TCP Congestion Control Comparison

A. Esterhuizen and A.E. Krzesinski
Department of Mathematical Sciences
University of Stellenbosch, 7600 Stellenbosch, South Africa
aek1@cs.sun.ac.za
Tel.: +27 21 808 4232 Fax: +27 21 882 9865

Abstract— **This paper investigates the effects that different TCP variants have on each other. The TCP variants differ in the congestion control algorithms they employ. The congestion control algorithms determine how much network traffic is generated by TCP at any one time, and aims to prevent a TCP connection from over utilising the network. We investigate the different congestion control algorithms that are included as loadable modules in the Linux kernel, and we present several experiments to investigate how these congestion control algorithms compete for network resources. We show that some TCP variants can co-exist, whilst others use excessive bandwidth, potentially smothering competing TCP connections.**

I. Introduction

The *Transmission Control Protocol* (TCP) [5] provides a reliable, connection-oriented transport protocol for transaction-oriented applications. TCP is used by almost all of the application protocols found on the Internet today, as most of them require a reliable, error-correcting transport layer to ensure that data are not lost or corrupted.

TCP controls how much data it transmits over a network by utilising a sender-side *congestion window* and a receiver side *advertised window*. TCP cannot send more data than the congestion window allows, and it cannot receive more data than the advertised window allows [2]. The size of the congestion window depends upon the instantaneous congestion conditions in the network. When the network experiences heavy traffic conditions, the congestion window is small. When the network is lightly loaded the congestion window becomes larger. How and when the congestion window is adjusted depends on the form of congestion control that the TCP protocol uses.

Congestion control algorithms rely on various indicators to determine the congestion state of the network. For example, packet loss is an implicit indication that the network is overloaded and that the routers are dropping packets due to limited buffer space. Routers can set flags in a packet header to inform the receiving host that congestion is about to occur [16]. The receiving host can then explicitly inform the sending host to reduce its sending rate. Other congestion control methods include measuring packet round trip times (RTTs) and packet queuing delays Some congestion control mechanisms allow for unfair usage of network bandwidth, while other congestion control mechanisms are able to share bandwidth equally.

Several congestion control mechanisms are available for use by the Linux kernel namely: TCP-HighSpeed (H-TCP), TCP-Hybla, TCP-Illinois, TCP Low Priority (TCP-LP), TCP-Vegas, TCP-Reno, TCP Binary Increase Congestion (TCP-BIC), TCP-Westwood, Yet Another Highspeed TCP (TCP-YeAH), TCP-CUBIC and Scalable TCP. The Linux socket interface allows the user to change the type of congestion control a TCP connection uses by setting the appropriate socket option.

Comparisons of TCP-Reno, TCP-Vegas and TCP-Westwood have been reported (see for example [15], [6], [8] and the references therein) where the experiments were conducted on testbeds or using ns2 simulations. Our intention is to compare many, readily available, significant TCP variants.

The remainder of this paper is organised as follows. Section II describes the various TCP variants. A description of the experiments is given in Section III. The results of the experiments are discussed in Section IV. Section V summarises the presented results.

II. TCP variants investigated

Much of the information in this Section was obtained from `en.wikipedia.org`. The source references are listed in the text.

TCP-Reno uses slow start, congestion avoidance, and fast retransmit triggered by triple duplicate ACKs. Reno uses packet loss to detect network congestion [1].

TCP-BIC. The Binary Increase Congestion (BIC) control is an implementation of TCP with an optimized congestion control algorithm for high speed networks with high latency. BIC has a unique congestion window algorithm which uses a binary search algorithm in an attempt to find the largest congestion window that will last the maximum amount of time [17].

TCP-CUBIC is a less aggressive and more systematic derivative of TCP-BIC, in which the congestion window is a cubic function of time since the last packet loss [9], with the inflection point set to the window prior to the congestion event. There are two components to window growth. The first is a concave portion where the window quickly ramps up to the window size as it was before the previous congestion event. Next is a convex growth where CUBIC probes for more bandwidth, slowly at first then very rapidly. CUBIC spends a lot of time at a plateau between the concave and convex growth region which allows the network to stabilize before CUBIC begins looking for more bandwidth.

HighSpeed TCP (H-TCP) is a modification of the TCP-Reno congestion control mechanism for use with TCP connections with large congestion windows. H-TCP is a loss-based algorithm, using additive-increase/multiplicative-decrease to control the TCP congestion window [7]. It is one of many TCP congestion avoidance algorithms which seeks to increase the aggressiveness of TCP on high bandwidth-delay product (BDP) paths, while maintaining 'TCP friendliness' for small BDP paths. H-TCP increases its aggressiveness (in particular, the rate of additive increase) as the time since the previous loss increases [12]. This avoids the problem encountered by TCP-BIC of making flows more aggressive if their windows are already large. Thus new flows can be expected to converge to fairness faster under H-TCP than TCP-BIC.

TCP-Hybla was designed with the primary goal of counteracting the performance unfairness of TCP connections with longer RTTs. TCP-Hybla is meant to overcome performance issues encountered by TCP connections over terrestrial and satellite radio links. These issues stem from packet loss due to errors in the transmission link being mistaken for congestion, and a long RTT which limits the size of the congestion window [4].

TCP-Illinois is targeted at high-speed, long-distance networks. TCP-Illinois is a loss-delay based algorithm, which uses packet loss as the primary congestion signal to determine the direction of window size change, and uses queuing delay as the secondary congestion signal to adjust the pace of window size change [13].

TCP Low Priority (TCP-LP) is a congestion control algorithm whose goal is to utilize only the excess network bandwidth as compared to the 'fair share' of bandwidth as targeted by TCP-Reno. The key mechanisms unique to TCP-LP congestion control are the use of one-way packet delays for congestion indications and a TCP-transparent congestion avoidance policy [11].

TCP-Vegas emphasizes packet delay, rather than packet loss, as a signal to determine the rate at which to send packets. Unlike TCP-Reno which detects congestion only after it has happened via packet drops, TCP-Vegas detects congestion at an incipient stage based on increasing RTT values of the packets in the connection. Thus, unlike Reno, Vegas is aware of congestion in the network before packet losses occur [1].

The Vegas algorithm depends heavily on the accurate calculation of the Base RTT value. If it is too small then the throughput of the connection will be less than the bandwidth available, while if the value is too large then it will overrun the connection. Vegas and Reno cannot co-exist. The performance of Vegas degrades because Vegas reduces its sending rate before Reno as it detects congestion earlier and hence gives greater bandwidth to co-existing TCP-Reno flows.

TCP-Westwood is a sender-side-only modification to TCP Reno that is intended to better handle large bandwidth-delay product paths with potential packet loss due to transmission or other errors, and with dynamic load. TCP Westwood relies on scanning the ACK stream for information to help it better set the congestion control parameters namely the Slow Start Threshold `ssthresh`, and the Congestion Window `cwin` [14]. TCP-Westwood estimates an 'eligible rate' which is used by the sender to update `ssthresh` and `cwin` upon loss indication, or during its 'agile probing' phase which is a proposed modification to the slow start phase. In addition, a scheme called Persistent Non Congestion Detection was devised to detect a persistent lack of congestion and induce an agile probing phase to utilize large dynamic bandwidth.

Yet Another Highspeed TCP (TCP-YeAH) is a sender-side high-speed enabled TCP congestion control algorithm which uses a mixed loss/delay approach to compute the congestion window [3]. The goal is to achieve high efficiency, a small RTT and Reno fairness, and resilience to link loss while keeping the load on the network elements as low as possible.

Scalable TCP is a simple change to the traditional TCP congestion control algorithm (RFC2581) which dramatically improves TCP performance in high speed wide area networks. Scalable TCP changes the algorithm to update TCP's congestion window to the following: `cwnd:=cwnd+0.01` for each ACK received while not in loss recovery and `cwnd:=0.875*cwnd` on each loss event [10].

III. Experiments performed

Experiments were carried out using client and server programs. A host running a client program generates data which are sent over the network to a host running the server program. The server receives data from multiple clients. Each client uses a different congestion control algorithm. The amount of data received from the clients is measured in megabits per second and is presented as a graph which shows how much bandwidth the client utilises. The average of a client's bandwidth measurements determines it's bandwidth usage. In these experiments, the highest possible throughput on the network is slightly over 100 Mbps.

The experiments were run on the Stellenbosch University local area network. All hosts involved were located on the same network segment. The TCP segments did not traverse any routers. The intention was to initially keep the network topology as simple as possible. If several TCP variants could not co-exist on a local area network then they would not be viable on a long haul network.

All experiments were carried out in the same manner, and all TCP variants were tested in pairs. Host A (a client) transmits data to host B (the server), after which host C (another client) starts to send data to host B. Thus the two client hosts do not begin their data transmissions simultaneously, there is always a short delay between one client beginning its transmission and the next client beginning its transmission. The first client to start transmission utilises the network without restraint, and is only required to share bandwidth when an additional client starts transmission. This was found to have an impact on the behaviour of certain TCP variants, as their bandwidth utilisation depends upon the order in which they were started (see below).

Each TCP congestion control algorithm was tested against all the other TCP algorithms, including itself.

TABLE I
THROUGHPUT COMPARISON OF TCP VARIANTS.

	Vegas	HTCP	Westwood	Illinois	BIC	Scalable	HighSpeed	YeAH	LP	Hybla	Reno	CUBIC	
Vegas	49,50	10,95	8,95	6,97	13,91	8,95	9,95	23,95	10,95	10,94	9,94	10,94	Vegas
HTCP	95,10	52,52	19,86	38,65	41,62	41,64	64,40	64,40	65,38	66,37	66,37	67,37	HTCP
Westwood	95, 8	86,19	46,57	69,34	84,19	80,23	89,15	89,14	90,13	90,14	90,13	91,13	Westwood
Illinois	97, 6	65,38	34,69	13,89	93,10	88,22	94, 9	93,10	95, 7	70,33	95,8	94,10	Illinois
BIC	91,13	62,41	19,84	53,51	54,49	55,51	73,30	73,30	74,29	75,28	75,28	77,26	BIC
Scalable	95, 8	62,42	23,80	50,53	51,55	51,52	71,32	72,31	76,27	74,29	77,26	77,26	Scalable
HighSpeed	95, 9	40,64	15,89	36,68	30,73	32,71	54,50	58,45	61,42	63,40	62,40	64,40	HighSpeed
YeAH	95,23	40,64	19,84	37,67	30,73	31,72	45,58	52,51	53,50	57,46	54,49	54,49	YeAH
LP	95,10	38,65	13,90	33,70	29,74	27,76	42,61	50,53	53,50	54,48	50,53	55,47	LP
Hybla	94,10	37,66	14,90	33,70	28,75	29,74	40,63	46,57	48,54	52,51	49,54	53,50	Hybla
Reno	94, 9	37,66	15,89	31,73	28,75	26,77	40,62	49,54	53,50	54,49	51,52	54,49	Reno
CUBIC	94,10	37,67	13,91	33,70	26,77	26,77	40,64	49,54	47,55	50,53	49,54	47,55	CUBIC

Various TCP congestion control algorithms are available in the Linux kernel as modules. One can force Linux to use a specific congestion control algorithm, but one does not wish to be limited to only one type of congestion control algorithm at any time. Fortunately, Linux makes provision for this. The socket interface allows one to set socket options that allows the use of a different TCP congestion control algorithm for each TCP socket that is created. Superuser rights are required to select the congestion control algorithm and to set the socket options.

IV. RESULTS

Table I presents the average throughputs of the TCP variants. Thus when a Westwood TCP connection is opened followed by an HTCP connection, Westwood attains an average throughput of 86 Mbps and HTCP 19 Mbps. When an HTPC connection is opened followed by a Westwood connection, the throughputs remain the same, HTCP averages 19 Mbps and Westwood 86 Mbps. When an Illinois connection is opened followed by a BIC connection, Illinois averages 93 Mbps and BIC 10 Mbps. But when a BIC connection is opened followed by an Illinois connection, BIC averages 53 Mbps and Illinois 51 Mbps. Table I shows (the darkly shaded entries) that the throughput of TCP-Illinois sometimes depends upon whether Illinois was started before or after the competing TCP connection. Table I shows (the lightly shaded entries) that many TCP variants cannot co-exist, but some (the unshaded entries) put.

Table II shows that HTCP can co-exist with itself. HTCP has a lower throughput than Illinois, BIC, Westwood, and Scalable. HTCP has a higher throughput than Hybla, LP, Vegas, Reno, YeAH, CUBIC and Highspeed. Against these it maintains an average throughput in the range of 64 to 67 Mbps. The exception is Vegas, against which HTCP has an average throughput of about 95 Mbps.

Table III shows that Hybla can co-exist with itself, LP, Reno, CUBIC, and YeAH. Hybla performs badly against Illinois, BIC, Westwood, and Scalable. Vegas cannot co-exist with Hybla.

TABLE II
HTCP VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
HTCP	Other TCP variants	
52	52	HTCP
19	86	Westwood
38	65	Illinois
41	62	BIC
41	64	Scalable
64	40	YeAH
64	40	Highspeed
65	38	LP
66	37	Hybla
66	37	Reno
67	37	CUBIC
95	10	Vegas

TABLE III
HYBLA VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Hybla	Other TCP variants	
46	57	YeAH
48	54	LP
49	54	Reno
52	51	Hybla
53	50	CUBIC
14	90	Westwood
28	75	BIC
29	74	Scalable
33	70	Illinois
37	66	HTCP
40	63	Highspeed
94	10	Vegas

Table IV shows that Illinois is TCP unfriendly and cannot fairly share bandwidth with any of the other TCP variants that were investigated. The only variant to maintain a higher average throughput than Illinois is Westwood. Illinois cannot co-exist even with itself, something that all the TCP variants are capable of.

TABLE IV
ILLINOIS VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Illinois	Other TCP variants	
13	89	Illinois
34	69	Westwood
65	38	HTCP
70	33	Hybla
88	22	Scalable
93	10	BIC
93	10	YeAH
94	9	CUBIC
94	9	Highspeed
95	7	LP
95	8	Reno
97	6	Vegas

Section III disclosed that in some cases the order in which clients start their transmission influences the bandwidth utilised. This applies to TCP Illinois. Fig. 1 shows that if TCP-Illinois is allowed to begin transmission, and afterwards a TCP-Yeah client begins transmitting data, the Illinois connection maintains a high average throughput.

Fig. 1. Illinois begins transmitting 10 seconds before YeAH. Illinois maintains an average throughput of 93 Mbps, while YeAH maintains a low average throughput of 10 Mbps.

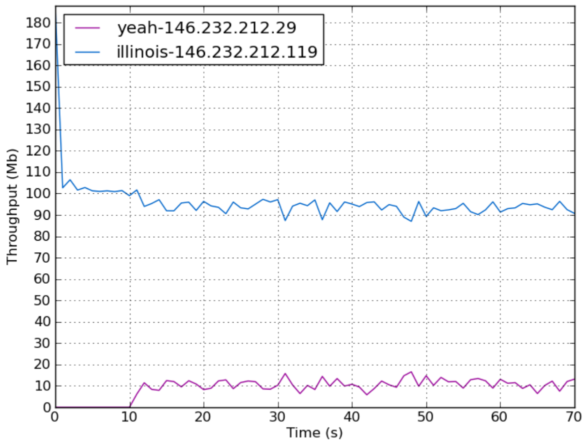

Fig. 2 shows that if the order of transmission is reversed, TCP-Yeah attains a much higher average throughput, although still less than TCP-Illinois.

Our experiments show that a TCP-Illinois connection, if started first, has a higher throughput than other TCP connections that are started later, with the exception of Westwood and Vegas, whose mean throughput remains the same regardless of the order in which they start transmitting.

Table V shows that LP can co-exist with itself, Hybla, Reno, YeAH, and CUBIC. As is the case with Hybla, LP fares badly against Illinois, BIC, Westwood, and Scalable.

Table VI shows that TCP-Vegas can co-exist with itself. Vegas performs poorly against all the other TCP variants,

Fig. 2. YeAH begins transmitting before Illinois. YeAH now has an average throughput of 37 Mbps, whilst Illinois has an average throughput of 67 Mbps.

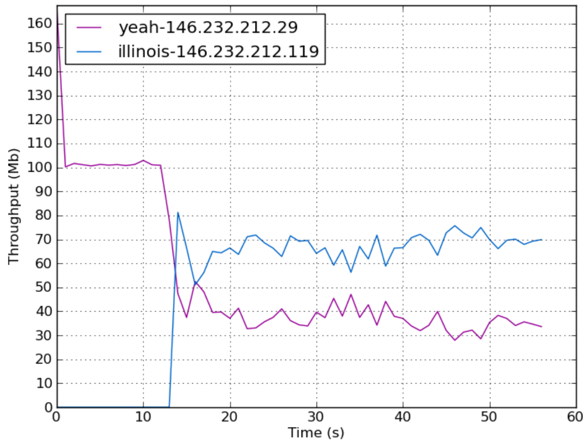

TABLE V
LP VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
LP	Other TCP variants	
50	53	Reno
50	53	YeAH
53	50	LP
54	48	Hybla
55	47	CUBIC
13	90	Westwood
27	76	Scalable
29	74	BIC
33	70	Illinois
38	65	HTCP
42	61	Highspeed
95	10	Vegas

TABLE VI
VEGAS VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Vegas	Other TCP variants	
49	50	Vegas
6	97	Illinois
8	95	Westwood
8	95	Scalable
9	94	Reno
9	95	Highspeed
10	95	HTCP
10	94	Hybla
10	95	LP
10	94	CUBIC
13	91	BIC
23	95	YeAH

achieving an average throughput of between 8 and 13 Mbps, depending on which TCP variant it competes against. TCP-Vegas cannot co-exist with other TCP variants.

Table VII shows that TCP-Reno can co-exist with itself,

TABLE VII

RENO VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Reno	Other TCP variants	
49	54	YeAH
51	52	Reno
53	50	LP
54	49	CUBIC
54	49	Hybla
15	89	Westwood
26	77	Scalable
28	75	BIC
31	73	Illinois
37	66	HTCP
40	62	Highspeed
94	9	Vegas

Hybla, LP, YeAH, and CUBIC. Again, as is that case with with Hybla and LP, Reno fares badly against Illinois, BIC, Westwood, and Scalable.

TABLE VIII

BIC VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
BIC	Other TCP variants	
53	51	Illinois
54	49	BIC
55	51	Scalable
19	84	Westwood
62	41	HTCP
73	30	YeAH
73	30	Highspeed
74	29	LP
75	28	Hybla
75	28	Reno
77	26	CUBIC
91	13	Vegas

Table VIII shows that BIC can co-exist with itself, Scalable and Illinois. BIC has a high average throughput when compared to Hybla, LP, Vegas, Reno, CUBIC and Highspeed. BIC can co-exist with Illinois, but Illinois cannot co-exist with BIC: it depends upon the order in which the BIC and Illinois streams are started.

Table IX shows that Westwood is very aggressive. The average throughput of Westwood is almost always above 80 Mbps, allowing very little bandwidth for competing TCP connections. The only exception is Westwood which can co-exist with itself.

Table X shows that TCP-YeAH can co-exist with itself, Hybla, LP, Reno, CUBIC, and to a lesser degree, Highspeed. As was the case with Hybla, LP, and Reno TCP-YeAH fares badly against Illinois, BIC, Westwood, and Scalable.

Table XI shows that CUBIC can co-exist with itself, Hybla, LP, Reno, and YeAH. CUBIC fares badly against Illinois, BIC, Westwood, and Scalable.

TABLE IX

WESTWOOD VS. OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Westwood	Other TCP variants	
46	57	Westwood
69	34	Illinois
80	23	Scalable
84	19	BIC
86	19	HTCP
89	14	YeAH
89	15	Highspeed
90	13	Reno
90	14	Hybla
91	13	CUBIC
90	13	LP
95	8	Vegas

TABLE X

YEAH VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
YeAH	Other TCP variants	
45	58	Highspeed
52	51	YeAH
53	50	LP
54	49	Reno
54	49	CUBIC
57	46	Hybla
19	84	Westwood
30	73	BIC
31	72	Scalable
37	67	Illinois
40	64	HTCP
95	23	Vegas

TABLE XI

CUBIC VERSUS OTHER TCP VARIANTS.

Mean throughput (Mbps)		
CUBIC	Other TCP variants	
47	55	CUBIC
47	55	LP
49	54	Reno
49	54	YeAH
50	53	Hybla
13	91	Westwood
26	77	BIC
26	77	Scalable
33	70	Illinois
37	67	HTCP
40	64	Highspeed
94	10	Vegas

Table XII shows that Highspeed shares bandwidth well with itself and reasonably well with YeAH. TCP variants like Illinois, Westwood, BIC, and Scalable tend to use most of the bandwidth when competing against Highspeed.

Table XIII shows that Scalable can co-exist with itself, BIC and Illinois. As noted previously, the behaviour of Illinois

TABLE XII

HIGHSPEED VS. OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Highspeed	Other TCP variants	
54	50	Highspeed
58	45	YeAH
15	89	Westwood
30	73	BIC
32	71	Scalable
36	68	Illinois
40	64	HTCP
61	42	LP
62	40	Reno
63	40	Hybla
64	40	CUBIC
95	9	Vegas

TABLE XIII

SCALABLE VS. OTHER TCP VARIANTS.

Mean throughput (Mbps)		
Scalable	Other TCP variants	
50	53	Illinois
51	52	Scalable
51	55	BIC
23	80	Westwood
62	42	HTCP
71	32	Highspeed
74	29	Hybla
76	27	LP
77	26	Reno
72	31	YeAH
77	26	CUBIC
95	8	Vegas

depends upon the order in which the Scalable and Illinois streams are started. Scalable is very aggressive and dominates the amount of available bandwidth when competing against other TCP variants. The only exception to this is Westwood, which maintains a very high average throughput against Scalable.

V. CONCLUSION

Comparing the congestion control algorithms to each other shows that whilst some of the algorithms can co-exist, others cannot. TCP-Vegas consistently had a low mean throughput of about 10 Mbps against all other TCP variants. TCP-Vegas is perhaps the algorithm that is most sensitive to network congestion, as it gives up bandwidth the most easily.

The most aggressive algorithms are HTCP, Westwood, Illinois, BIC and Scalable. Of these four, Westwood is the most greedy, taking nearly all available bandwidth for itself. It is perhaps the most insensitive to congestion on the network.

YeAH, HighSpeed, LP, Hybla, Reno, CUBIC, and Reno share the available bandwidth more or less equally among themselves.

Most of the congestion control algorithms that were investigated in this paper are intended for high speed networks with large RTTs namely HTCP, Illinois, Scalable, BIC, CUBIC, YeAH, and HighSpeed. Since these algorithms all try to exploit large amounts of bandwidth by rapidly increasing the size of their congestion windows, it is to be expected that they would not share bandwidth well with other TCP connections. The exception is CUBIC and YeAH, which form part of the well-behaved group. These high speed congestion control algorithms seem particularly effective at not overwhelming the network, and sharing available bandwidth with other TCP connections.

REFERENCES

[1] Omar AitHellal and Eitan Altman. Analysis of tcp vegas and tcp reno. *Telecommunication Systems*, 15:381–404, 2000. 10.1023/A:1019159332202.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[3] Andrea Baiocchi, Angelo P. Castellani, and Francesco Vacirca. Yeah-tcp: Yet another highspeed tcp. In *5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, March 2007.

[4] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22, 2004.

[5] Douglas E. Comer. *Internetworking with TCP/IP Principles, Protocols and Architectures*. Prentice Hall, New Jersey, USA, fourth edition, 2000.

[6] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno and sack tcp. *SIGCOMM Comput. Commun. Rev.*, 26(3):5–21, July 1996.

[7] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[8] Luigi A. Grieco and Saverio Mascolo. Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control. *SIGCOMM Comput. Commun. Rev.*, 34(2):25–38, April 2004.

[9] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[10] Tom Kelly. Scalable tcp: improving performance in highspeed wide area networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, April 2003.

[11] Aleksandar Kuzmanovic and Edward W. Knightly. Tcp-lp: low-priority service via end-point congestion control. *IEEE/ACM Trans. Netw.*, 14(4):739–752, August 2006.

[12] Douglas Leith and Robert Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *2nd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, February 2004.

[13] Shao Liu, Tamer Başar, and R. Srikant. Tcp-illinois: a loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, New York, NY, USA, 2006. ACM.

[14] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 287–297, New York, NY, USA, 2001. ACM.

[15] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand. Analysis and comparison of tcp reno and vegas. In *In Proceedings of IEEE Infocom*, pages 1556–1563, 1999.

[16] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.

[17] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514 – 2524 vol.4, March 2004.

Arnè Esterhuizen is a BSc (Honours) student at the Department of Mathematical Science (Computer Science Division) at the University of Stellenbosch.
Anthony Krzesinski is a Professor of Computer Science at the University of Stellenbosch, South Africa. His research interests centre on the performance evaluation of telecommunication networks.