# The effect of reverse traffic on the performance of new TCP congestion control algorithms

Saverio Mascolo* and Francesco Vacirca†

*Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari
Via Orabona 4, 70125 Bari, Italy
Email: mascolo@poliba.it
†Infocom Department
University of Rome "La Sapienza"
Via Eudossiana 18, 00184 Rome, Italy
Email: vacirca@infocom.uniroma1.it

***Abstract*— In recent years, several new TCP congestion control algorithms have been proposed to speed up the TCP over very fast networks. All these algorithms propose their own modifications of the increasing/decreasing phases of classic Reno/NewReno TCP. On the other hand, all of them preserve the self-clocking mechanism, which is a fundamental part of the classic Van Jacobson TCP congestion control. For this reason, it is also said that the TCP dynamics is ack-clocked. An important consequence of the ack-clocking is that a TCP flow can be heavily affected by the presence of reverse traffic provoking congestion along its ack path. In this paper we use the ns2 simulator to investigate the effect of reverse traffic on the behavior of the new TCP stacks: BIC TCP, HSTCP, Hamilton TCP (H-TCP), STCP and FAST TCP. Also Westwood+ TCP is considered even though this stack was not designed to address the special case of very fast networks and it employs the classic slow-start+congestion avoidance probing phases of the TCP. We consider reverse traffic of the same type of the forward traffic in order to study the behavior of each single protocol with respect to stability and fairness, which are fundamental properties to be investigated before other important issues such as inter-protocol friendliness. Simulation results reported in this paper reveal that new TCP stacks designed for fast networks experience goodputs comparable to those of standard TCP Sack or TCP Westwood+ but with larger packet retransmission probabilities and timeout events when in the presence of reverse and web background traffic.**

***Index Terms*— HighSpeed Network, TCP congestion control, ns2 simulations**

## I. INTRODUCTION

In recent years, issues regarding the behavior of TCP in high-speed and long-distance networks have been extensively addressed in the networking research community, both because TCP is the most widespread transport protocol in the current Internet and because bandwidth-delay product continues to grow. The well known problem of TCP in high bandwidth-delay product networks is that the TCP Additive Increase probing mechanism is too slow in adapting the sending rate to the available bandwidth.

This work is supported by the Italian Ministry for University and Research (MIUR) under the PRIN project FAMOUS

To overcome this problem, many modifications have been proposed such as FAST TCP [1], STCP [2], HSTCP [3], H-TCP [4], BIC TCP [5] and CUBIC TCP [6].
It is important to remark that all new proposal must preserve backward compatibility with standard TCP implementations, a requirement that is usually referred to as "TCP friendliness". Friendliness enables new TCP stacks to coexists in the same networks with older stacks. However, before investigating inter-protocol issues such as friendliness, it is mandatory to investigate the behaviour of each single protocol by itself in order to test fundamental properties such as stability and fairness. To the purpose, in order to investigate the basic behaviour of these new stacks, we will consider a simple single bottleneck scenario with few sender entities. In all tests TCP senders will be equipped with the same stack to investigate the intra-protocol behaviour of each protocol. In particular, the effect of reverse and web background traffic will be analyzed using the ns2 simulator [7].
The paper is organized as follows: Section II provides background on new generation TCP congestion control; Section III reports simulation results and Section IV draws the conclusions.

## II. BACKGROUND ON HIGH SPEED TCP CONGESTION CONTROL

In this Section we briefly summarize main features of the TCP stacks proposed for gigabit networks that we are going to investigate in this paper. In particular, we will consider the standard TCP NewReno SACK, STCP, HSTCP, H-TCP, BIC and FAST TCP. In the following of the paper we will refer to the set of TCP new proposals as New Generation TCPs (NGTCPs) to distinguish them from standard TCP. Also Westwood+ TCP is considered even though this stack was not designed to address the special case of very fast networks and it employs the classic slow-start+congestion avoidance probing phases of the TCP.The reason is that the Westwood+ setting of the control windows after congestion, which is based on measuring the end-to-end available bandwidth, reveals to be

effective in gigabit networks as shown in [10].

The classic Van Jacobson TCP congestion control [11] is essentially made of a probing phase and a decreasing phase. The probing phase of standard TCP consists of an exponential growing phase (i.e. the slow-start phase) and of a linear increasing phase (i.e. the congestion avoidance phase). The probing phase stops when congestion is experienced in the form of 3 duplicate acknowledgments or a timeout. At this point Reno or NewReno TCP implements a multiplicative decrease behavior. The Reno/NewReno setting of the congestion window during the congestion avoidance phase is: on ACK reception, $cwnd$ is increased by $1/cwnd$, when a loss occurs (3 DUPACKs are received), $cwnd$ is halved.

This behavior can be generalized as follows:

a) On ACK reception:

$$cwnd \longleftarrow cwnd + a \tag{1}$$

b) When 3 DUPACKs are received:

$$cwnd \longleftarrow cwnd - b \cdot cwnd \tag{2}$$

where $a$ is $1/cwnd$ and $b$ is 0.5. Some of the NGTCP algorithms can be easily described in terms of particular settings of $a$ and $b$ parameters.

*STCP* [2]. In the Scalable TCP , the increase factor $a$ is set to a constant value to make the growth of $cwnd$ independent of $cwnd$ itself. The author suggests to set $a$ = 0.01, which permits the rate to double in about 70 round trip times for any rate. The decreasing factor $b$ is set to 0.125.

*HSTCP* [3]. The High Speed TCP congestion control algorithm is divided into two parts depending on the value of the congestion windows. If $cwnd$ is lower than a threshold $cwnd_{low}$ of 38 packets , the values of $a$ and $b$ are the same of NewReno. When $cwnd$ is between $cwnd_{low}$ and $cwnd_{high}$ packets, $a$ and $b$ are computed as:

$$b = (0.1 - 0.5) \cdot \frac{\log cwnd - \log cwnd_{low}}{\log cwnd_{high} - \log cwnd_{low}} + 0.5 \tag{3}$$

$$a = 2 \cdot cwnd^2 \cdot p(cwnd) \frac{b}{2 - b} \tag{4}$$

where $p(w) = 0.078/w^{1.2}$. The rationale behind Equations 3 and 4 is the choice of a predefined response function that maps the steady-state packet drop probability to the average sending rate. The original TCP response function $w = 1.2/\sqrt{p}$ is modified to $w = 0.12/p^{0.835}$

*H-TCP* [4]. Hamilton TCP is another enhancement of TCP for high bandwidth-delay networks. It defines $\Delta_i$ as the time elapsed from the last congestion event for the $i$-th source. If $\Delta_i$ is lower than a threshold $\Delta_{th}$ the protocol behaves like NewReno TCP. If $\Delta_i > \Delta_{th}$, the increment factor $a$ is equal to:

$$a = 1 + 10(\Delta_i - \Delta_{th}) + (\frac{\Delta_i - \Delta_{th}}{2})^2 \tag{5}$$

It can be noticed that the increment factor has a quadratic trend in relation with the time elapsed from the last congestion event. The decrement factor $b$ is set dynamically by considering that full link utilization is achieved when the bottleneck buffer does not remain empty for a long period. To this purpose the authors propose to set $b = 1 - \frac{RTT_{min}}{RTT_{max}}$. This setting is similar to FAST and Westwood+ TCP window settings.

*BIC TCP* [5]. The BIC protocol consists of two parts: a binary search increase phase and an additive increase phase. In the binary search phase the congestion window setting is performed as a binary search problem. After a packet loss, the congestion window is reduced by a constant factor $b$, $cwnd_{max}$ is set to the window size before the lost and $cwnd_{min}$ is set to the value of congestion window after the loss ($cwnd_{min} = b \cdot cwnd_{max}$). If the difference between the congestion window middle point $(cwnd_{max} + cwnd_{min})/2$ and the minimum congestion window $cwnd_{min}$ is lower than a threshold $S_{max}$ the protocol starts a binary search algorithm increasing the congestion window to the middle point, otherwise the protocol enters a "linear increase" phase and increments the congestion window by one for each received ACK. If BIC does not get a loss indication at this window size, then the actual window size become the new minimum window; otherwise, if it gets a packet loss then the actual window size becomes the new maximum. The process goes on till the window increment becomes lower than the $S_{min}$ threshold and the congestion window is set to $cwnd_{max}$. If the window grows more than $cwnd_{max}$, the protocol enters into a new phase ("max probing") that is specular to the previous phase; that is, it uses the inverse of the binary search phase first and then the additive increase. The default value of $b$ is 0.2.

*Westwood+ TCP* [8], [9]. The main idea of Westwood+ TCP is to set the control windows after congestion such that the bandwidth available at the time of congestion is exactly matched. The available bandwidth is estimated by counting and averaging the stream of returning ACK packets. In particular, when three DUPACKs are received, the congestion window ($cwnd$) is set equal to the estimated bandwidth ($BWE$) times the minimum measured round trip time ($RTT_{min}$).

*FAST TCP* [1]. FAST TCP employs an alternative congestion control algorithm. It uses 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. At every update the congestion window is set equal to:

$$\min\left\{2 \cdot cwnd, (1-\gamma)cwnd + \gamma\left(\frac{baseRTT}{RTT}cwnd + \alpha\right)\right\} \tag{6}$$

where $baseRTT$ is the minimum RTT observed, $RTT$ is the estimation of the average round trip time, $\gamma$ belong to the interval between 0 and 1 and $\alpha$ is a parameter that controls the fairness between flows and the number of packets that each flow attempts to keep into the network buffers.

## III. SIMULATION RESULTS

Figure 1 shows the reference scenario employed for carrying out computer simulations. It consists of 6 TCP Senders establishing connections with 6 TCP Receivers and two networks of hosts (X and Y) that generate Web traffic.

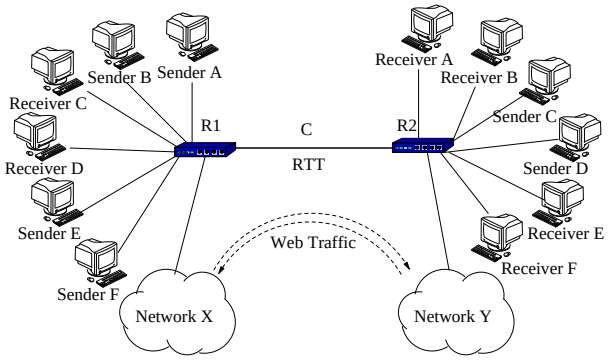



Fig. 1. Simulation scenario.

The bottleneck link between routers R1 and R2 is provisioned with a capacity $C$ of 250 Mbit/s. Links between router R1 and TCP senders/receivers and links between router R2 and TCP senders/receivers are provisioned with a capacity greater than $C$ so that the link R1-R2 is the bottleneck for each TCP flow. In all simulations the bottleneck buffer size $B$ is set equal to the bandwidth-delay product [17]; the packet size is 1500 bytes.

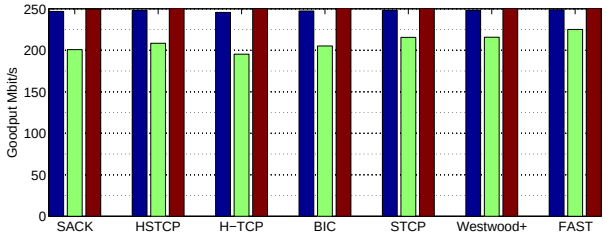
Simulations have been performed using the network simulator [7] version 2.28 with code and settings shown in [13], [14], [15] and [16]. The limited slow start algorithm [20] has been enabled to avoid that at the starting of the connections a large number of packets is lost causing starvation.

### A. Scenario with on-off reverse traffic

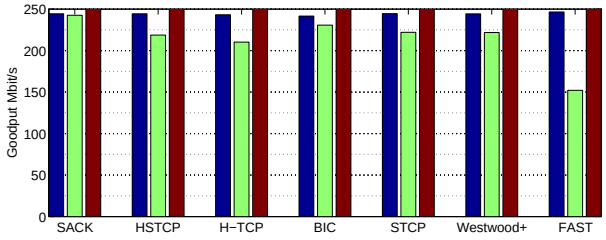
In the first simulation scenario that we consider, all TCP senders are greedy FTP users and employ the same flavor of the TCP protocol stack. Senders A and B start randomly within the first ten seconds of the simulation and last for the whole length of the simulation, which is 1000 seconds, whereas senders C and D start at second 333 and terminate at second 666. Every simulation run can hence be divided into three different phases: during the first phase (between 0 and 333 seconds) connection A and B share the whole bottleneck capacity; during the second phase (between 333 seconds and 666 seconds) connections C and D are also active and generate reverse traffic for connections A and B, which share the bandwidth with the ACKs belonging to connections C and D; during the last part of the simulation C and D are switched off again. Propagation delays of the dedicated links between R1 and the hosts on the left side of the figure and between R2 and the hosts on the right side are 2.5 ms. The propagation delay of the R1-R2 link is set to have two simulation cases: one case with connections having RTTs equal to 40ms and the other with RTT=160ms.

This particular scenario clearly aims at investigating the behaviour of new TCP stacks in the presence of reverse traffic of the same nature, which is a more than reasonable assumption. In the following, performance results always refer to those of Sender-Receiver A and Sender-Receiver B connections.

It is worth noting that all the NGTCP protocols used in this paper are used with the SACK option enabled except in the case of Westwood+ TCP. In fact, in case the SACK option is not enabled, we have verified that all NGTCP protocols analyzed in this paper perform very badly due to the large number of packets that are lost in one window. In fact, classical TCP congestion control increases the congestion window of at maximum one packet per round trip time, enabling TCP congestion control to wait an entire round trip time for a new probing. On the other hand, since NGTCPs have been designed to be faster than normal TCP to reach the steady state in the presence of high bandwidth-delay product networks, the congestion window is always increased of more than one packet per round trip time, leading to a large number of packets that need to be recovered during the Fast Recovery phase. Therefore, without the SACK information, NGTCPs are not able to recover all packets that are lost in the same window in about one round trip time, and starvation of data transfer can happen if the TCP transmission and receiver buffers are not large enough to store all the out-of-order packets received in the recovery phase. This behaviour will be further investigated in the extended version of this paper.




(a) RTT=40ms

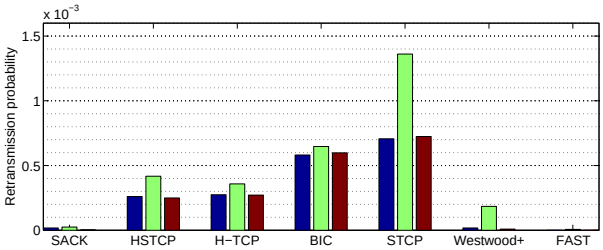



(b) RTT=160ms

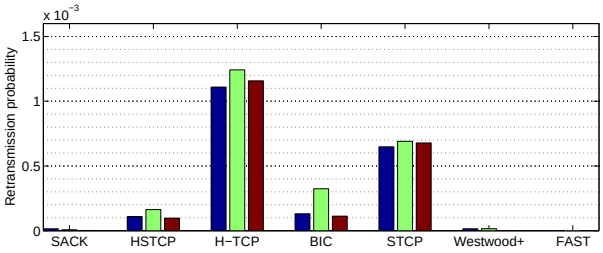
Fig. 2. Goodput in the three phases of simulations with Web traffi c disabled for the different TCP stacks.

Figure 2 shows the goodputs of each protocol stack during the three phases of the simulation scenario in the cases of RTT equal to 40ms (Figure 2(a)) and RTT equal to 160ms (Figure 2(b)).

When RTT is 40ms (corresponding to a bandwidth-delay of 833 packets), all protocols achieves a high goodput level (above the 98% of link utilization) in the first and in the third phase of the simulation. During the second phase of the simulation, all TCP stacks dynamics are affected by the

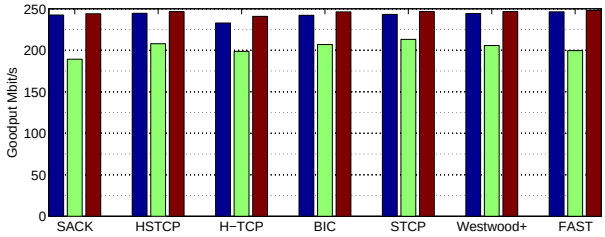



(a) RTT=40ms

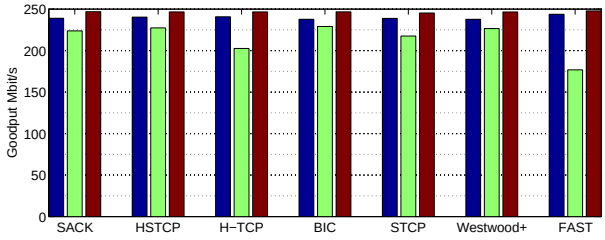



(b) RTT=160ms

Fig. 3. Retransmission probabilities in the three phases of simulations with Web traffi c disabled for the different TCP stacks.




(a) RTT=40ms




(b) RTT=160ms

Fig. 4. Goodput in the three phases of simulations with Web traffi c enabled for the different TCP stacks.

presence of reverse traffic, which does not allow TCP protocols to achieve full link utilization. The link utilization varies from the 90% of FAST TCP to the 78% obtained with H-TCP.

When RTT is 160ms, all protocols provide high link utilization (above 96%), including SACK TCP and Westwood+ TCP that are fast to reach the maximum congestion window thanks to the slow start phase. In the second phase of the simulation, when the reverse traffic is active, results change and the highest link utilization is achieved by SACK TCP whereas FAST TCP achieves the lowest link utilization (i.e. about 60% of the available bandwidth). Among other protocols, BIC obtains 92% of link utilization, followed by STCP, Westwood+, HSTCP and H-TCP in the order. In the last 333 seconds of simulations, all protocols provide full link utilization.

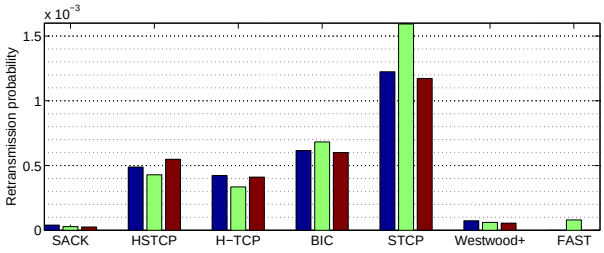
Besides link utilization provided by each protocol, it is also important to look at the packet retransmission probability that each congestion control algorithm experiences when trying to achieve full link utilization. Figure 3 shows the packet retransmission probability measured during the three phases of simulations for RTT equals to 40ms (a) and 160ms (b).

It is worth noting that: i) FAST TCP congestion control experiences a very small amount of packet losses as expected; ii) H-TCP increases its aggressiveness when the round trip delay increases, provoking a larger number of packet losses, iii) the packet retransmission probability of BIC TCP and HSTCP decreases with the increasing RTT, iv) STCP packet loss probability is almost independent of the RTT when the reverse traffic is not present and v) SACK TCP and Westwood+ TCP packet retransmission probabilities are smaller than those of other protocols and decrease when RTT increases. To conclude, from Figures 2 and 3 it can be said that, except from FAST TCP, all new TCPs provide goodputs comparable to those of standard TCP SACK but with larger packet drop rates.

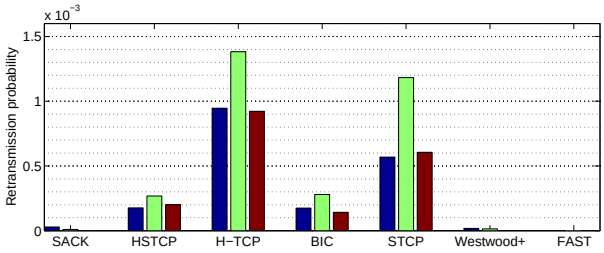
### *B. Scenario with reverse traffic + background Web traffic*

This scenario is derived from the previous one by superposing background Web traffic in both directions in order to avoid phase effects. Hosts in network X and network Y shown in Figure 1 generate Web traffic with standard SACK TCP as transport protocols from network X to network Y and vice versa. Web traffic is generated according to the following model. The interarrival time between new connections is generated with an exponential distribution and a random number of packets is associated to each new flow, drawn from an empirical distribution. Here 4 different lengths have been considered: 50% of flows have 1 packets, 20% have 6 packets, 20% have 18 packets, the remaining 10% have 190 packets. This simple web traffic model allows us to control the traffic load generated by networks X and Y. In all simulations we set the inter-arrival time to 50 ms, which contributes an offered load of about 5.8 Mbit/s per direction.

Figures 4 and 5 show goodputs and packet retransmission probabilities during the three different phases of the simulations with the Web traffic on for the whole length of the simulations. Goodputs are close to the ones obtained when the Web traffic is not active minus the portion of bandwidth used by the Web traffic. The only difference is the goodput experienced by FAST TCP when RTT is 160ms, that increases




(a) RTT=40ms




(b) RTT=160ms

Fig. 5. Retransmission probability in the three phases of simulations with Web traffi c enabled for the different TCP stacks.

from 150 Mbit/s to 175Mbit/s when both Web traffic and the reverse traffic are active.

As far as regards the packet retransmission probability, Figure 5 shows that packet retransmissions increase in most of the cases when in the presence of Web traffic.

A further insight into the behaviour of the different TCP stacks is provided by Figures 6-12, which plot congestion window dynamics during the simulations. Figures show the congestion window of Sender A and Sender B for the different TCP stacks (SACK, HSTCP, H-TCP, BIC, STCP Westwood+ and FAST TCP respectively) in the presence of homogeneous reverse traffic, i.e. same TCP stack and Web traffic active. In all the figures the upper and lower part represent the case of RTT equals to 40 and 160 ms respectively. During the first phase of the simulation (between 0 and 333 seconds) connections A and B share the whole bottleneck capacity; during the second phase (between 333 seconds and 666 seconds) connections C and D are active and connections A and B share the bandwidth with ACKs belonging to connections C and D. In the last part of the simulation C and D are switched off again.

From the figures it is possible to notice that in the presence of background Web traffic, the congestion windows of H-TCP, HSTCP, BIC and STCP drop to 1 several times because of timeout expiration. The lower is the RTT, the higher is the number of timeouts that occur. Another result that is worth noting is the congestion window behaviour of STCP when RTT is 160ms: in this case the congestion windows of the two forward connections are always remarkably different meaning that the fairness is low.

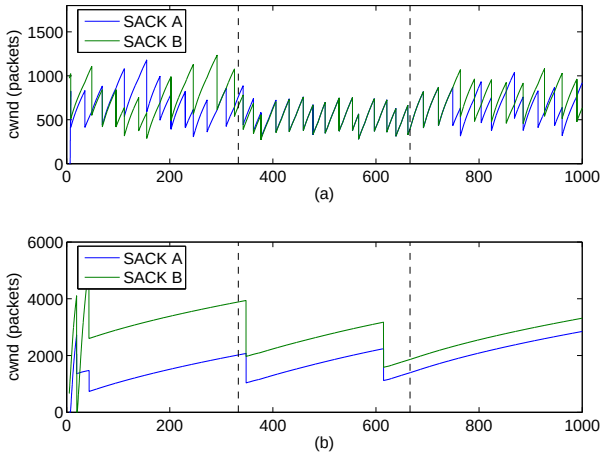



Fig. 6. SACK TCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

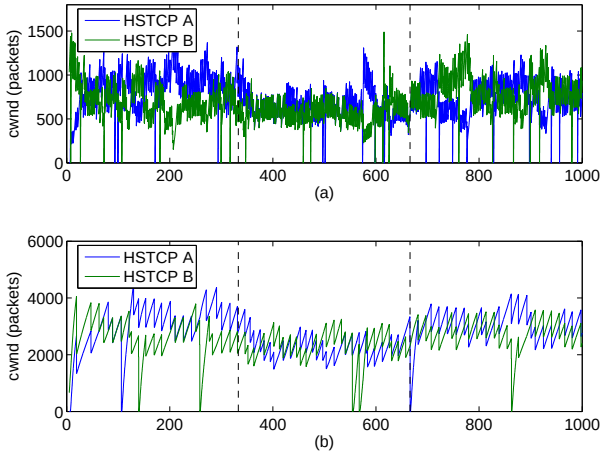



Fig. 7. HSTCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

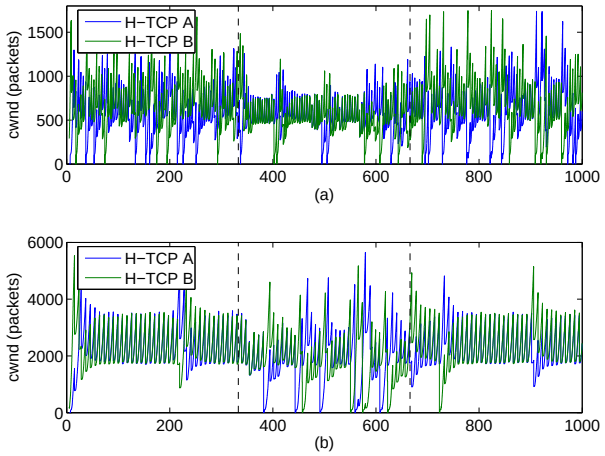



Fig. 8. H-TCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

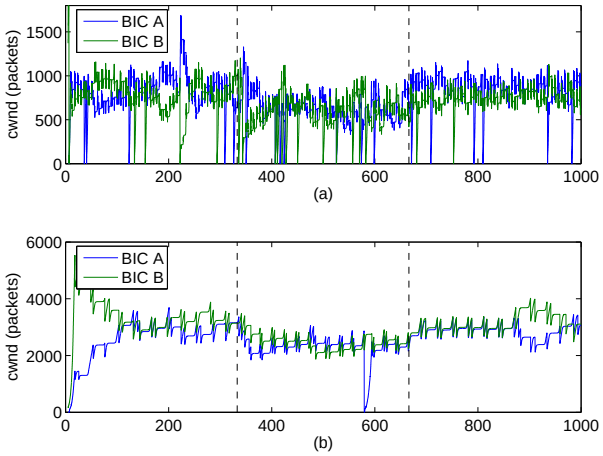



Fig. 9. BIC: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

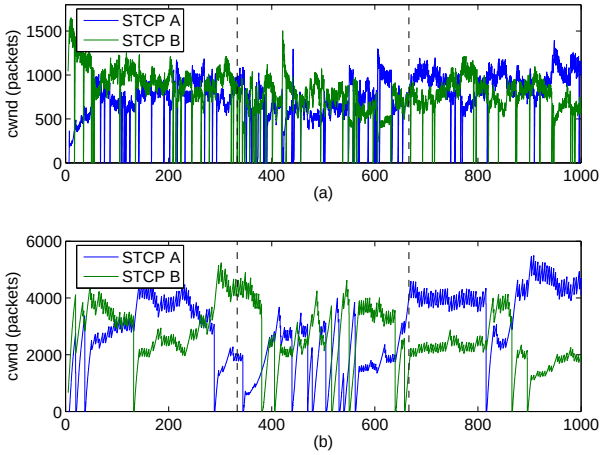



Fig. 10. STCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

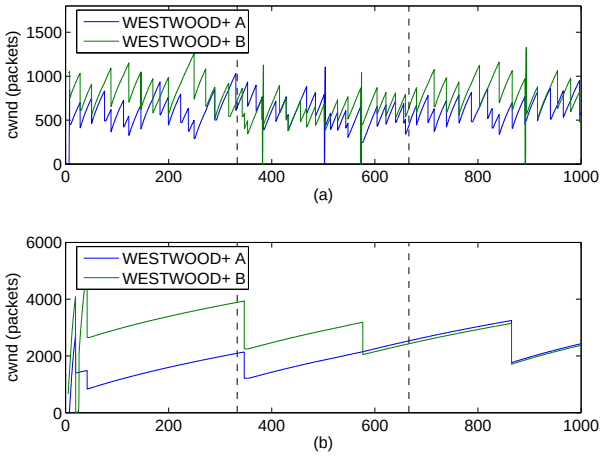



Fig. 11. Westwood+ TCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

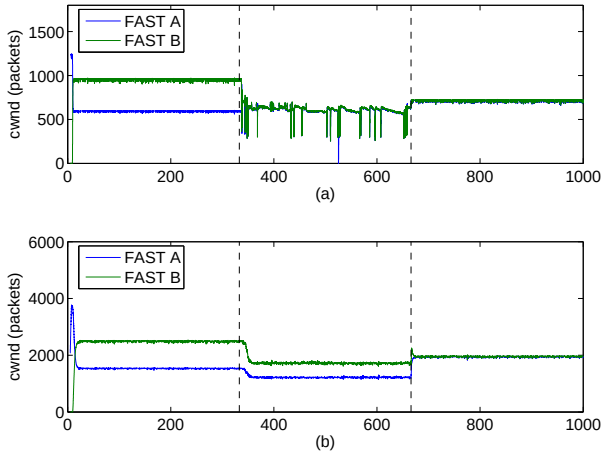



Fig. 12. FAST TCP: Congestion window evolution with Web traffi c enabled: (a) RTT=40ms and (b) RTT=160ms.

As far as regards SACK and Westwood+, both protocols are not affected by reverse traffic and timeout events are quite infrequent.

Also FAST TCP flows are not affected by timeout events; however looking at the congestion window behaviors, it is possible to observe that they do not share the bandwidth fairly: when RTT is 40ms the congestion windows of Sender A and B in the first phase of the simulation are stabilized to two distinct values (about 550 and 950 packets) indicating a high level of unfairness between the two flows. When the reverse connections start, the two congestion window converge to the same value and the congestion windows oscillates frequently. In the case of RTT=160ms, the unfairness between Sender A and Sender B persists also when the reverse traffic is active; moreover, as also shown in Figure 4, values of congestion windows during the phase with reverse traffic lead to low link utilization.

TABLE I

TIMEOUT EVENTS WITHOUT WEB TRAFFIC.

| TCP stack | RTT=40ms | | | RTT=160ms | | |
|---|---|---|---|---|---|---|
| Phase | 1 | 2 | 3 | 1 | 2 | 3 |
| SACK | 0 | 1 | 0 | 0 | 0 | 0 |
| HSTCP | 0 | 15 | 0 | 0 | 4 | 0 |
| H-TCP | 0 | 26 | 0 | 0 | 7 | 0 |
| BIC | 1 | 19 | 0 | 0 | 3 | 0 |
| STCP | 0 | 45 | 0 | 0 | 10 | 0 |
| Westwood+ | 0 | 5 | 0 | 0 | 1 | 0 |
| FAST | 0 | 0 | 0 | 0 | 0 | 0 |

The last metric we investigate is the number of timeout events that occur during the three phases of the simulations. Tables I and II report the number of timeout events with and without background Web traffic, respectively. When the Web traffic is off (Table I), timeouts occur only when the reverse traffic is active; SACK TCP, FAST TCP and Westwood+ experience a very small number of timeouts, whereas BIC, HSTCP, H-TCP and STCP experience a larger number of timeout events

TABLE II
TIMEOUT EVENTS WITH WEB TRAFFIC.

| TCP stack | RTT=40ms | | | RTT=160ms | | |
|---|---|---|---|---|---|---|
| Phase | 1 | 2 | 3 | 1 | 2 | 3 |
| SACK | 0 | 0 | 0 | 1 | 0 | 0 |
| HSTCP | 11 | 12 | 15 | 3 | 2 | 2 |
| H-TCP | 13 | 9 | 15 | 0 | 9 | 1 |
| BIC | 10 | 19 | 8 | 0 | 1 | 0 |
| STCP | 32 | 51 | 24 | 4 | 14 | 3 |
| Westwood+ | 0 | 4 | 1 | 1 | 0 | 0 |
| FAST | 0 | 1 | 0 | 0 | 0 | 0 |

that decreases when the RTT increases. STCP experiences the highest number of timeouts: 45 timeout events when RTT is 40ms and 10 timeouts when RTT is 160ms. When the Web traffic is on (Table II), timeouts occur during all simulation phases. For instance, in the case of STCP the number of timeouts in the three phases of the simulation is 32, 51 and 24 when RTT is 40ms, whereas when RTT is 160ms, the number of timeouts is much lower: 4, 14 and 3.

It is worth noticing that a high number of timeouts does not necessarily impact the performance of the protocols in terms of link utilization; however a high number of timeouts indicates that the protocol is highly unstable and, in case of congestion, its control mechanism is not able to decrease the congestion window as a protocol designed for congestion avoidance should be.

It is also worth noting that, when the reverse traffic is active, SACK TCP congestion control is able to achieve the same performance of NGTCP protocols in terms of link utilization but with a smaller packet retransmission probability and with a smaller number of timeout events. In particular, all protocols provide goodput values that are comparable to each other within a 10% margin with the only exception of FAST TCP during the phase with reverse traffic. However, the packet retransmission probabilities of SACK and Westwood+ are between one and two order of magnitude smaller than those of the other protocols. This indicates that all these NGTCPs, except from FAST TCP, are designed to be very aggressive and thus do not avoid congestion in the sense that classic Van Jacobson algorithm does. For these reasons, we believe that stability of these protocols still need to be tested in large scale scenarios with hundreds of connections.

### *C. Scenario with reverse traffic + background Web traffic + different RTTs*

This scenario is obtained by adding two more TCP flows in the forward direction (Connections between Sender-Receiver E and F of Figure 1) to the previous simulation scenario. The RTT experienced by connection A, B,C and D is 80ms, whereas RTT experienced by Connection E and F is 40ms. Connection A and B start at the beginning of simulation, whereas Connection E and F start around time 250s. Between 333s and 666s, the reverse traffic (Connection C and D) is active. At 750s, connections B and F are turned off. In the last 250 seconds of the simulation only connections A and E, which have different RTTs, are active. The purpose of this scenario is to investigate how bandwidth is allocated by different NGTCPs flows with different RTTs. Figure 13 plots the congestion window dynamics of connections A, B, E and F when different TCP stacks are employed. Figures 14, 15 and 16 show the goodput of the forward connections in the first 250 seconds, between 333 and 666 seconds and between 750 and 1000 seconds respectively. In the first figure, the bar plot depicts goodputs of connections A, B both having a RTT equal to 80ms. BIC and H-TCP provide a fair bandwidth sharing whereas other protocols do not. By comparing the goodputs with Figure 13, we notice that in case of STCP, HSTCP, SACK and Westwood+ TCP, the unfairness between the two flows is mainly due to a transient phase of the congestion windows. As far as regards FAST TCP, the unfairness is due to the setting of the congestion window. Figure 15 shows the goodputs of connections A, B (both with RTT=80ms) and E, F (both with RTT=40ms). FAST TCP and H-TCP connections obtain all the same goodput showing that both protocols are fair in sharing the bandwidth no matter the experienced RTT. In case of SACK and Westwood+ TCP, goodputs of connections with RTT=80ms are roughly one half of the goodput of connections with RTT40ms . Other protocols are highly unfair when RTTs are different. Same results are obtained in the last part of the simulation, when only connections A and E - with different RTT- are active. As it is shown in Figure 16, all protocols except from FAST TCP and H-TCP are not able to provide fair bandwidth sharing. In the case of STCP, for instance, the ratio between Connections A and E goodputs is about 1:17.

## IV. CONCLUSION

In this paper the effect of reverse traffic on new transport protocols designed for highspeed networks has been investigated. Simulation results show that all new protocols are fast enough to reach full link utilization and exhibit a remarkable window oscillation behavior in the presence of reverse traffic of the same nature, which is a normal network operating condition. All analyzed protocols suffer in some particular scenarios. For instance, FAST TCP performs very well in the scenario with different RTTs, providing fair bandwidth sharing. However, the same protocol is not able to fully exploit the bandwidth when the reverse traffic is active. In the presence of background Web traffic and reverse traffic, the number of timeouts and retransmissions experienced by NGTCP protocols is very high compared to standard TCP, which means that these protocol do not implement an efficient congestion avoidance phase such as the Van Jacobson algorithm does. This feature may impact network stability in the case of large scale scenarios and needs further investigation.

As a further work, the effect of the bottleneck buffer size on new transport protocols will be investigated [19].

## REFERENCES

[1] C. Jin, D. X. Wei and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," Proc. of INFOCOM 2004, March 2004, Hong Kong, China.

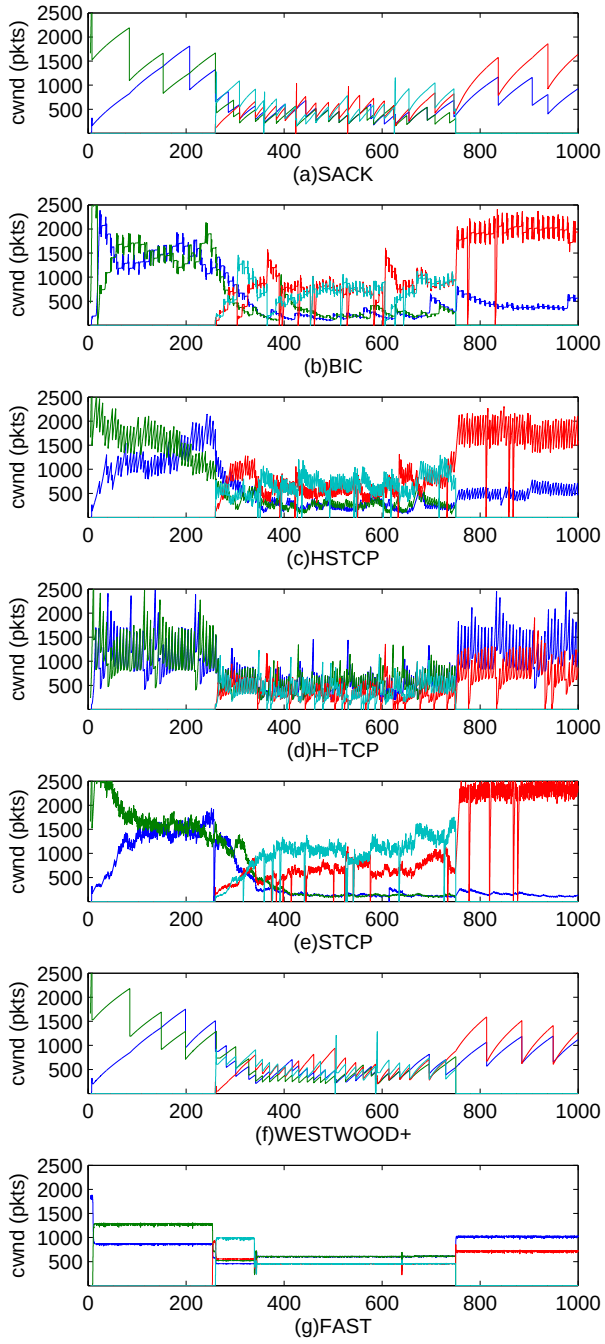



Fig. 13. Congestion window evolution with Web traffi c enabled.

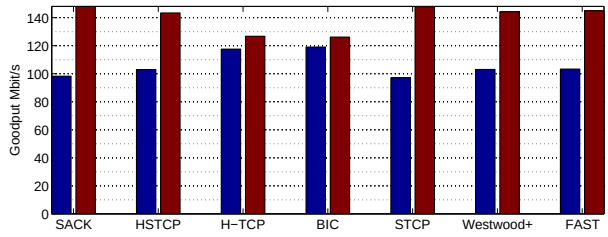



Fig. 14. Goodputs of Connection A and B during the fi rst 250 seconds.

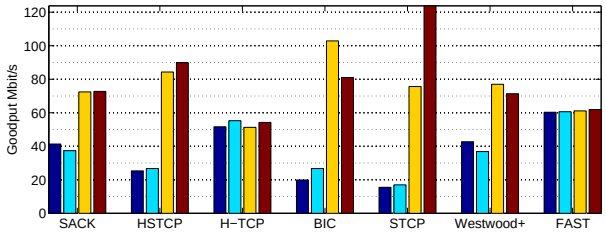



Fig. 15. Goodputs of Connection A, B, E and F between 333s and 666s.

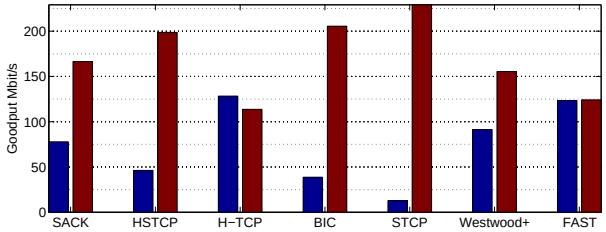



Fig. 16. Goodputs of Connection A and E in the last 250 seconds.


[2] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," Proc of PFLDnet 2003, February 2003, Geneva, Switzerland.

[3] Sally Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Experimental, December 2003.

[4] R.N. Shorten, D.J. Leith, "H-TCP: TCP for high-speed and long-distance networks" Proc. PFLDnet, Argonne, 2004.

[5] L. Xu, K. Harfoush, I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," Proc. of INFOCOM 2004, March 2004, Hong Kong, China.

[6] I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," Prof. of PFLDnet 2005, February 2005, Lyon, France.

[7] The network simulator. URL: www.isi.edu/nsnam/ns/

[8] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, R. Wang, "TCP Westwood: End-to-End Bandwidth Estimation for Effi cient Transport over Wired and Wireless Networks," Proceedings of ACM Mobicom, Rome, Italy, July 2001.

[9] L. A. Grieco, S. Mascolo "Performance evaluation and comparison of Westwood+, Vegas and New Reno TCP congestion control," ACM Computer Communication Review, April 2004.

[10] S. Mascolo, G Racanelli, "Testing TCP Westwood+ over Transatlantic Links at 10 Gigabit/Second rate," in proc. of PFLDNet 2005, Lyon, France, Feb. 2005.

[11] V. Jacobson, R. Braden, D. Borman, "TCP Extensions for High Performance," RFC 1323, May 1992.

[12] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing Router Buffers," Proc. of Sigcomm 2004, August 30 - Sept. 3, 2004, Portland, Oregon, USA.

[13] http://dsd.lbl.gov/ evandro/hstcp/simul/simul.html

[14] http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.htm

[15] http://www.hamilton.ie/net/index.htm

[16] T.Cui and L. Andrew, "FAST TCP simulator module for ns-2, version 1.1", available from http://www.cubinlab.ee.mu.oz.au/ns2fasttcp

[17] Villamizar, C. and Song C. (1995), "High Performance TCP in ANSNET", ACM Computer Communication Review, vol. 24, no. 5, pp. 45-60.

[18] Y. Li, D. Leith and R. N. Shorten, "Experimental Evaluation of TCP Protocols for High-Speed Networks," Hamilton Institute Technical report, available at http://www.hamilton.ie/net/eval/results_HI2005.pdf

[19] S. Mascolo, F. Vacirca, "Congestion Control and Sizing Router Buffers in the Internet, in Proc. of 44th Conference on Decision and Control, Sevilla, Spain, December 2005.

[20] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," IETF RFC 3742, March 2004.