TCP Westwood Simulation Studies in Multiple-Path Cases

M. Gerla, S. S. Lee, and G. Pau
Computer Science Department
University of California, Los Angeles
{gerla, sslee, gpau@cs.ucla.edu}

***Abstract*- Best-effort traffic has been the dominant traffic type in the Internet, and TCP has been playing the major role for carrying such traffic. Today, Internet is again moving toward more reliable and robust network services including QoS. Provisioning multiple paths as backup routes is one of the feasible solutions to designing fault-tolerant network. When multiple paths are selected, traffic transmission can be either flow-based or packet-based allocation to the multiple paths. The packet-based connection allocation usually shows a better load balancing effect and ease of flow management in the presence of network failures. However, it causes TCP segments to be distributed over delay-different multiple paths and to be delivered out-of-sequence. These out-of-sequence packet deliveries make TCP receiver generate many duplicate acknowledgments (ACKs), and TCP sender, in turn, misinterprets network congestion from the duplicate ACKs. In this paper, we expose the side effects of the multiple path deployment on TCP Reno and SACK performance. Also, based on simulation experiments, we shows that TCP Westwood effectively handles the multiple path situation and outperforms the other TCP flavors.**

I. Introduction

TCP has been the key transport layer protocol for best-effort traffic in the Internet. It has been repeatedly revised and enhanced over the past 3 decades. Mostly, the enhanced results were directed to the congestion control mechanisms. The essential TCP congestion control approach implicitly estimates network congestion from packet loss (i.e., from timeouts or duplicate ACKs). It adjusts the slow start threshold and the congestion window accordingly. [1], [2].

Recently, a new approach to TCP congestion control has been proposed in TCP Westwood [3]. TCP Westwood estimates available bandwidth by monitoring the returning ACKs. It uses the bandwidth estimates to set the slow start threshold and the congestion window. The benefits of this approach have been demonstrated via analysis, simulation, and actual Internet experiments [3], [4].

In this paper, we investigate the performance of TCP Westwood in a different scenario from those addressed in [3], [4]. The context is QoS provisioning and fault tolerant QoS support in the Internet. Specifically, in order to provide fault tolerant QoS, multiple paths can be provisioned and used in parallel [5]. When multiple paths are provisioned, application traffic can be distributed over the multiple paths either flow-based or packet-based according to different performance demands. When packets are distributed (scattered) over multiple paths and reordered at destinations, usually a better load balancing effect is shown due to the finer load granularity (i.e., packet level) [5], and also ease of flow management in the network layer can be achieved without need to remember path allocation of each flow. In conjunction with the multiple QoS path approach for reliable QoS provisioning, we must also assure that the data transmission protocols survive the multiple path environment. Mostly QoS-sensitive traffic is transmitted by UDP and is thus insensitive to out-of-sequence delivery over multiple paths unless the re-sequencing process introduces severe reconstruction delay. Conventional TCP, on the other hand, is much more sensitive to the out-of-sequence packet delivery because of the associated duplicate ACKs. These duplicate ACKs are interpreted as packet loss (although the packets have already been delivered to their destination). This continuous generation of duplicate ACKs not only forces the slow start threshold to a very small value but also forces the congestion window to fluctuate around the small slow start threshold. It prevents the slow start threshold and the window from growing again. Therefore, the end-to-end throughput becomes very low although no link congestion ever takes place. In contrast, TCP Westwood is able to maintain high link utilization and throughput since it samples incoming ACKs to not only detect packet loss but also estimate bandwidth. This explicit bandwidth estimation of TCP Westwood is not affected by repeated duplicate ACKs.

We show simulation-based experiments in which TCP Reno, SACK, and Westwood show different performance results. This paper is organized as follows. Section II presents an overview of conventional TCP congestion control mechanisms and their limitations in the multiple path environment. Section III explains the fundamental concepts of the TCP Westwood approach and its innovative features: bandwidth estimation and adaptation to the multiple path environment. Section IV shows various simulation experiments to prove the efficacy of TCP Westwood in producing high end-to-end throughput in multiple path environments.

II. The Conventional TCP Congestion Control Mechanisms and Limitations

In this section, we review the well-known TCP congestion control mechanisms briefly and discuss their limitations when applied to multiple path environment. There have been many different variants of TCP congestion controls developed and enhanced so far. In this paper, we mainly focus on Reno and Selective Acknowledgment (SACK) [6], [7], [8], [9].

A. Mechanism Overview

In principle, the idea of TCP congestion control is to make each TCP traffic source determine how much capacity is available in the network so that it knows how many packets it can safely have in transit. For this purpose, TCP maintains two important variables called *congestion window (cwnd)* and

slow start threshold (ssthresh) to probe the underlying network with the mechanism called *additive increase / multiplicative decrease* [2]. More specifically, TCP performs four different actions based on the current network information probed by packet losses or consecutive duplicate ACKs, which are *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery*. These congestion control phases are controlled by adjusting cwnd and ssthresh accordingly. Also, these congestion control phases are essential to all the TCP variants regardless of their specifications [6]. These essential control mechanisms are compactly summarized as in Fig. 1 and 2 [2].

1. cwnd $\leftarrow$ one segment and ssthresh $\leftarrow$ 65535 bytes.
2. Send segments of the minimum of cwnd and the receiver's advertised window.
3. When congestion occurs, ssthresh $\leftarrow$ cwnd / 2, and if the congestion is indicated by a timeout, cwnd $\leftarrow$ one segment (i.e., slow start).
4. When new data is acknowledged, cwnd $\leftarrow$ cwnd + 1 segment size if cwnd $\leq$ ssthresh, or cwnd $\leftarrow$ cwnd + (1 / cwnd) if cwnd $>$ ssthresh.

Fig. 1. The combined algorithm of slow start and congestion avoidance.

1. When the third duplicate ACK is received, ssthresh $\leftarrow$ cwnd / 2, retransmit the missing segment, and cwnd $\leftarrow$ ssthresh + 3 times the segment size.
2. Each time another duplicate ACK arrives, cwnd $\leftarrow$ cwnd + 1 segment size.
3. When a new ACK arrives, cwnd $\leftarrow$ ssthresh.

Fig. 2. The combined algorithm of fast retransmit and fast recovery.

These control phases have been proved via enormous research works over decades, and TCPs equipped with these mechanisms have been practically deployed in systems and showing expected performance. This is mainly because the underlying network topology is rather static than it was originally expected, and mostly packets are forwarded over single route without any severe out-of-sequence deliveries. Thus, both timeouts and duplicate acknowledgments are proper indications about congestion and packet losses. However, if packets are spread over multiple paths between source and destination, the packets are highly likely to arrive not in the order they were initially sent. Due to the nature of TCP (i.e., the ACK handling mechanism in Fig. 2, these out-of-sequence deliveries trigger acknowledgments to inform the source as if some packets were lost although the packets are not dropped in the network. This misleading information may cause drastic results in terms of link utilization and end-to-end throughput. More detailed case studies are presented in the next section.

B. Incorrect Interpretation of Duplicate Acknowledgments

In the multiple path environment, TCP has to deal with out-of-sequence packet deliveries in both data packet and acknowledgment processes. This phenomenon is a side effect of the delay asymmetry between the multiple paths. In this section, we will show the limitations of the TCP Reno and TCP SACK congestion avoidance mechanisms when a multiple path network is considered. We studied two different categories of multiple path scenarios:

1. *Full multiple path scenario* : both data packets and ACKs of the same TCP connection are scattered over multiple paths with different path characteristics shown in Fig. 3.
2. *Simple multiple path scenario* : data packets are scattered across multiple paths, and ACKs instead use the single shortest path as in Fig. 4.

First, we consider the the topology shown in Fig. 3, in which both paths from the source (node 0) to the destination (node 5) are used at the same time. When multiple paths are deployed, packets are expected to be spread over the multiple paths in a round-robin fashion. This premise applies to all simulation experiments in this paper. The round-trip time (RTT) of path P_0 (i.e., path along the nodes 0, 1, 2, 3, and 5) is 244 ms and the RTT of path P_1 (i.e., path along the nodes 0, 1, 4, 3, and 5) is 64 ms. Both considered paths have the same capacity of 10 Mbps and the total bottleneck bandwidth between the source and the destination is the sum of the two paths' capacities: 20 Mbps.

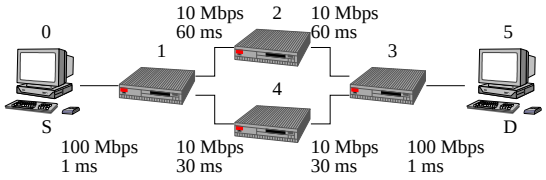

Fig. 3. Multiple path topology.

We now focus on the TCP generic behaviors in the considered topology and the configuration settings. In particular, we focus on the slow start phase studying how it behaves when multiple paths are enabled in the network. Note the fragment of the packet trace shown in Table I for a single TCP connection from the source node 0 to the destination node 5 in Fig. 3.

The table shows analytically how the ACK management process at the sender is affected by the delay asymmetry between the different paths. This triggers an enormous number of duplicate ACKs resulting that the congestion avoidance phase is started too early and the congestion windows never reaches the optimal size.

Now we move on to a simpler scenario considering multiple paths only for data packets and assuming the shortest single path for returning ACKs (Fig. 4). This produces a "serialization" of ACKs so that the connection suffers from the path delay asymmetry just for data packets.

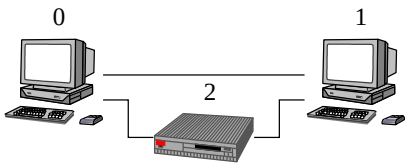

Fig. 4. Simple topology with two paths between the source and the destination. Links are all 1.5 Mbps and have 30-ms propagation delays.

In order to show the behavior of both TCP Reno and SACK congestion avoidance mechanisms in presence of multiple paths, we carried out a simulation using the scenario depicted in Fig. 4.

TABLE I

PACKET SEQUENCE FOR A SINGLE TCP CONNECTION IN THE SCENARIO DEPICTED IN FIG. 3

Event ♯	Source	Destination	Comments
1	send pktid(0)		cwnd = 1, ssthresh = 200
2		recv pktid(0)	
3		send ackid(0)	
4	recv ackid(0)		cwnd = 2
5	send pktid(1)		
6	send pktid(2)		
7		recv pktid(2)	pktid(2) has followed the shortest path P(1).
8		send ackid(0)	DUPACK ♯1 for pktid(0) → pktid(1) still in the network.
9		recv pktid(1)	pktid(1) finally arrived from P(0)
..	..	..	
..	..	..	
46	recv ackid(10)		cwnd = 6
47	send pktid(12)		
48	send pktid(13)		
49	send pktid(14)		
50	send pktid(15)		
51	send pktid(16)		
52		recv pktid(12)	
53		send ackid(12)	
54		recv pktid(14)	
55		send ackid(12)	DUPACK ♯1 pktid(12) → pktid(13) still in the network.
56		recv pktid(16)	
57		send ackid(12)	DUPACK ♯2 pktid(12) → pktid(13) still in the network.
58	recv ackid(12)		cwnd = 7
59	send pktid(17)		
60	send pktid(18)		
61	send pktid(19)		
62	recv ackid(12)		DUPACK ♯1
63		recv pktid(18)	
64		send ackid(12)	NOTE: DUPACK ♯3 pktid(12) → pktid(13) still in the network. NOTE: This is the third DUPACK for pktid(12) when it is received, TCP will trigger fast recovery.
65	recv ackid(8)		
66	recv ackid(11)		
67		recv pktid(13)	pktid(13) has followed the longer path P_0 → ♯3 DUPACK for pktid(12) has been sent.
68		send ackid(14)	
69		recv pktid(15)	
70		send ackid(16)	
71	recv ackid(14)		cwnd = 8
72	send pktid(20)		
73	send pktid(21)		
74	send pktid(22)		
75	recv ackid(12)		DUPACK ♯2
..	..	..	
..	..	..	

In Fig. 7 and 8, we show the general behaviors of TCP Reno in the first few seconds of the connection life. As expected from the previous analysis (Table I), the congestion window does not grow since the congestion avoidance phase is invoked too early. Moreover, whenever three duplicate ACKs are detected, the fast retransmit and fast recovery algorithms are triggered, and this keeps cwnd and ssthresh with unnecessarily low values.

III. TCP WESTWOOD CONGESTION CONTROL MECHANISMS

Since TCP Westwood was first introduced [3], it has brought many significant improvements and research aspects into the world. Its renowned performance has been proved via simulation and real testbed results for both wired and wireless networks [4]. Along with this strong background, we now investigate its performance over multiple paths.

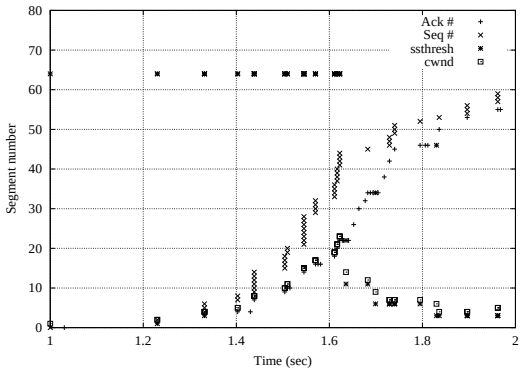

Fig. 5. The impact of continuous duplicate ACKs upon the slow start threshold and the congestion window in TCP Reno. The FTP connection starts at 1 sec and the initial slow start threshold is set to 64 packets.

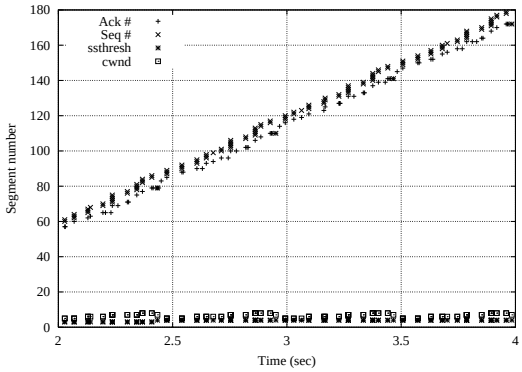

Fig. 6. Another example of the impact of continuous duplicate ACKs in TCP Reno. The slow start threshold and the congestion window do not have chances to increase.

A. Mechanism Overview

TCP Westwood considers the network as a "black box," without requiring from the network any support to the TCP congestion control scheme. The key innovative idea in TCP Westwood is that the TCP sender continuously computes the connection bandwidth estimate (BWE) which is defined as the share of bottleneck bandwidth available to the connection. BWE is equal to the rate at which data is delivered to the connection destination. The delivery rate estimate is based on information in the ACKs, and the rate at which the ACKs are received. After detecting a packet loss indication, which is mostly due to congestion but could also be due to link errors, the sender uses the estimated bandwidth information to properly set the congestion window and the slow start threshold. Packet loss symptoms used by a sender are the reception of 3 duplicate ACKs or a coarse timeout expiration [4]. The BWE as defined above is the rate actually achieved by the individual flow. When the bottleneck becomes saturated and packets are dropped, TCP Westwood selects congestion windows which enforce the current measured rates. In particular, assuming that a sender has determined the connection bandwidth estimate, it is used to properly set *cwnd* and *ssthresh* after a packet loss indication. In TCP Westwood, the congestion window dynamics during slow start and congestion avoidance are unchanged, which means that they increase exponentially and linearly, respectively, as in current TCP Reno [3].

A packet loss is indicated by the reception of 3 duplicate ac-

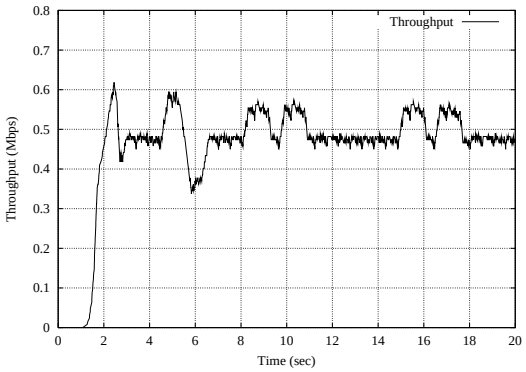

Fig. 7. The end-to-end throughput of the FTP connection using TCP Reno.

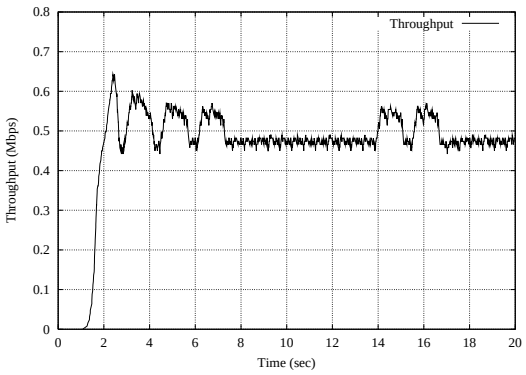

Fig. 8. The end-to-end throughput of the FTP connection using TCP SACK.

knowledgments (DUPACKs) or a coarse timeout expiration. In case the loss indication is 3 DUPACKs, TCP Westwood sets cwnd and ssthresh as follows:

```
if (3 DUPACKs are received)
    ssthresh = (BWE * RTTmin) / seg_size;
    if (cwnd > ssthresh) /* congestion avoid. */
        cwnd = ssthresh;
    endif
endif
```

In the pseudo-code, seg_size identifies the length of a TCP segment in bits. The reception of n DUPACKs is followed by the retransmission of the missing segment, as in the standard fast retransmit algorithm implemented in TCP Reno. Also, the window growth after the cwnd is reset to ssthresh follows the rules established in the fast retransmit algorithm (i.e., cwnd grows by one for each further ACK, and is reset to ssthresh after the first ACK acknowledging new data). During the congestion avoidance phase, we are probing for extra available bandwidth. Therefore, when n DUPACKs are received, it means that we have hit the network capacity (or that, in the case of wireless links, one or more segments were dropped due to sporadic losses). Thus, the slow start threshold is set equal to the window capable of producing the measured rate BWE when the bottleneck buffer is empty (namely, BWE $\times$ RTTmin). The congestion window is set equal to the ssthresh and the congestion avoidance phase is entered again to gently probe for new available bandwidth. The value RTTmin is set as the smallest round-trip time (RTT) estimated by TCP, using its own RTT estimation algorithm. Note that after ssthresh has been set, the congestion window is set equal to the slow start threshold only if cwnd $>$ ssthresh. It is possible that the current cwnd may be below threshold. This occurs after a timeout for example, when the window is dropped to 1 as discussed in the following section. During slow start, cwnd still features an exponential increase as in the current implementation of TCP Reno [4].

In case a packet loss is indicated by a timeout expiration, cwnd and ssthresh are set as follows:

```
if (coarse timeout expires)
    cwnd = 1;
    ssthresh = (BWE * RTTmin) / seg_size;
    if (ssthresh < 2)
        ssthresh = 2;
    endif;
endif
```

The rationale of the algorithm above is that after a timeout, cwnd and ssthresh are set equal to 1 and estimated bandwidth, respectively. Thus, the basic Reno behavior is still captured, while a speedy recovery is ensured by setting ssthresh to the value of the estimated bandwidth.

The bandwidth estimation which is the core of this innovative approach is computed using the information available in the ACKs. More precisely, the sender uses the following information: the ACK reception rate and the information ACKs which convey regarding the amount of data recently delivered to the destination. The detailed description of the TCP Westwood Bandwidth Estimation is thoroughly addressed in [3], [4].

B. Adaptation to Multiple Path Environment

As previously discussed, the main reason that the TCP variants stemming from the conventional approach (e.g., Reno or SACK) are not appropriate for multiple path environment is that they *blindly* treat duplicate ACKs as indications of packet losses and they keep pulling slow start threshold and congestion window down to unnecessarily small values. However, this is a quite inevitable circumstance since there is no explicit notification telling if duplicate ACKs are caused by packet losses or by out of sequence induced by multiple paths.

The multiple path issue can be resolved by TCP Westwood in a dexterous manner. We recall that with TCP Westwood, the traffic source keeps computing bandwidth estimate and sets the threshold and the window accordingly. Thus, TCP Westwood maintains ssthresh sufficiently high to represent the current bandwidth it measures as long as it receives ACKs and correctly derives bandwidth estimate. This relaxes the need to know the cause of duplicate ACKs.

In order to investigate how suitable for multiple path environment TCP Westwood is, we use the previous simple topology in Fig. 4 and demonstrate its capability. The simulation topology consists of two paths and we designed the source node 0 to distribute outgoing packets in a round-robin fashion as in Section II. Since the link capacities are the same, there must not any congested links at all. Thus, what we expect to see is that the end-to-end throughput becomes 3 Mbps (i.e., two 1.5 Mbps paths). Under the exactly same simulation conditions shown in Section II, we collected the end-to-end throughput of FTP traffic at the transport layer level with TCP Westwood and Fig. 9 shows the statistics.

As illustrated in the figure, TCP Westwood achieves the maximum possible end-to-end throughput. As Reno and SACK experience continuous generation of duplicate ACKs, as does TCP

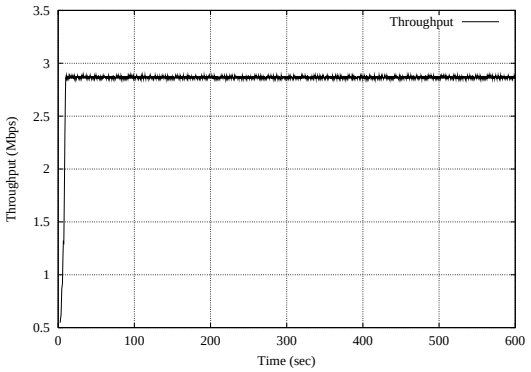

Fig. 9. End-to-end throughput achieved by TCP Westwood over the two paths.

Westwood. But the duplicate ACKs do not harm the TCP performance although they trigger TCP Westwood to adjust ssthresh and cwnd. They are instead adjusted to the currently measured bandwidth estimate (which is unaffected by duplicate ACKs). Thus, high end-to-end throughput is insured by the bandwidth estimating feature. Fig. 10 shows the bandwidth estimate measured by the traffic source with TCP Westwood.

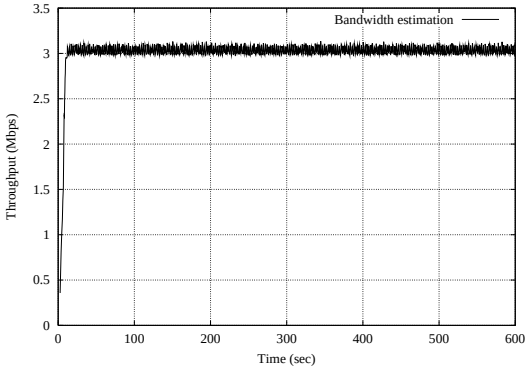

Fig. 10. End-to-end bandwidth estimation of TCP Westwood over the two paths.

IV. Simulation Experiments

Based on the previous assessment of TCP Westwood suitability to multiple path environment, we present more comprehensive simulation studies in this section.

In order to make the simulation experiments practically more meaningful, we designed all the simulation scenarios with multiple paths for both data packets and returning ACKs. When multiple paths are deployed from a source to a destination, it is likely to have multiple paths deployed for the opposite direction as well.

All the simulation experiments are executed with a network simulator named SENSE [10] which is a discrete-time event-driven network simulation platform and much specialized for the fault tolerant QoS provisioning simulation experiments as introduced in [11], [5].

A. Performance Verification with a Small Topology

The first set of simulation experiments described in this section investigates the influence of different number of multiple

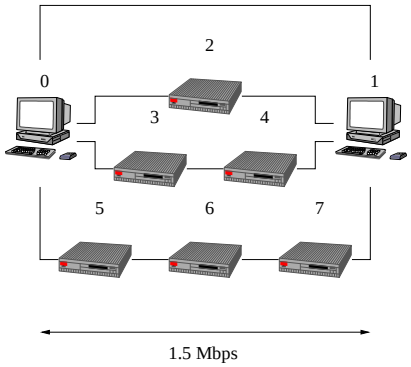

Fig. 11. Simulation topology used to provide multiple paths between the source and the destination. Link bandwidth is 1.5 Mbps for all the links. The physical propagation delay of each link is parameterized with a variable, and different values for the variable are used.

paths and their physical link delay differences. For this purpose, a relatively simple network topology is used and Fig. 11 presents the topology. The traffic model is FTP which has infinite backlog of data packets. The traffic source and the destination are node 0 and 1 respectively. As depicted in the figure, there are four different possible paths between the source and the destination. Each link in the network is a T1 line (i.e., 1.5 Mbps), and physical link propagation delay is parameterized with a variable. The variable holds a link delay from 10 ms to 60 ms. Thus, we generated totally 18 different simulation scenarios with different number of multiple paths (i.e., from 2 to 4) and with six different link delay choices. With these settings, we scheduled a single FTP connection between the source and the destination with different TCP flavors: Reno, SACK, and Westwood.

As in the simulation cases of the previous sections, the topology and traffic pattern do not produce any network congestion. If TCP fully utilizes the given number of multiple paths, the end-to-end throughput can be approximated by the sum of the capacities of the multiple paths. However, conventional TCP versions suffer from the multiple paths and large delay differences. As the number of multiple paths and their delay differences become larger, more out-of-sequence packet deliveries take place causing many duplicate ACKs. TCP Westwood is expected not to suffer from this phenomenon for the previously mentioned reason.

Fig. 12, 13, and 14 present the throughput results. The number of multiple paths varies from 2 to 4 as discussed (i.e., n multiple paths mean the first n shortest paths between the source and the destination), and clearly this affects negatively the Reno and SACK performance, while it does not harm Westwood performance. Note that the link delay values in the graphs apply to each single link (i.e., 10 ms with 2 paths in the topology mean that the first two shortest paths are used with each link being 10-ms long).

The superior performance of TCP Westwood stems from the capability of estimating bandwidth as previously described. The bandwidth estimates are depicted in Fig. 15. As shown in the figure, when the number of multiple paths increases, the TCP Westwood sender correctly captures the bandwidth regardless of the increases of the link delays.

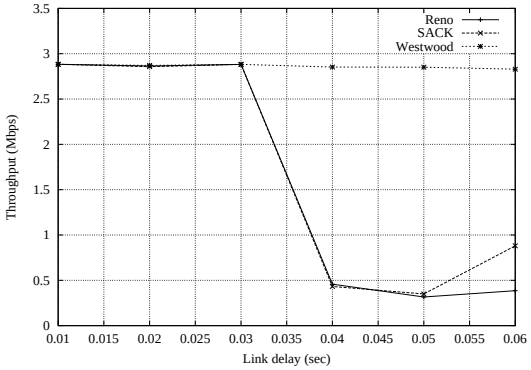

Fig. 12. Throughput comparison of Reno, SACK, and Westwood when only two paths are used.

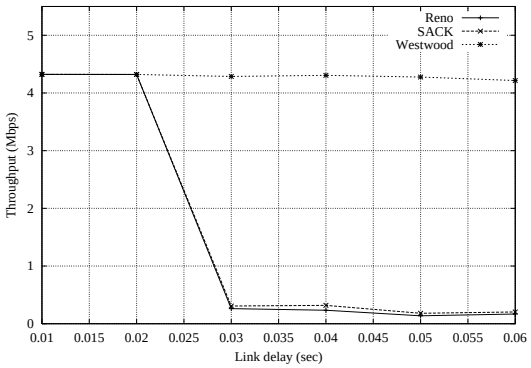

Fig. 13. Throughput comparison of Reno, SACK, and Westwood when only three paths are used.

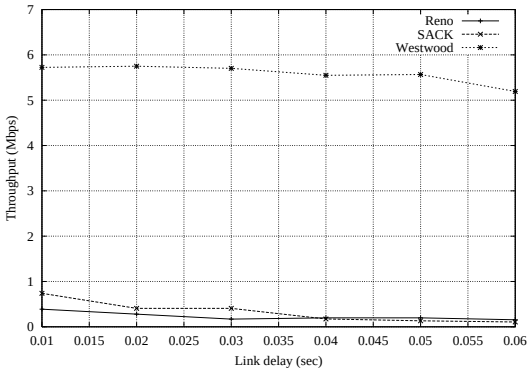

Fig. 14. Throughput comparison of Reno, SACK, and Westwood when only four paths are used.

B. More Generic Simulations with Random Connection Distribution

As shown previously, TCP Westwood outperforms other TCPs with respect to achieving higher throughput in a multiple path environment. Next, we randomly distribute TCP connections to multiple paths. For this purpose, we built an 8×8 grid topology and designated the inside 36 (6×6) nodes to be traffic source and destination as in Fig. 16. The reason for excluding the nodes on the edges and the corners in the topology is to let all the nodes have four possible alternate paths since the nodes at the corners can have only two possible disjoint paths and the

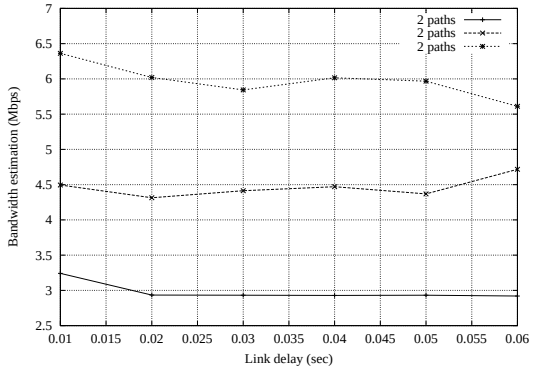

Fig. 15. Bandwidth estimation monitored by TCP Westwood over the different number of multiple paths.

edge nodes have three. Thus, excluding the edge and the corner nodes, we expect that all the nodes in the topology can have fair conditions in terms of the number of multiple paths.

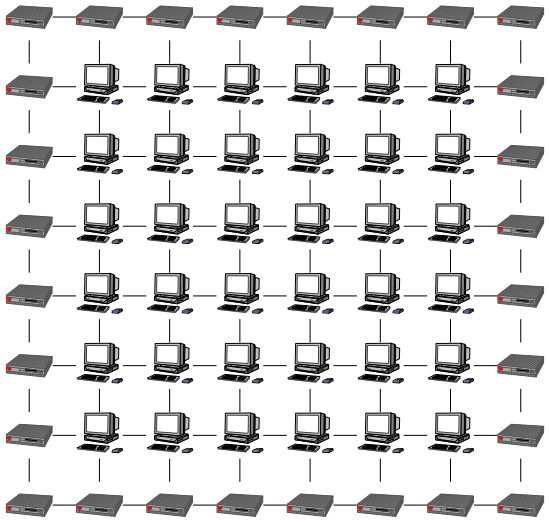

Fig. 16. 8×8 grid topology. The links are 1.5 Mbps, and the physical propagation delays of the links are parameterized with values from 10 ms to 60 ms. Only the inside nodes (6×6) are picked to be sources and destinations.

Over the 8×8 grid topology (i.e., 6×6 sources and destinations), FTP connections are generated by selecting their sources and destinations randomly. Connections arrive with an interval of 2 seconds and last for 180 seconds. The interval and duration of the connections are exponentially distributed. With these settings, approximately 300 connections are generated over the simulation time (i.e., 600 seconds). Each source destination pair has 4 multiple paths.

We measure average throughput of the connections over the simulation time. As in the previous set of experiments, the link delays are variable from 10 ms to 60 ms. However, the experiment scenarios are likely to have longer paths due to the topology properties than the previous set of experiments which had at most 4 links per path. Reno, SACK, and Westwood throughput results are depicted in Fig. 17.

As shown in the figure, Reno and SACK suffer from multiple path syndrome and get worse as the link delays increase with SACK minimally producing a little better performance over

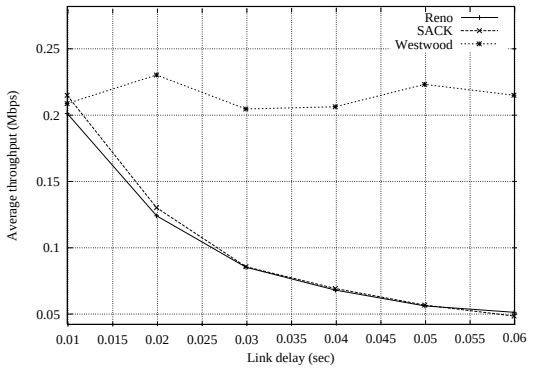

Fig. 17. Average throughput comparison of Reno, SACK, and Westwood.

Reno. Westwood remains quite stable and insensitive to the delay changes.

V. Conclusion

When multiple paths are deployed for fault tolerance, load balancing, security, etc., generic TCP variants do not survive the multiple path environment. This is primarily because TCP blindly treats duplicate ACKs as indications of packet losses and incipient network congestion even though their corresponding data packets have been correctly (but out-of-sequence) delivered to their destination. This limitation is avoided by the bandwidth estimating feature of TCP Westwood with which the traffic source can estimate bandwidth by intelligently monitoring ACKs and adjusting ssthresh and cwnd more adequately to represent the current network condition. This new congestion control mechanism outperforms the "blind" approach of conventional TCPs to handle threshold and window. Consequently, TCP Westwood is deemed to be the preferred approach for an Internet featuring multiple paths for load balancing and for fast response fault tolerant QoS provisioning. For a similar context, recently another TCP variant has been introduced (TCP DSACK). As one of the future works, we will investigate the performance comparison and possible conjunction with DSACK and Westwood.

References

[1] J. Postel. Transmission Control Protocol. Request for Comments 793, Internet Engineering Task Force, September 1981.

[2] W. R. Stevens. *TCP/IP Illustrated: Volume 1; The Protocols*. Addison-Wesley Profesional Computing Series, Addison-Wesley, Reading, Massachusetts, 1996.

[3] C. Casetti, M. Gerla, S. S. Lee, S. Mascolo, and M. Sanadidi. TCP with Faster Recovery. In *IEEE Milcom*, 2000.

[4] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *IEEE Mobicom*, July 2001.

[5] Scott Seongwook Lee and Mario Gerla. Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths. *Lecture Notes in Computer Science*, 2092:155–, 2001. International Workshop on Quality of Service (IWQoS).

[6] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. 26(3):5–21, July 1996.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. Request for Comments 2018, Internet Engineering Task Force, October 1996.

[8] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. Request for Comments 2883, Internet Engineering Task Force, July 2000.

[9] Sally Floyd. A Report on Recent Developments in TCP Congestion Control. 39(4), April 2001.

[10] Scott Seongwook Lee. Simulation Environment for Network System Enhancement (SENSE). http://www.cs.ucla.edu/~sslee/sense.

[11] Alex Dubrovsky, Mario Gerla, Scott Seongwook Lee, and Dirceu Cavendish. Internet QoS Routing with IP Telephony and TCP Traffic. In *Proc. of ICC*, June 2000.