Journal of King Saud University – Computer and Information Sciences (2012) **24**, 145–155

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

REVIEW ARTICLE

Exploration and evaluation of traditional TCP congestion control techniques

Ghassan A. Abed *, Mahamod Ismail, Kasmiran Jumari

Department of Electrical, Electronic and Systems Engineering, Faculty of Engineering and Built Environment, National University of Malaysia, 43600 UKM, Bangi, Selangor, Malaysia

Received 27 October 2011; revised 8 March 2012; accepted 13 March 2012
Available online 5 April 2012

KEYWORDS

TCP;
Congestion control;
Slow-start;
Congestion avoidance;
Fast recovery;
Fast retransmit

Abstract TCP or Transmission Control Protocol represents one of the prevailing "languages" of the Internet Protocol Suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as TCP/IP. TCP provides reliability to data transferring in all end-to-end data stream services on the internet. This protocol is utilized by major internet applications such as the e-mail, file transfer, remote administration and world-wide-web. Other applications which do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a datagram service that emphasizes reduced latency over reliability. The task of determining the available bandwidth of TCP packets flow is in fact, very tedious and complicated. The complexity arises due to the effects of congestion control of both the network dynamics and TCP. Congestion control is an approved mechanism used to detect the optimum bandwidth in which the packets are to be sent by TCP sender. The understanding of TCP behaviour and the approaches used to enhance the performance of TCP in fact, still remain a major challenge. In conjunction to this, a considerable amount of researches has been made, in view of developing a good mechanism to raise the efficiency of TCP performance. The article analyses and investigates the congestion control technique applied by TCP, and indicates the main parameters and requirements required to design and develop a new congestion control mechanism.

* Corresponding author.
E-mail addresses: ghass@ieee.org, ghassan@eng.ukm.my (G.A. Abed), Mahamod@eng.ukm.my (M. Ismail), kbj@eng.ukm.my (K. Jumari).

Peer review under responsibility of King Saud University.

1319-1578 © 2012 King Saud University. Production and hosting by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jksuci.2012.03.002

Contents

1. Introduction

The massive and rapid growth in internet propagation and with the widespread use of TCP/IP, the congestion control mechanism becomes the decisive factor in improving the efficiency or performance of TCP. The TCP congestion control algorithm is the key factor which plays a critical role in the level of performance and the demeanor of the amount of data stream within networks. Now that the internet users are exponentially growing at rapid pace, internet congestion is much expected, and is one of the main issues in computer network. Congestion occurs when the number of received packages of a node is more than its output capacity (Shirazi, 2009). TCP Tahoe is the first TCP variant which includes the first congestion control algorithm. This algorithm is developed by Jacobson and Karels in 1986. Based on the same concept presented by Jacobson and Karels, many more algorithms are then introduced. Following that, many enhancements and modifications are conducted on Tahoe, and leads to design and development of new TCP variants with different congestion window algorithms (Mo et al., 1999). The performance of TCP variants are directly affected by its own congestion control mechanisms, where the packet amount transferred over network connections is based on the work and the behaviour of the congestion control and its role in exploiting the capacity of the network path (Sarolahti, 2002). RFC 793 standardized the first TCP version with its basic configuration based on a scheme of window-based flow control. TCP Tahoe represents the second generation of TCP versions, which includes two new techniques, congestion avoidance and fast transmission. Reno is the third version of the first developed series, and it is standardized in RFC 2011, where the congestion control mechanism is further extended by fast recovery algorithm (Yuan-Cheng and Chang-Li, 2001).

The implementation of the early TCP versions utilizes a simplified go-back-n model, but this model does not tell of any assumptions to the congestion control. In this model, the flow rate of data is transmitted from sender to receiver, where the sender is not awaiting ACK (acknowledgement) from the receiver to send another set of new data. The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends. The receiver will ignore any frame that does not match the sequence of number it expects (this means the receiver will ignore the frame that is a "past" duplicate of the frame it has ACK'ed and "future" frame past the last packet it is waiting for). Once the sender has sent all the frames in its window, it will detect the outstanding frames, and will go back to sequence number of the last ACK it received from the receiver process, and will start fill its window starting from that frame and continues the process over again. This means the sender will retransmit all the segments, beginning from the oldest segment lost (Fahmy and Karwa, 2001). The go-back-n model is opposed to the stop-and-wait model. The other approach of sending data by TCP protocol depends on using end-to-end congestion control without adopting the congestion control of network-assisted connection (due to the in IP layer, there is no explicit feedback to the system end relaying status of network congestion). Realistically, when TCP connection transmits data into connection pipe, the data amount is controlled and limited by congestion control of the sender, where the congestion window determines the essential send rate (Kurose and Ross, 2006). The window-based congestion control technique employed by TCP will try to adjust the data flow rate by adjusting the size of the window to avoid network congestion and at the same time, providing a fair share of bandwidth of the network over all possible connections (Kodama et al., 2008; Iguchi et al., 2005).

All TCP variants are considered as the properties and characteristics of wired networks, which are not dependent on the lower network layers. However, in heterogeneous networks, the congestion control of TCP proves to be lacking performance. TCP also suffers from poor performance in high bandwidth links as well, mainly due to the slow response of congestion control over large bandwidth connection as well as poor exploitation of the available bandwidth (Sharma, 2006). Another setback to the regular TCP variants is its inability to fully adjust itself to its limited resources and its lacking ability to recognize congested or random lost packets. In some standard TCP variants, namely, Reno, the congestion control increases exponentially with the packets over the connection, where this initial slow start increasing period must be controlled to avoid declining performance of TCP due to the expected overflow of the receiver buffer. One of the few ways to modify TCP is based on the estimated available bandwidth sufficient to provide fair sharing to all flows and adjusting the window according to the available bandwidth and flows number (Qian and Dongfeng, 2010). The accurate bandwidth estimation is dependent on many complex parameters and factors such as traffic stability throughout the network path length.

The mechanism of congestion control is classified into 4 main PHASES, namely, SLOW START, CONGESTION AVOIDANCE, FAST RETRANSMIT and FAST-RECOVERY. Throughout the past ten years, new approaches and techniques have been researched and suggested to deal with the complexity of the nature of wireless link. The modern high-speed wired-wireless networks have expanded tremendously for the past ten years, and thus this problem is brought into light, and many researchers are still actively developing methods to expand the effectiveness of TCP domain (Jacobson, 1990).

2. Slow-start phase

The initial phase of implementation of TCP software is the slow-start phase, where the slow-start algorithm is used by TCP sender to adjust the data flow rate to the receiver where the period of new slow-start begins with every acknowledgement (ACK) received from the TCP receiver. This means the transmission rate from the TCP sender is fully dependent on the acknowledgements (ACK) returned by the TCP receiver. The process of slow-start is based on a simple concept, and it has not been revised until 1988. Slow-start technique may only represent a fragment of the congestion control, but undoubtedly it can affect the network behaviour and performance as a whole considerably much (Law and Hung, 2001). Commonly, after re-transmission time-out, the slow-start phase is launched at the sending site of TCP connection, and later on, this will significantly affects the performance of this connection as a whole. The straightforward objective of slow start mechanism is to assist the sender in realizing the obtainable bandwidth in the network path. This is done by progressively growing the quantity of segments that can be introduced into the pipeline from the earliest window size of one segment. In this way, bottleneck of connection with huge flow of segments can be avoided (Wang et al., 2000). The creative slow-start process of Jacobson (Jacobson, 1988) begins by using a congestion window of only one segment size, and with every single ACK acknowledged, it grows the size of cwnd by one additional segment. As a consequence of this process, the cwnd will twin its magnitude at each RTT of the TCP period, producing an exponential upsurge of the amount of the segments added into the network of each RTT (Cavendish et al., 2005). This slow-start phase is slow when the network is not congested and the network response time is good. For instance, the major positive transmission and acknowledgement of the segment will expand the window to double segments. Consequently, when the transmission of these two segments has completed and ACK received, the window size is further enlarged to four segments. This process is repetitively repeated to produce eight segments, then sixteen segments, and so on until it reaches the full size of the window promoted by the receiver or until congestion happens, whichever comes first (Kristoff, 2002). As shown in Fig. 1, when the connection initiates, cwnd will be sent by the sender to one segment at the beginning, and this further increases by one segment (exponential growth of cwnd) for every RTT with each new successful ACK received (Yuan-Cheng and Chang-Li, 2001).

With reference to Fig. 1, the slow-start begins with one segment for cwnd (cwnd = 1 segment), and on each successive ACK received, the cwnd increases by one (new cwnd = previous cwnd + 1). The exponential growth of cwnd with each

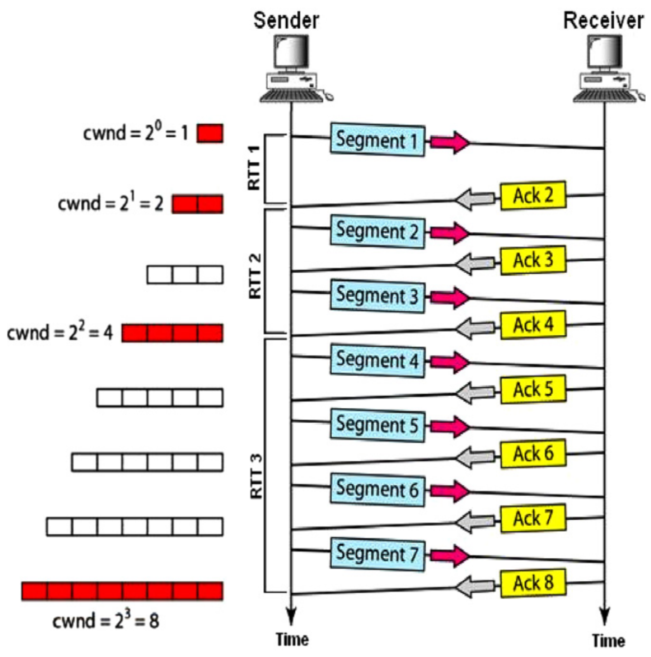

Figure 1 TCP slow-start increases exponentially with RTT.

RTT will duplicate cwnd size (new cwnd = previous cwnd × 2), until cwnd reaches the congestion point of the link. TCP link kicks off in a slow-start state and the TCP regulates the transmission amount grounded on the rate at which the acknowledgements are established from receiver. The TCP slow-start is applied via two variables, congestion window (cwnd) and slow-start threshold (ssthresh). The variable ssthresh is the threshold value at which TCP leaves the period of slow-start. If at one point of time when the cwnd rises outside ssthresh, the TCP period at that point of time is measured to be out of slow-start phase. With more RTT interactions, cwnd will eventually go beyond ssthresh, and the session will be deliberated out of the period of slow-start. At this point of time, the TCP source goes into the next stage (congestion avoidance phase) where it analyses for extra bandwidth by growing the size of congestion window more gradually this time (Cheng et al., 2005). At this point of time, the connection of TCP from the side of client is out of slow-start phase, but nonetheless, the end of server is quite in slow-start phase as it has yet to transmit any segment to its client. The exiting of slow-start phase indicates that the TCP connection is already in a stable state where the congestion window carefully ties the volume of the network path. However, the congestion window will not change geometrically, but will move linearly when the connection goes out of the slow-start phase.

3. Congestion avoidance phase

Back in 1986, Jacobson proposed congestion avoidance mechanism to fix the internet meltdown problem. This mechanism is still currently being used to date in TCP implementations. Once the threshold value is hit (ssthresh), TCP slows the rate of increased window size (slows down from exponential growth in the slow-start state to linear growth in the congestion avoidance state). After a period of time, TCP transmission rate

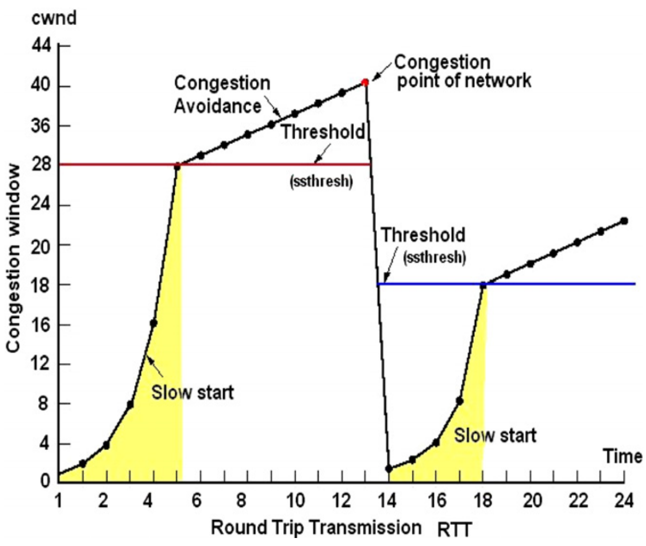

Figure 2 Combined operations of slow-start and congestion avoidance.

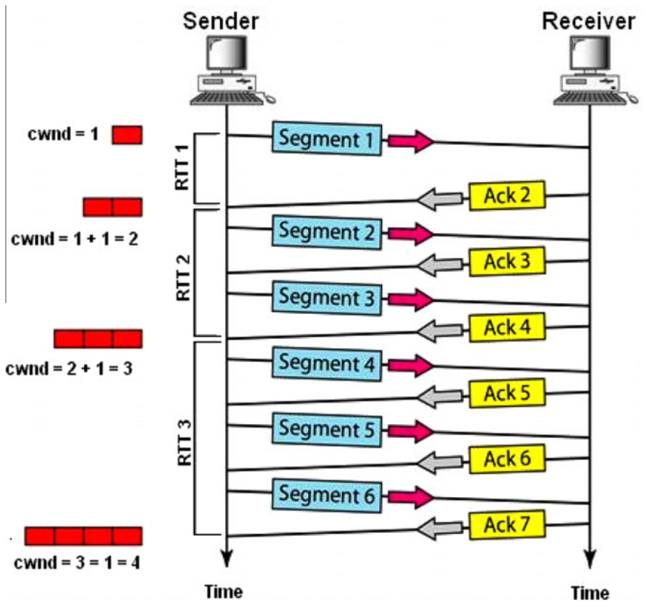

Figure 4 Traditional illustration of congestion avoidance.

exceeds the network link capacity and thus inducing packet loss. TCP will immediately detect the packet loss and reduces the congestion window size to almost half of its actual value (Vallamsunda, 2007). Fig. 2 explains the combined operations and the function relationship between slow-start and congestion avoidance phase.

The congestion avoidance mechanism is implemented when the congestion window size is bigger than the recent slow-start threshold. In congestion avoidance, the congestion window is enlarged further by the dimension of one full-sized segment once in RTT, and then it is reduce to half of its earlier dimension when a TCP sender senses congestion (packet loss) (Sarolahti, 2007). In some TCP variants such as Reno, the slow-start starts at half of its previous size, however, in other TCP variants such as Tahoe, the slow-start begins from cwnd = 1 (as opposed to cwnd = half of its previous size). Ideally, the slow-start phase gives a smooth exponential growth, however, in actuality; the growth is less drastic (TCP, 2006). Also, in this algorithm, the assumption in packet loss is indeed very minor, much less than 1%. The loss of packet hints congestion somewhere in the connection between the source and the destination. Here, two indications that indicate such a scenario is, firstly, when time-out occurs, secondly, when the sender received duplicate ACKs. The congestion avoidance mechanism pushes the behaviour of TCP (forcefully) to steady state, after which it will consequently request for increment of congestion window in a constant volume to complete one round trip. This cycle will again be repeated by again decreasing it in a constant multiplicative element once congestion has occurred yet again (Mathis et al., 1997). The stability of congestion avoidance mechanism is used to save the transmission flow rate from oscillating near the capacity approximation estimated through slow-start period. Throughout the duration of congestion avoidance, the transfer amount is increased predictably in a linear style (Eddy and Swami, 2005).

Actually, congestion avoidance and slow-start are liberated mechanisms with each having a different objective. Congestion can happen anytime throughout, and in practice, the two mechanisms are concurrently executed (Rodriguez and Corporation, 2002). The algorithm of slow-start is the primary algorithm for transferring data for TCP connection, despite it being used as clarified, however, it is to be noted that through slow-start process, the connection network is forced to drop

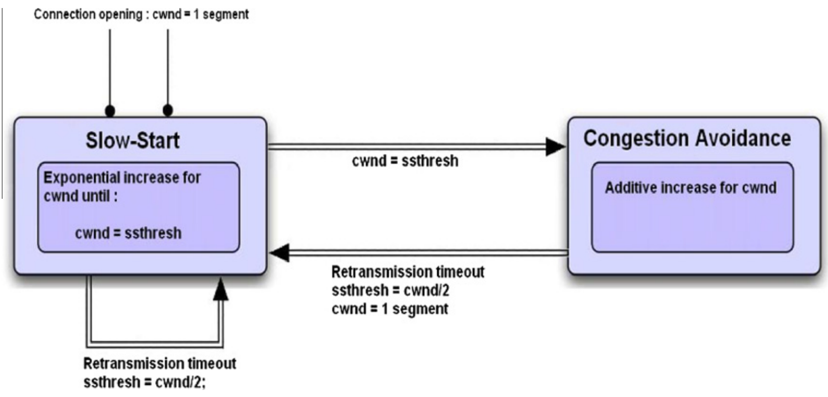

Figure 3 Slow-start and congestion avoidance cycling.

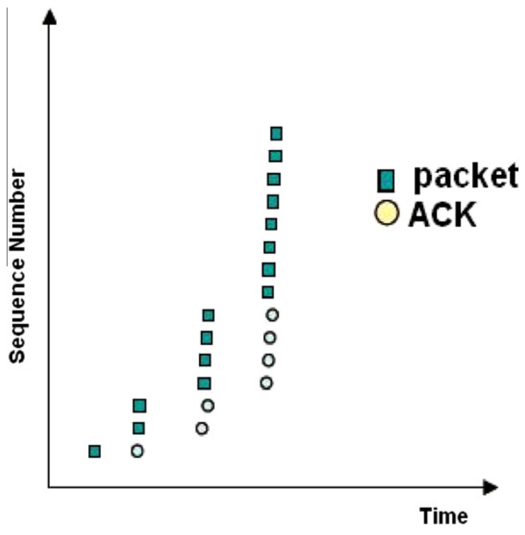

Figure 5 Congestion window doubles every RTT.

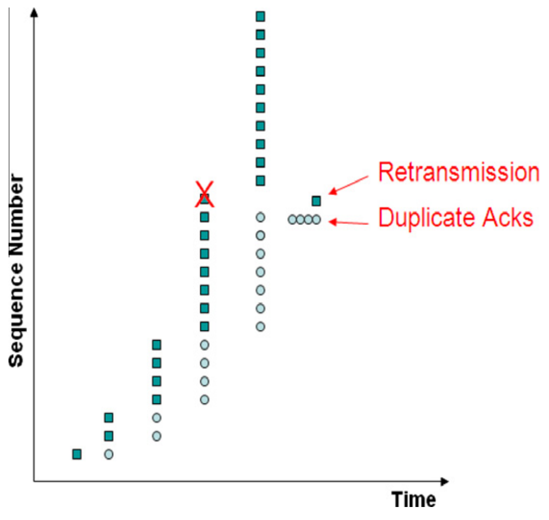

Figure 6 Sequence number with segment loss scenario.

one or more packets due to congestion or surplus. When this occurs, the congestion avoidance mechanism is applied to slow the flow rate. During the process of congestion avoidance, the timer of segment retransmission is beginning to expire or reception of ACK duplication will indirectly hint the sender that a connection congestion state is happening. Then, the sender instantly changes the size of transmission window to half of the recent window size. But when congestion was detected by a time-out, the congestion window will be reorganized to only one segment, and this will then trigger slow-start state automatically as shown in Fig. 3.

With reference to Fig. 4, the cwnd size is doubled for every RTT, resulting in an exponential increment if compared against RTT. This growth remains constant until the cwnd reaches or exceeds the value of ssthresh, or the size of the connection bandwidth.

If cwnd > ssthresh, the TCP initiates the congestion avoidance algorithm, while if cwnd = ssthresh, either slow-start or congestion may be applied (Bansal, 2005). However, it is worth mentioning that the slow-start is only used up to the intermediate point where congestion initially occurred. After this intermediate point, the congestion window is enlarged by one segment for all segments in the transmission window that are acknowledged. This mechanism will push the sender to further slowly raise its flow rate (at much slower rate) as it reaches the line where congestion had earlier been spotted (Qian and Dongfeng, 2010). Fig. 4 is a traditional diagram of the activities of the TCP congestion control mechanism. The figure illustrates that in the opening, the TCP sender doubles the size of congestion for every RTT until the congestion window size is bigger than the ssthresh. From that point on, the congestion window increases by one segment size each RTT as explained in Fig. 5.

The scheme of congestion avoidance permits a network to run in the area of high throughput and low latency, while congestion control represents a recovery process. Similar to other control systems, they consist of two fragments, a control mechanism and a feedback mechanism. The feedback mechanism permits the network to notify the clients (both source and destination) of the recent state of the network, while the control mechanism lets the clients regulate their load on the network. The feedback signal in a congestion avoidance structure signals the clients on the current state of functionality of the network whether it is functioning under or beyond the border (Jain and Ramakrishnan, 1988). In the slow-start phase of the window expansion algorithm, the source node transmits two segments for each ACK received. In the congestion avoidance phase, the source node normally transmits one segment for each ACK received (Floyd and Jacobson, 1991). The basic principles of both slow-start and congestion avoidance phase (algorithm) and their correlation with each other plays a big contribution in this thesis as it supports the development of a new congestion avoidance mechanism aiming to enhance the performance of TCP in large bandwidth and low latency wired wireless links.

4. Fast recovery and fast retransmit phase

Slow-start and congestion avoidance of TCP congestion control coordinate the throughput radically after segment damage is detected. Fast retransmit and fast recovery provides a mechanism to speed-up the retrieval of the connection. This mechanism perceives the losses in segments by duplicate acknowledgements. This mechanism is the recovery method used by TCP to evade the waiting time of retransmission time-out for each loss segment (Olsén, 2003). The mechanism of fast recovery adjusts sending of new segments till a non-duplicate acknowledgement is received by the sender. Let's say after receiving a packet (sequence number 1), the receiver sends an acknowledgement adding value of 1 to the sequence number (sequence number = 1 + 1 = 2), which means that receiver receives packet number 1 and is expecting packet number 2 from the sender.

Assuming three subsequent packets have been lost, the receiver receives packet number 5. After receiving packet number 5, the receiver sends an acknowledgement with the sequence number 2 and 6. When the receiver receives packet number 6, it sends an acknowledgement with the sequence number 2 and 7. In this way, the sender receives more than one acknowledgement with the same sequence number 2, which is called duplicate acknowledgement. If a TCP sender receives a specified number of acknowledgements which is usually set to three duplicates (that is four acknowledgements with the same ACK number), the sender can then be reasonably confident that the segment with the next higher sequence number was dropped, and will not arrive out of order. By this process, the acknowledgement clocking can be conserved, and when a non-duplicate ACK reaches, the fast recovery is finished, and cwnd is emptied (Gurtov, 2000). With reference to Fig. 6, once a loss of segment occurs, the TCP receiver will preserve transmitting ACK segments specifying the next predictable order number. The sequence number will relate to the loss segment. When an individual segment is lost, TCP will preserve creating ACKs for the next segments. This will then induce the reception of duplicate ACKs by the sender. Duplicate ACK represents the lost packet.

In fast retransmit stage, once TCP gets duplicate ACKs, it adopts to resend the segment, where no waiting time is required for the segment timer to expire. This process will speed up the recovery of segment losses. In fast recovery, when the segment is lost, the TCP attempts to keep the existing flow rate without returning to slow-start. The fast retransmit mechanism was initially introduced in TCP Tahoe (Wang et al., 2000), based on the concept of resending the unacknowledged segment after three duplicate ACKs are received. When this happens, cwnd size is reset to one packet and the process of slow-start is initiated. The fast retransmit mechanism for TCP Tahoe is illustrated in Fig. 7. The fast recovery mechanism initially appeared in Reno (Floyd, 2004). Fast recovery is a mechanism that replaces slow-start when fast retransmit is used. While duplicate ACKs indicate that a segment has been lost, nevertheless it also indicates that packets are still flowing since the source received a packet with a sequence number higher than the missing packet. In this scenario, the assumption is that a single packet has been dropped and the network is not fully congested. Therefore, the TCP sender does not need to fully drop back to slow-start mode, but to half the previous rate (Lin and Kung, 1998). The first two stages are applied via the sender of TCP to regulate the quantity of packets inserted into the connection, whereas fast retransmit and fast recovery are used to recuperate from the losses in packets without the need to resend RTOs (de AE et al., 2003). Reno is an improved version of Tahoe where the fast retransmit process is further improvised to comprise fast recovery. Commonly, Reno promotes fast retransmit recovery, and the process is followed by congestion avoidance (instead of slow-start) (Abrahamsson et al., 2002). The mechanism stops the connection to stay unfilled for a quick instant after fast retransmit, thus evading the need to slow-start to fill-up for each single loss in packet. Fig. 8 shows the fast recovery scenario mechanism and the timing relation with slow-start and congestion avoidance phases. Fast retransmit decreases the cwnd to half and at the same time, continues sending segments at this reduced level.

The fast recovery is inserted into the function of TCP sender when receiving a primary threshold of duplicate ACK and the value of this threshold is typically fixed to three (that means four ACK with the same sequence number). When the threshold value has been reached, TCP sender decreases cwnd by half and resends single packet. As an alternative to Tahoe sender, the sender in Reno makes use of the extra incoming duplicate acknowledgement to clock the subsequent departing packets (Fall and Floyd, 1996). If any losses happen,

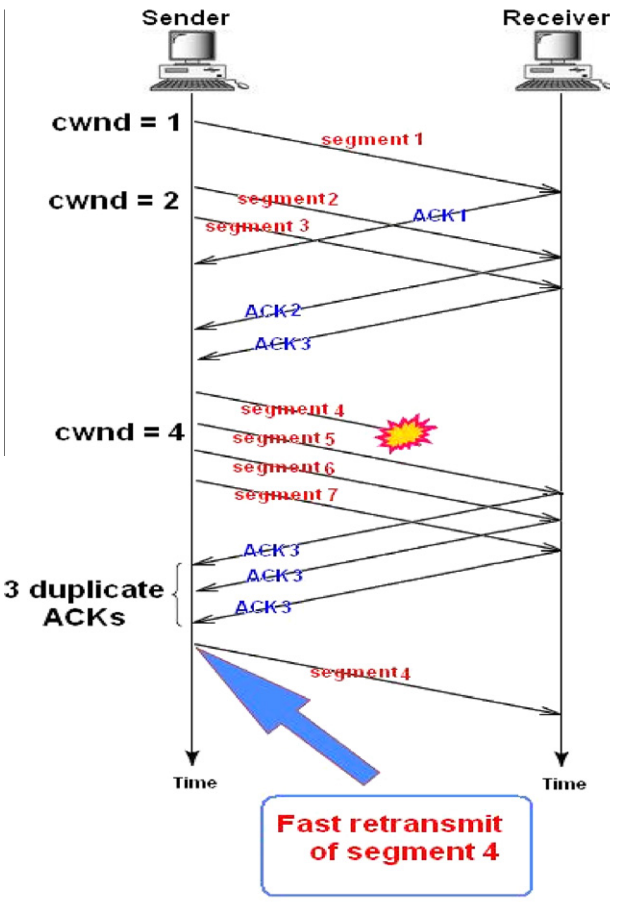

Figure 7 Fast retransmit mechanism of Tahoe.

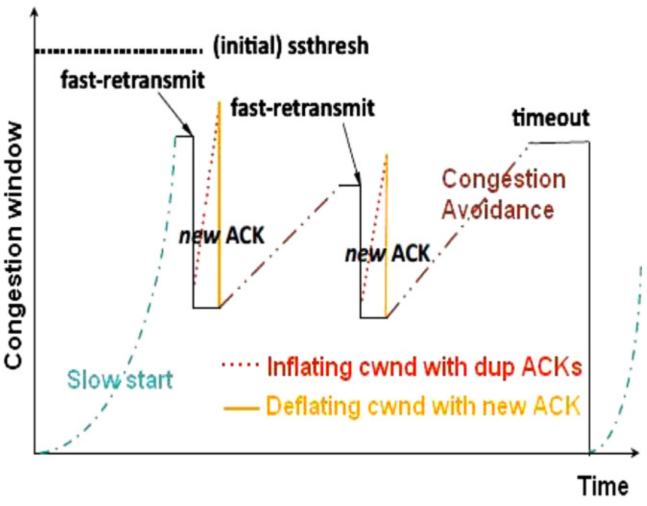

Figure 8 Fast recovery mechanism.

the holes of the sequences will be rearranged and the receiver will be ready to accept new segments which are appropriate with its window, but are not the predictable segment. The receiver will transmit ACKs indicating the sequence hole for each segment over-time, thus duplicated ACKs will be induced. If three or more duplicate ACKs for a segment are

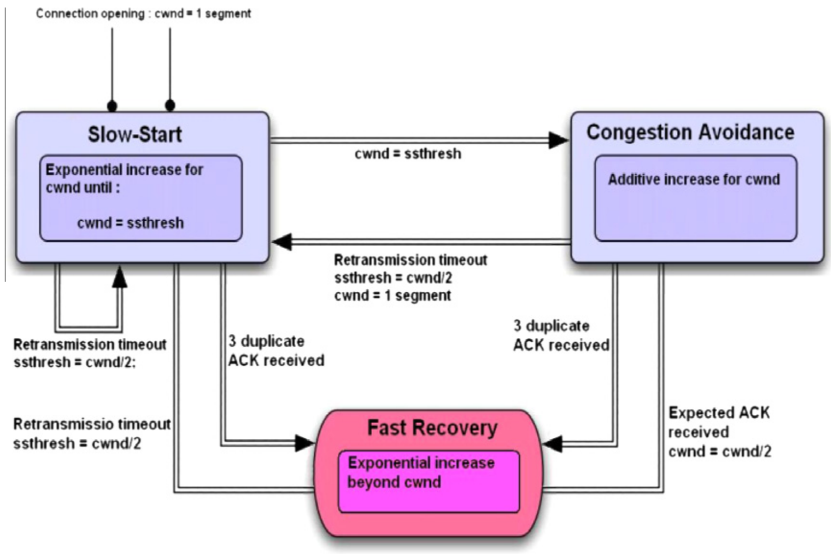

Figure 9 Functional interference among slow-start, congestion avoidance, and fast recovery phases.

transmitted from receiver to sender, the sender adopts immediately that this segment has been missed, and thus, will resend it before RTO expires (Bartók and Cselényi, 2001). Fig. 9 demonstrates the functional interference among slow-start, congestion avoidance, and fast recovery phases with all expected probabilities.

The other implementation of fast recovery algorithm is established in TCP NewReno (Floyd and Henderson, 1999). The fast recovery algorithm is developed in NewReno by means of construing a fractional acknowledgement as a mark of additional lost packet at this sequence number, where fractional ACK is acknowledged after the earlier point, and still in the window of recovery process. This will further enhance the connection throughput when many packets are lost for data of a single window. Comparatively, TCP Reno is waiting RTO for each packet under this situation, followed by a slow-start, while TCP NewReno deals with this problem by its enhanced recovery (Bartók and Cselényi, 2001). Reno's performance suffers in this situation when many losses in packets occur in window (Subedi et al., 2008). If segments reaching the destination are out of sequence, the host cannot distribute these segments to the end request as it needs to buffer these segments till the accurate sequence is obtained. The receiver must transmit an instant duplicate ACK to update the sender of the loss packet (based on the segments that are out of sequence) and the expected number of the sequence. By decreasing cwnd to one single segment after receiving three duplicate ACKs, the network is still able to send segments irrespective of congestion situation (Hassan et al., 2005). The object behind the differences of TCP is that each type has some distinct such as the base TCP has become known as TCP Thaoe. TCP Reno adds one new mechanism called fast recovery to TCP Tahoe. Newreno uses the newest retransmission mechanism of TCP Reno. The use of Sacks permits the receiver to specify several additional data packets that have been received out of order within one dupack. TCP Vegas proposes its own unique retransmission and congestion control strategies. TCP Fack is Reno TCP with forward acknowledgment (Abed et al., 2010).

5. TCP over heterogeneous networks

Given the expansion of wireless technologies and the recent propagation of movable user equipment, there is a perceived and accumulative attractiveness of wireless environments, such as Wireless Local Area Networks (WLANs), Wireless Wide Area Networks (WWANs), and mobile ad hoc networks. In WLANs, such as Wi-Fi, or WWANs, such as 2.5G, 3G, or 4G cellular systems, mobile hosts connect via an access point or base station that is connected to the wired part of the network. Unfortunately, wireless and wired networks are dramatically dissimilar in their parameters of link reliability, bandwidth, and delay time of propagation (Sharma, 2006). In mobile and wireless networks, there are several challenges that are associated with wired networks, because wireless networks have several essential adverse features that will considerably weaken TCP performance, if no actions are taken. These features involve mobility, link asymmetry, and, channel errors which are explained in some detail below.

5.1. Connection asymmetry

The wireless link between a mobile terminal and base station is naturally an asymmetric link, because mobile terminals have restricted power, capability of processing, and limited buffer space.

5.2. Channel errors

In wireless links, comparatively high bit error rates are expected, due to the fading of multipath and shadowing that may damage packets during the data transfer process, leading to a wide loss in TCP components, such as segments or acknowledgments.

5.3. Mobility and handover

Cellular systems are characterized by handovers, due to the user's mobility. Handovers usually cause short-term connection interruptions, resulting in missing packets and adding extra delays over network pipelines.

Theoretically, TCP is independent of underlying layers, and if the underlying layers reduce reliability, then for the wired region (which TCP was first intended to work on), TCP will exhibit some of its deficiencies. However, wireless networks are known for their high probability and large numbers of random errors and irregular connectivity of links. In addition, the congestion control mechanism interprets packet loss as a sign of congestion. In reaction to these missed packets, some TCP versions decrease the size of the congestion window and the flow rate. As a result, a dramatic collapse in TCP performance and throughput can occur. Obviously, users of mobile equipment can dramatically affect the throughput of the TCP, because of mobility features and handover possibilities, which may frequently cause connection interruptions (Mahmoodi, 2009). Commonly, when an IP packet is sent on a wireless link, the layer of IP sees the available capacity, delay characteristics, and amount of loss rate that varies over time. The use of accessible link layer control techniques, such as retransmission scheduling, power controlling, adaption of flow rate, and forward error corrections, permits balance between loss, latency, and capacity. However, it is not easy to reach the constant level of low latency, low loss, and high capacity, which represents the main characteristics of wired links. For most TCP variants, the congestion control mechanism is believed to adapt to the style of collective turbulences within the wired networks, where accessible bandwidth changes according to cross traffic, irregular routing, changing capacity, and delays, which are initiated by constant values of queuing and propagation delays (Möller, 2005).

If TCP is used within a cellular infrastructure, performance in both end-to-end throughput or in the employment of radio links is frequently very poor. This is because the self-motivated characteristics of wireless links and TCPs do not fit well together. Working on the problems of TCPs over wireless links, represents the main objective to achieving a good, or at least acceptable, level of end-to-end throughput and an effective use of radio link resources, in parallel with small fluctuations to current infrastructure and protocols. TCP strategy over wireless networks, takes into account the natural individualities of the wireless network and its requirements. For example, satellite networks involve large propagation delay, and ad-hoc networks are without an infrastructure. The trouble is that all wireless networks should face a large BER. In wired-wireless networks TCP design, one of the main objectives is to identify the reason for packet loss. Some developments target the discovery of explicit techniques to notify the TCP source side of the reason beyond dropping of the packet, whether it is congestion or random error events (Ye et al., 2005).

Table 1 Network links parameters.

Link	Bandwidth	Delay (ms)
N0–R1	500 Kbps	2
N1–R1	10 Mbps	2
N2–R1	500 Kbps	2
R1–R2	4 Mbps	6
R2–R3	4 Mbps	4
R2–N3	8 Mbps	2
N3–N4	8 Mbps	2
R3–N4	8 Mbps	2
R3–R4	4 Mbps	2
R4–N6	10 Mbps	2
R4–N5	10 Mbps	2
R4–N7	500 Kbps	2

6. Performance evaluation

To monitor the performance of standard TCP versions over wired-wireless networks with high congestion connection, we assumed a network topology with 2 wireless nodes and 3 cascaded bottleneck. R1, R2, R3, and R4 represent control node to adjust the packet flow through the network where these construct a three bottleneck. Also, the link R2–N3–N4–R3 creates a re-routing scenario to the packet flow randomly to add more reality to the proposed topology. So, these networks include wired-wireless link, wired nodes, wireless node, three cascaded bottlenecks, and fast re-routing event. All these conditions aim to make the proposed topology scenario similar to the real heterogeneous network with high congestion events. The proposed topology is shown in Fig. 10 and the network parameters illustrated in Table 1. The evaluation of TCP performed with 6 TCP versions, Tahoe, Reno, Newreno, Sack, Fack, and Vegas where these variants include different congestion control mechanisms with different methodologies, except Reno and Newreno which include similar congestion control approach.

The behaviour of the congestion control for these variants shows the poor performance and high packet loss due to these variants not developed to run over wireless medium and standard congestion control algorithm take in to account the

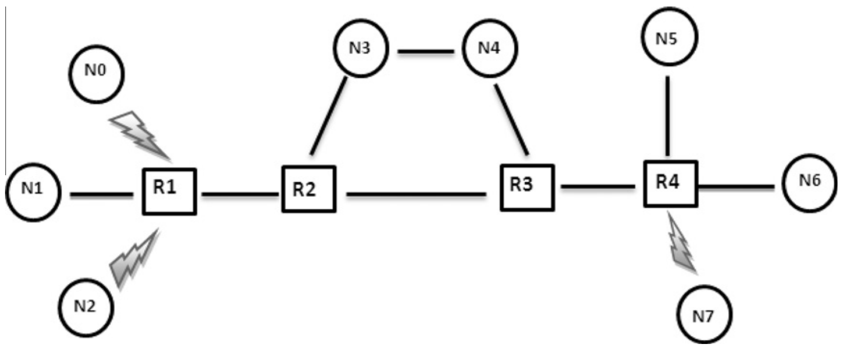

Figure 10 Network topology.

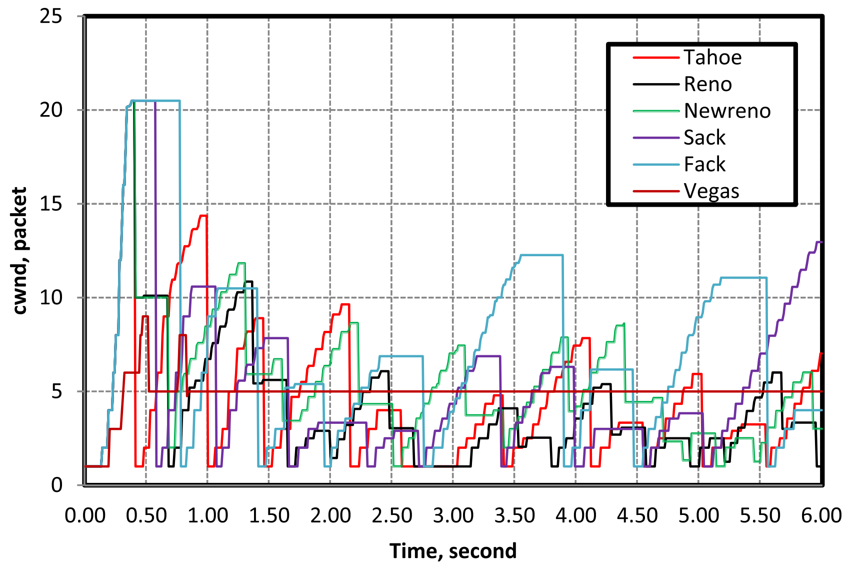

Figure 11 Behavior of congestion window of different TCP variants.

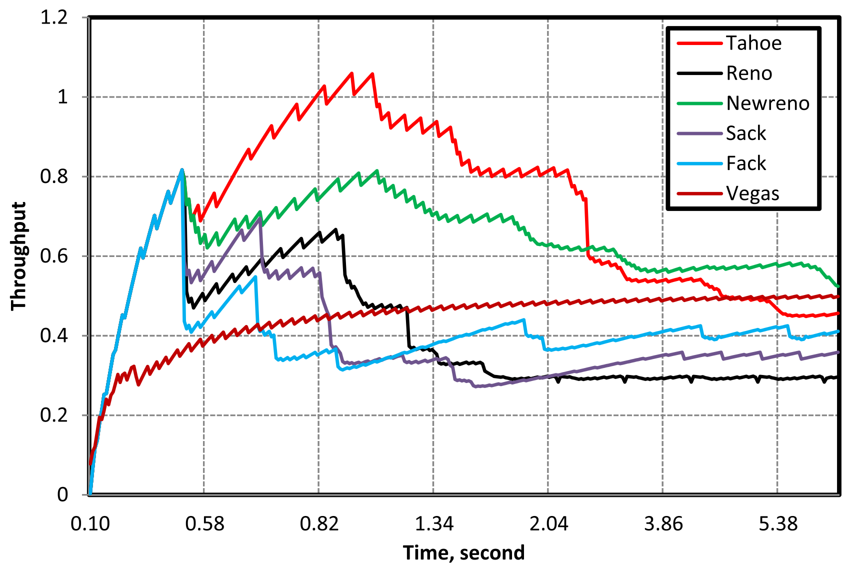

Figure 12 Comparative throughputs of different TCP variants.

packet loss in congestion occurrence and not interest with the loss that which happen in air-link connections.

In Fig. 11, the behaviour of cwnd for all TCP variants is showed and only Reno and Newreno can give stable window with less packet loss, also these two variants can provide recovering the lost packet. In fact, the classic algorithm used in these variants not permit the TCP to give high throughput as shown in Fig. 12, even the special approach used in Vegas, but it still cannot give high throughput in spite of the stability.

Other hand, all these variants based on use exponential slow-start increasing in initial window growing, where this scheme is very slow and need long period of time to reach the ssthresh or the network bandwidth. The throughput comparison shows that when TCP Tahoe start give high output put later the performance begins this is because the congestion control algorithm represents the first generation and it suffer when the network enter congestion mode or when a lot of handovers and packet loss events happen when the network includes wired-wireless connections as the proposed topology. Generally, Newreno can give better output form the other variants due to the intelligent congestion control algorithm which used with it as explained in previous sections.

The behaviour of congestion window for these 6 TCP variants not appear clearly here in Fig. 11 where all these variants (not including Vegas) share same slow-start algorithm to increase the congestion window after each initialization or after recovering phase. The standard slow-start algorithm based on increment one segment only for each ACK so the exponential increase degrades the throughput of these TCP over congestion links. For these reason, the throughput of Vegas is more stable due to it not use exponential increasing in slow-start phase while the other variants can start with high throughput bur after packet begins drop, the throughput go to down level. TCP Tahoe can give good output in beginning of simulation but it will not be able to recover all the loss happens due to the network pipe congestion. On the other side, the throughput of Newreno TCP gives better performance than other TCP versions which is because the intelligent congestion avoidance used in the congestion control mechanism of Newreno. In spite of the reasonable performance of Newreno it is unable to achieve the requirements of next generation networks where the loss happens not because the congestion only but because the handover in linked over wireless nodes.

In recent days and with the growth in the network facilities, the current congestion control algorithms used with standard TCP's arenot able to run over networks that include low propagation delay and large bandwidth exceeds several hundreds of Mbps such as LTE (Long Term Evolution) and LTE–Advanced. For these reasons the future congestion control mechanism should base on use high speed algorithm with multiple flows to provide parallel TCP's that can give multiple throughput over same connections such as MulTCP (Crowcroft and Oechslin, 1998) and TCP-FIT (Wang et al., 2011).

7. Conclusion and discussion

Most data applications are built on top of TCP since TCP provides end-to-end reliability via retransmissions of missing IP packets. TCP is originally designed for wired networks where the packet losses are due to network congestion and hence the window size of TCP is adjusted upon detection of packet losses. There are a few different kinds of congestion control algorithms, which are implemented in different TCP versions. The ordinary TCP assumes that 99% of the losses in packets in the network are caused by congestion and the remaining is caused by damage. Many enhancements of the ordinary TCP already been suggested to solve the hand-off and the BER problems such as Freeze TCP, I-TCP, M-TCP, and Snoop TCP.

A critical design issue of TCP is its congestion control that allows the protocol to adjust the end-to-end communication rate to the bandwidth on the bottleneck link. However, TCP congestion control may function poorly in high bandwidth networks due to its slow response with large congestion windows. In other words, cwnd represents number of packets that is allowed to be transmitted without getting acknowledged. Theoretically, TCP should be independent of the underlying medium, and when the underlying medium detracts from the reliable, wired media that TCP was originally designed to serve, it exhibits a number of shortcomings. Wireless networks are characterized by random and high probability of errors and also intermittent connectivity. On the other hand, congestion control algorithms interpret packet loss as the indication of congestion. Most of the TCP variants based on congestion control decreases the cwnd and the transmission rate accordingly, and thus a dramatic degradation in TCP throughput can occur. Recently, many congestion control protocols have been proposed, especially for streaming multimedia applications.

The effective bit error rates in wireless networks are significantly higher than those of the wired networks. Since TCP does not possess any mechanism to differentiate between congestion losses and wireless random losses, the latter may cause severe throughput degradation. Therefore, the enhancement of TCP congestion control is essentially required to cover the new requirements over new applications.

References

Abed, G.A., Ismail, M., Jumari, K., 2010. Behavior of cwnd for TCP source variants over parameters of LTE networks. Information Technology Journal 10.

Abrahamsson, H., Hagsand, O., Marsh, I., 2002. TCP over high speed variable capacity links: a simulation study for bandwidth allocation, 117–129.

Bansal, D., Third-Party TCP Rate Control, 2005.

Bartók and Cselényi, I., Implementation and Evaluation of the BLUE Active Queue Management Algorithm, ed: Citeseer, 2001.

Cavendish, D., Kumazoe, K., Tsuru, M., Oie, Y., Gerla, M., 2005. CapStart: an adaptive tcp slow start for high speed networks, 15–20.

Cheng, R.S., Lin, H.T., Hwang, W.S., Shieh, C.K., 2005. Improving the ramping up behavior of TCP slow start 1, 807–812.

Crowcroft, J., Oechslin, P., 1998. Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP. ACM SIGCOMM Computer Communication Review 28 (3), 53–69.

de AE, L., da Fonseca, N.L.S., de Rezende, J.F., 2003. On the performance of TCP loss recovery mechanisms. In: IEEE International Conference on Communications, vol. 3, pp. 1812–1816.

Eddy, W., Swami, Y., 2005. Adapting End Host Congestion Control for Mobility. Citeseer.

Fahmy, S., Karwa, T.P., TCP congestion control: overview and survey of ongoing research, CSD-TR-01-016, 2001.

Fall, K., Floyd, S., 1996. Simulation-based comparisons of Tahoe, Reno and SACK TCP. ACM SIGCOMM Computer Communication Review 26, 5–21.

Floyd, S., 2004. Limited slow-start for TCP with large congestion windows.

Floyd, S., Henderson, T., 1999. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC2582.

Floyd, S., Jacobson, V., 1991. Traffic phase effects in packet-switched gateways. ACM SIGCOMM Computer Communication Review 21, 26–42.

Gurtov, A., 2000. TCP performance in the presence of congestion and corruption losses, Master's Theses, Department of Computer Science.

Hassan, A., 2005. Predicting TCP congestion through active and passive measurements.

Iguchi, T., Hasegawa, G., Murata, M., A new congestion control mechanism of TCP with inline network measurement, in Proceedings of ICOIN 2005, Jan. 2005.

Jacobson, V., Congestion Avoidance and control, SIGCOMM'88, ACM, August. 1988.

Jacobson, V., TCP Congestion Avoidance Algorithm, End2end-interest mailing list, April. 1990.

Jain, R., Ramakrishnan, K., 1988. Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology, 134–143.

Kodama, M., Hasegawa, G., Murata, M. Implementation experiments of TCP Symbiosis: bio-inspired mechanisms for Internet congestion control, in Proceedings of CQR 2008, Apr. 2008.

Kristoff, J., 2002. TCP Congestion Control. Tech Notes.

Kurose, J.F., Ross, K.W. Computer Networking-Complete Package, third ed., 2006.

Lin, D., Kung, H., 1998. TCP fast recovery strategies: analysis and improvements 1, 263–271.

Mahmoodi, T. Transport Layer Performance Enhancements over Wireless Networks, Licentiate Thesis, August 2009.

Mathis, M., Semke, J., Mahdavi, J., 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. ACM SIGCOMM Computer Communication Review 27, 67–82.

Mo, J., La, R.J., Anantharam, V., Walrand, J., 1999. Analysis and comparison of TCP Reno and Vegas, in INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 1999, pp. 1556–1563 vol.3.

Möller, N. Automatic control in TCP over wireless, Licentiate Thesis, September 2005.

Olsén, J. Stochastic modeling and simulation of the TCP protocol, ed: Uppsala University, Uppsala, Sweden, 2003.

Law, K.L.E., Hung, W.C., 2001. Problems and Solutions for the TCP Slow-Start Process, Citeseer.

Qian, W., Dongfeng, Y. An improved TCP congestion control mechanism with adaptive congestion window, in Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2010 International Symposium on, 2010, pp. 231–235.

Rodriguez, A., Corporation, I.B.M., TCP/IP tutorial and technical overview: Prentice Hall PTR, 2002.

Sarolahti, P., Congestion Control in Linux TCP, in USENIX 2002 Annual Technical Conference, 2002, pp. 49–62.

Sarolahti, P., 2007. TCP Performance in Heterogeneous Wireless Networks.

Sharma, P., Performance Analysis of High-Speed Transport Control Protocols, Master of Science (ComputerScience) Thesis, Clemson University, August 2006.

Shirazi, H.M., 2009. Smart Congestion Control in TCP/IP Networks. Journal of Information and Communication Technology 2 (2), 73–78.

Subedi, L., Najiminaini, M., Trajkovi, L., 2008. Performance Evaluation of TCP Tahoe, Reno, Reno with SACK, and NewReno Using OPNET Modeler. Citeseer.

TCP Revisited, Hughes Systique Corporation2006.

Vallamsundar, B., 2007. Congestion control for adaptive satellite communication systems with intelligent systems.

Wang, H., et al., A simple refinement of slow-start of TCP congestion control, 2000, pp. 98-105.

Wang, H., Xin, H., Reeves, D.S., Shin, K.G., 2000. A simple refinement of slow-start of TCP congestion control, 98–105.

Wang, J., Wen, J., Zhang, J., Han, Y., 2011. TCP-FIT: An Improved TCP Congestion Control. IEEE INFOCOM 2011 2894–2902.

Ye, T., Xu, K., Ansari, N., 2005. TCP in wireless environments: problems and solutions. Communications Magazine, IEEE 43, S27–S32.

Yuan-Cheng, L. Chang-Li, Y., 2001. TCP congestion control algorithms and a performance comparison, in Computer Communications and Networks, Proceedings. Tenth International Conference on 2001, pp. 523–526.