer overflow. As shown in [15], the precautionary decongestion is optimal only when the flows that implement it do not compete with "greedy" sources, such as legacy TCP; in competition with "greedy" flows, the precautionary decongestion is not able to compete since it tends to release bandwidth to the greedy sources and to starve.

To avoid unfair competition with legacy flows, the YeAH-TCP protocol implements a mechanism to detect if it is competing with "greedy" sources. Consider the case of competition with a Reno flows, that do not implement the queue decongestion; when $Q > Q_{max}$ YeAH-TCP attempts to remove packets from the queue, the queuing delay increases on because legacy flows are "greedily" filling up the buffer. In this case, YeAH-TCP will stay hardly ever in "Fast" mode state and frequently in "Slow" mode. Whenever a YeAH-TCP flow competes with other YeAH-TCP flows, that implement the precautionary decongestion, the algorithm state oscillates frequently between "Fast" and "Slow" mode, since every flow applies the decongestion when $Q > Q_{max}$.

Thanks to this empirical remarks, it is possible to distinguish between the two different competition circumstances, counting the number of RTTs that the algorithm is in the two states. To this aim, two counting variables are defined: $count_{reno}$ and $count_{fast}$. $count_{fast}$ represents the number of RTTs in "Fast" mode. $count_{reno}$ is an estimate of the value of the congestion windows of competing Reno flows. The decongestion takes place only during the "Slow" mode and if $cwnd > count_{reno}$ to avoid that the congestion window decreases below the estimated value of the Reno flows congestion window. At the start-up $count_{reno}$ is initialized to $cwnd/2$, it is incremented by one every RTT in "Slow" mode and, when a packet loss is detected $count_{reno}$ is halved. The variable is reset to the current $cwnd/2$ whenever $count_{fast}$ is greater than a threshold, indicating that the flow is competing with other non-greedy flows. At the same time $count_{fast}$ is reset to 0. Figure 1 depicts two examples of the evolution of YeAH-TCP congestion window when competing with a YeAH-TCP flow (a) and when competing with a Reno flow (b). In the first case, when the second YeAH-TCP flow starts, the two flows converge steeply towards the same congestion window. In the second case (b), when the legacy TCP flow starts, the YeAH-TCP decrements the congestion window till the moment it gets aware to compete with a "greedy" flow and from this moment on, the two flows share the bandwidth in the Reno way.



(a) YeAH vs YeAH



(b) YeAH vs Reno

Fig. 1. YeAH-TCP congestion window evolution.

Last issue is what happen in case of packet losses. When a loss is detected by three duplicate ACKs, the current estimate of the bottleneck queue Q, can be exploited to find the value of packets that should be removed from the congestion window to empty the bottleneck buffer, yet leaving the pipe full. This rule is similar in principle to the one used by Westwood TCP [16]. This rule permits to obtain the full link utilization after a loss, for every value of the bottleneck buffer size and in case of losses independent of the congestion of the network (e.g. wireless links) In case of three duplicate ACKs, when YeAH-TCP does not compete with Reno flows[4], $cwnd$ is decreased by $\min\{max\{cwnd/8, Q\}, cwnd/2\}$ segments. If YeAH-TCP competes with Reno flows, the congestion window is halved.

IV. Experimental Testbed

To investigate the effectiveness of the new congestion control proposal, a testbed has been designed and implemented. Its primary scope was to recreate a realistic high-speed long-distance network environment to test congestion control protocols. The

[3]As it will be explained in the following, the decongestion is employed only when the YeAH-TCP is not competing with Reno flows.

[4]This fact is recognized by comparing the number of consecutive RTTs spent in "Slow" mode up to the current time with a threshold.

testbed development platform is based upon the GNU/Linux operating system, with three PCs running a modified version of the 2.6.16.2 kernel release. The physical network topology of the connections is based on 1000BaseTX physical connections, between the hosts. The logical topology of the testbed is depicted in Figure 2. Host 1 and Host 2 are connected to router

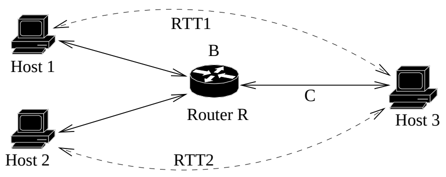


Fig. 2. Testbed logical topology

R with two full duplex 1Gbps links; the link between router R1 and host 3 is the bottleneck link and its capacity C varies between 10kbps and 500Mbps. The RTT between host 3 and host 1 is RTT_1, whereas the RTT between host 3 and host 2 is RTT_2. Both RTTs can varies between 12ms and 480ms. The router buffer B is always configured as a fraction of the BDP=$C \cdot min(RTT_1, RTT_2)$; the $ssthresh$ is initially set to a value equal to 50% of BDP. Every experiment has a fixed duration of 600s[5] and each measurement point is the average of at least three experiments.

It is worth to pinpoint that to evaluate the congestion control algorithm it is required that the bottleneck link is not directly connected to the sender. In fact, whenever the outgoing network interface has been filled by the sender, the congestion window stops to increase (disabling the congestion control) and the sender transmits at full rate.

The Linux TCP/IP internetworking stack has been modified to make it fully RFC compliant; Linux implementation, in fact, does not always respect RFCs as reported in [18]. Moreover, Africa TCP and Compound TCP have been implemented in the Linux kernel to test their performance. A patch for Linux kernel is publicly available at [19].

V. Experimental Results

A. Round-trip time effect on congestion control

First, we analyze the effect of different round trip times on different congestion control mechanisms. In this scenario, the round trip times RTT_1 and RTT_2 are equal and vary between 15ms and 480ms; the bottleneck link capacity C is 500Mbps, the buffer size B is 100% of BDP and random packet loss events are generated with probability p_{loss}=$5 \cdot 10^{-7}$.

Figure 3 depicts the goodput of different congestion control mechanisms, varying RTT_1 and RTT_2. It can be observed that all the algorithms are able to exploit the whole link capacity for BDP lower than few thousands of packets; at higher BDP values, p_{loss} has a big impact on their efficiency. Hybrid approaches (Africa and Compound) degradation is more relevant since the congestion window is halved in case of packet loss detection, whereas other protocols use lower decreasing factor (e.g., 0.2 for CUBIC TCP). As far as regards Reno TCP, it experiences high goodput degradation when RTT increases since it is not able to increase its congestion window during

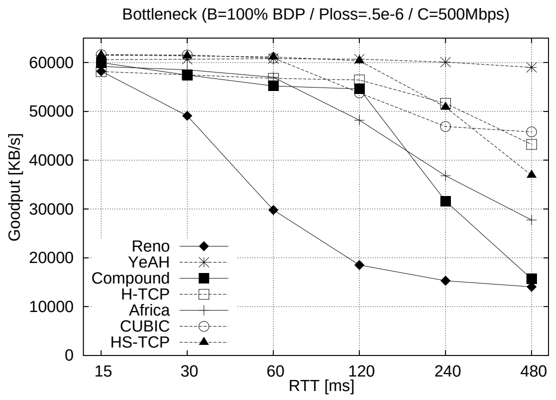


Fig. 3. TCP Goodput varying the round-trip time.

the 600s of the experiment. YeAH-TCP is able to fully exploit the network capacity, irrespective of the BDP and of the independent packet losses.

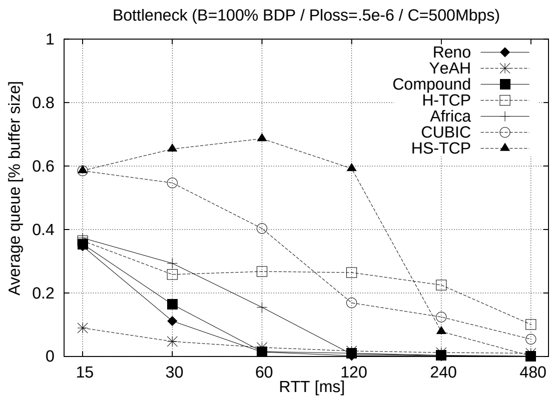


Fig. 4. Normalized queue length varying the round-trip time.

In Figure 4, the average queue length normalized to the buffer size is depicted. As we can see, Reno puts a relevant load on the network as long as it is able to exploit the network capacity; when its goodput starts to decrease, the induced load decreases. Africa and Compound load is comparable to Reno, at low RTTs; as RTT grows the load level is still comparable with the previous values, however the goodput stays stable to the previously obtained levels. The load measured for hybrid algorithms appear to lower as RTT grows, but this measure is strictly dependent on the duration of the experiments, as it requires a longer time to the Reno component to rise to a value equal to BDP. Africa load is higher with the selected experiment duration, however, the load level would be comparable in case of longer experiments. As far as regards HSTCP and CUBIC, their queue length is significantly higher than Reno, Africa and Compound ones as long as they fully exploit the network capacity (RTT between 15ms and 120ms). High queue length induces performance degradation of real-time and/or interactive application significantly more than Reno. As soon as RTT (and BDP as well) increases (between 240ms and 480 ms) the goodput decreases and the jitter lowers as well; H-TCP queue level is similar to the Reno one and it is almost constant as RTT increases. YeAH TCP induced load is stably lower than other algorithms and especially it offers always low load while

[5]We checked that the initial transient phase of TCP is substancially died out except of Reno TCP.

fully utilizing the link; this characteristic is achieved by means of the precautionary queue decongestion algorithm.

B. Bottleneck buffer size effect on congestion control

As a second issue, we analyze the performance of different TCPs varying the bottleneck buffer size with respect to BDP. In this case, C is 500Mbps, p_{loss} is $5 \cdot 10^{-7}$ and RTT_1=RTT_2=80ms. Figure 5 depicts the goodput varying the buffer size. As regards Reno TCP, it is not able to fully exploit the available bandwidth independently of the buffer size. All the loss based algorithms experience a serious goodput degradation when operating with low buffer sizes, whereas YeAH-TCP, Africa and Compound performance are not affected by lower buffer sizes due to their fixed buffer requirement. The normalized average queue length (not shown here) confirms the results presented in Figure 4.

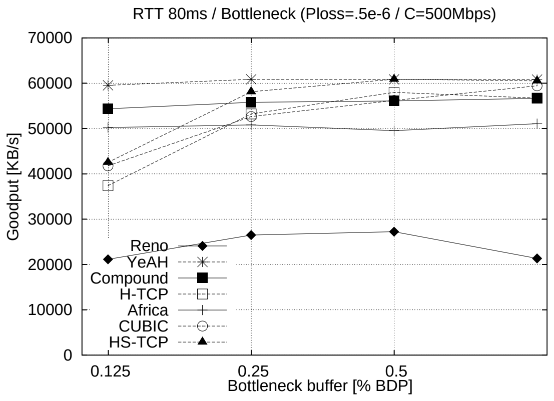


Fig. 5. Goodput varying bottleneck buffer size.

C. Link loss probability impact on performance

In this section, the effect of a random packet losses on congestion control performance is reported. In this scenario, C is 500Mbps, RTT_1=RTT_2=200ms, B is 100% of the BDP and p_{loss} varies between 10^{-8} and 10^{-4}; results are reported in Figure 6. All congestion control proposals are highly impacted as the link loss probability grows up. Reno TCP obtains the worst results. All protocol performance degrades because of the congestion control loss-based component that reduces the congestion window when a packet loss is detected. YeAH-TCP is able to sustain higher link loss rate against other protocols because it does not reduce the congestion window according to a constant factor, but dependently on the estimated BDP. However, when p_{loss} is very high, also YeAH-TCP performance degrades substantially.

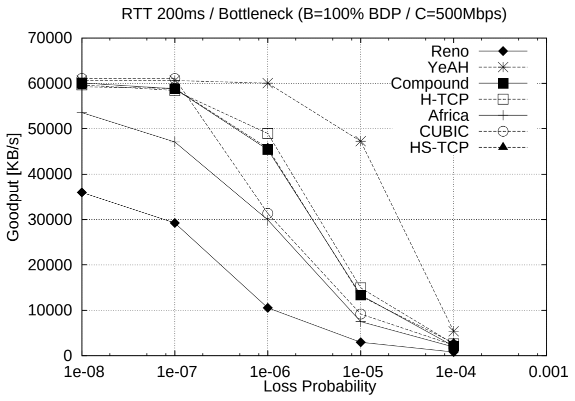


Fig. 6. Goodput varying link loss probability.

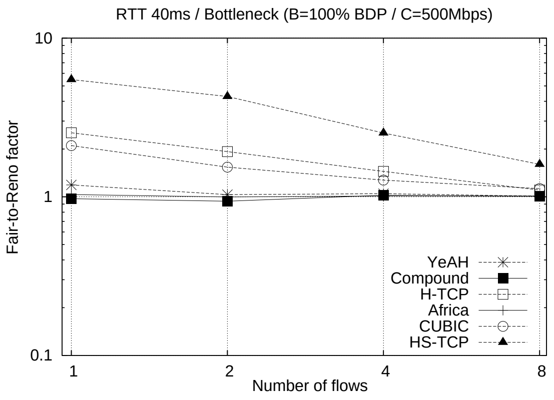


Fig. 7. Fair-to-Reno factor varying number of Reno flows

D. Reno friendliness

In this section, the friendliness of highspeed congestion control with Reno legacy flows has been analyzed. We define the *Fair-to-Reno* factor as the ratio of the aggregated goodput of n Reno flows competing against a Reno flow, to the aggregated goodput of n Reno flows competing against the selected algorithm. Figure 7 reports the *Fair-to-Reno* factor as a function of n. All the loss-based variants degrade substantially the performance of Reno traffic due to their highly aggressive increasing factors and to their lower decrement factors. When the number of flows grows the effect becomes less relevant. The effect obtained by the combination of these modifications leads to “stealing bandwidth” from the legacy flows. It is interesting to point out that H-TCP and CUBIC fairness is closer to 1, in particular when the number of flows increases, whereas HSTCP is the most unfair. Africa, Compound and YeAH-TCP are Reno-friendly independently of the number of competing flows, since their “fast component” is disabled when competing with Reno flows and their behavior is similar to Reno one. It is worth to emphasize that in the selected scenario, the bottleneck capacity is always fully exploited.

E. Internal and RTT fairness

The internal and RTT fairness has been evaluated by computing the Jain’s fairness index varying the ratio between RTT_2 and RTT_1. In this scenario RTT_1 is 25ms, C=500Mbps and B=100% of BDP. When the RTT ratio is 1, we are evaluating the internal fairness of the considered protocol. Results are depicted in Figure 8. When the RTT ratio is 1, all algorithms are internally fair. However, when the RTT ratio increases, all protocols, except H-TCP, CUBIC and YeAH-TCP, are highly RTT unfair; this was a known limit of Reno that has been inherited by new proposals. CUBIC is less RTT unfair thanks to its absolute time-dependant growth that allows the competing flows to grow by the same value the congestion window between consecutive packet losses; however this doesn’t enable it to get a completely fair behavior. As far as regards H-TCP, it

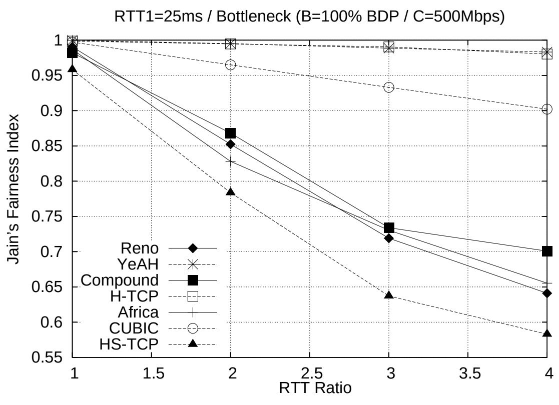


Fig. 8. Jain's index varying round-trip time ratio

uses a time-dependant growth in conjunction with a reduction rule, that reduces by a lower factor the congestion window of the flow with higher RTT; the reduction is proportional to the bottleneck router buffer size normalized to the BDP of the flow, this property enables H-TCP to be RTT fair. YeAH TCP also proves to be RTT fair, this was an expected result because every flow tries to keep in the bottleneck buffer a fixed number of packets, whatever is the round-trip time; thus every flow should get a fair share of the bottleneck.

F. Impact of background traffic on congestion control

As a last issue, we consider the effect of background traffic on the congestion control algorithms. For background traffic generation, we use a simple tool that generates web traffic according to the Scalable URL Reference GEnerator (SURGE) model [20]. Every single web client entity has been configured to generate a very high load of requests (i.e. 500kbps per client), so each of them can be assumed to be a web proxy. Figure 9 depicts the goodput, varying the number of web client

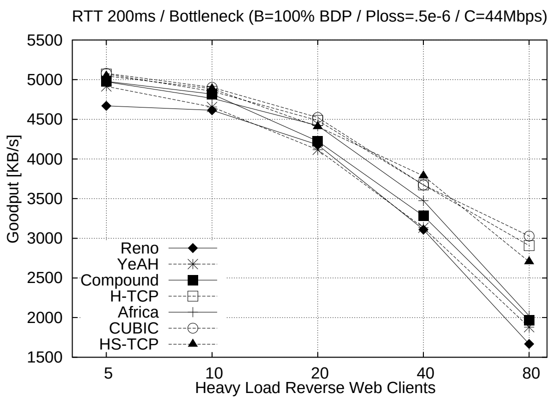


Fig. 9. Goodput varying the number of web client sessions

sessions. C is 44Mbps, RTT_1=RTT_2=200ms, B=100% BDP and p_{loss}=$5 \cdot 10^{-7}$. We can note that loss-based congestion control (i.e. H-TCP, HSTCP and CUBIC) obtain a higher goodput with respect to other proposals; YeAH-TCP, Africa and Compound obtain a goodput comparable to Reno, not impacting the performance of legacy flows. Results, not shown here for lack of space, confirm that YeAH-TCP performance are not affected by background traffic.

VI. CONCLUSION

We have shown a comparison of many high speed TCP proposals in a simple, parametric large BDP networking testbed, along with a new yet significant proposal, so called YeAH-TCP. Experimental results show that, when BDP grows up, all the aggressive loss-based approaches, like HSTCP, H-TCP, CUBIC, experience growing queuing delays and TCP Reno unfriendliness, besides they are not able to fully exploit the link bandwidth when the packet loss probability is not negligible. Hybrid approaches, such as Africa and Compound, have better properties yet they fail to get high goodput on lossy links, still inducing a relevant network stress at the bottleneck. As regards YeAH-TCP, it is able to exploit efficiently the available bandwidth, without inducing stress to the network elements. The protocol is internally and RTT fair, TCP Reno friendly and reacts correctly to packet losses independent of congestion. Further work is required to verify the performance of our proposal in different network scenarios and to formalize analytically some heuristics utilized of the design.

REFERENCES

[1] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," in proc. of PFLDnet 2003, Feb. 2003, Switzerland.
[2] Sally Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Experimental, December 2003.
[3] R.N. Shorten, D.J. Leith, "H-TCP: TCP for high-speed and long-distance networks" in proc. of PFLDnet, Argonne, 2004.
[4] L. Xu, K. Harfoush, I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," in proc. of INFOCOM 2004, Hong Kong, China, Mar. 2004.
[5] I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," in proc. of PFLDnet 2005, February 2005, Lyon, France.
[6] C. Jin, D. X. Wei and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in proc. of INFOCOM 2004, Hong Kong, China, Mar. 2004.
[7] R. King, R. Baraniuk, R. Riedi, "TCP Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP," in proc. of IEEE INFOCOM 2005, Miami, USA, Mar. 2005
[8] K. Tan, J. Song, Q. Zhang, M. Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," in proc. of IEEE INFOCOM 2006, Barcelona, Spain, Apr. 2006.
[9] D. Katabi, M. Handley, C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in proc. of ACM SIGCOMM 2002, Pittsburgh, USA, Aug. 2002.
[10] end2end-interest – discussion of end-2-end research and design principles http://www.postel.org/mailman/listinfo/end2end-interest
[11] S. Mascolo, F. Vacirca, "The effect of reverse traffic on the performance of new TCP congestion control algorithms for gigabit networks," in proc. of PFLDnet 2006, Nara, Japan, Feb. 2006.
[12] Yee at al., "Evaluating the Performance of TCP Stacks for High-Speed Networks," to appear on IEEE Transactions on Networking.
[13] S. Ha, Y. Kim, L. Le, I. Rhee, L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants," in proc. of PFLDnet 2006, Nara, Japan, Feb. 2006.
[14] Sally Floyd, IETF INTERNET-DRAFT, "Metrics for the Evaluation of Congestion Control Mechanisms," 7 August 2006.
[15] L. Brakmo, L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," IEEE Journal on Selected Areas in Communication, vol. 13, no.8 , pp. 1465–1480, Oct. 1995
[16] S. Mascolo at al. "TCP Westwood: End-to-End Bandwidth Estimation for Efficient Transport over Wired and Wireless Networks," proc. of ACM Mobicom, Rome, Italy, July 2001.
[17] Villamizar C., Song C., "High Performance TCP in ANSNET", ACM CCR, vol. 24, no. 5, pp. 45-60, Oct. 1994.
[18] P. Sarolahti, A. Kuznetsov, "Congestion Control in Linux TCP," in proc. of USENIX'02, Jun. 2002.
[19] http://infocom.uniroma1.it/~vacirca/yeah
[20] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," proc. of ACM SIGMETRICS Conference, Madison, USA, 1998.