Journal of Mobile Communication 2 (3): 93-98, 2008
ISSN: 1990-794X

Congestion Control Algorithm in Computer Communication Network

S. Arumugam, T. Jebarajan and D.C. Joy Winnie Wise
Department of Computer Science and Engineering, Kumaracoil,
Thuckalay, Kanyakumari Dt., Tamilnadu, India

Abstract: In this study, we propose a new TCP based multimedia congestion control protocol called MCCP. MCCP implements a novel window based congestion algorithm on end-to-end available bandwidth estimation. MCCP effectively estimates the bottleneck bandwidth share of a connection. The estimate is based on information in the acknowledgements (ACKs) and the rate at which the ACKs are received. After a packet loss indication, which could be due to either congestion or link errors, the sender uses the estimated bandwidth to properly set the congestion window and the slow start threshold, thus maintaining the reasonable window size in case of random losses. The goal of MCCP is to estimate the connection available bandwidth to achieve high utilization of bandwidth, without starving other connections. MCCP deals well with highly dynamic bandwidth, large propagation time/bandwidth and random losses in the current and future heterogeneous Internet. MCCP uses fast probing, a mechanism that repeatedly resets slow start threshold based on available bandwidth share estimate. In congestion avoidance, MCCP invokes fast probing upon detection of extra available bandwidth via a scheme we call non-congestion detection (NCD). Fast probing is actually invoked under the following conditions: A large amount of bandwidth that suddenly becomes available due to change in network conditions. Random loss during slow-start that causes the connection to prematurely exit the slow-start phase. MCCP sender gets an accurate estimate of the connection's fair share of the bottleneck bandwidth and effectively adjusts the sending rate to the changing estimate. As a result fast probing enhances probing during slow start and whenever non-congested conditions are detected. Experimental results in ns-2 simulation show that MCCP can significantly improve link utilization over a wide range of bandwidth, propagation delay and dynamic network loading.

Key words: Bandwidth estimation, congestion control, fair share, high-speed networks, random errors, high bandwidth

INTRODUCTION

Transmission control protocol (TCP) has been widely used in the Internet for numerous applications. The success of the congestion control mechanisms introduced in Allman *et al.* (2002) and their succeeding enhancements has been remarkable. The current implementation of TCP Reno/NewReno runs in 2 phases: slow-start and congestion avoidance. In slow-start, upon receiving an acknowledgment, the sender increases the congestion window (cwnd) exponentially, doubling cwnd every round-trip time (RTT), until it reaches the slow-start threshold (ssthresh). Then, the connection switches to congestion avoidance, where cwnd grows more conservatively, by one packet every RTT (linearly). Then, upon a packet loss, the sender reduces cwnd to half.

After a packet loss, instead of simply cutting cwnd by half as in standard TCP, MCCP resets cwnd along with ssthresh according to the MCCP sender's available bandwidth estimate, thus maintaining a reasonable window size in case of random losses and preventing overreaction when transmission speed is high (Floyd, 2003). MCCP relies on an adaptive estimation technique to determine a sender available bandwidth estimate at all times. The goal of MCCP is to estimate the connection eligible sending rate to achieve high utilization, without starving other connections. A brief overview of MCCP and available bandwidth estimation is given in the study.

In this study, we present MCCP with fast probing, a sender-side only enhancement, that intelligently deals well with highly dynamic bandwidth, large propagation times and bandwidth and random loss in the current and future heterogeneous Internet.

The first mechanism is fast probing, which is invoked at connection startup and after extra available bandwidth is detected. Fast probing adaptively and repeatedly resets

Corresponding Author: T. Jebarajan, Department of Computer Science and Engineering, Kumaracoil, Thuckalay, Kanyakumari Dt., Tamilnadu, India

ssthresh based on available bandwidth estimation. The result is fast convergence of cwnd to a more appropriate ssthresh value. In slow-start, fast probing increases utilization of bandwidth by reaching cruising speed faster than existing protocols, this is especially important to short-lived connections. In this study, we extend the use of fast probing to congestion avoidance when extra unused bandwidth is detected.

The second mechanism concerns how to detect extra unused bandwidth. We realized that if a TCP sender identifies the newly materialized extra bandwidth and invokes fast probing properly, the connection can converge to the desired window faster than usual linear increase. This also, applies in the case when a random error occurs during startup, causing a connection to exit slow-start prematurely and switch to congestion avoidance. In this study, we propose a no-congestion detection (NCD) mechanism, which identifies the availability of persistent extra bandwidth in congestion avoidance and invokes fast probing accordingly.

TCP STARTUP PERFORMANCE

In this study, we state briefly the current TCP slow-start mechanisms and evaluate their startup performance in large bandwidth delay networks by simulation. We illustrate the inadequacy of the current schemes when facing networks with large BDP and reveal the reason behind it. This study serves as motivation of our work.

Tcp reno/newreno: In TCP Reno/NewReno, a sender starts in slow-start, cwnd < ssthresh and every ACK received results in an increase of cwnd by one packet. Thus, the sender exponentially increases cwnd. When cwnd hits ssthresh, the sender switches to congestion avoidance phase, increasing cwnd linearly. In this section, we evaluate Reno/NewReno startup performance in large BDP networks. If the initial ssthresh is too low, a connection exits slow-start and switches to congestion-avoidance prematurely, resulting in poor utilization (Hoe, 1996). Figure 1 shows the Reno cwnd dynamics in the startup stage. The results are obtained for a bottleneck bandwidth of 40 Mb/s and RTT values of 40, 100 and 200 ms. The bottleneck buffer size is set equal to BDP in each case.

From Fig. 1, we see that when RTT = 100 ms, Reno stops exponentially growing cwnd long before it reaches the ideal value (BDP = 500). After that, cwnd increases slowly and has not reached 500 by 20 sec. As a result, the achieved throughput is only 12.90 Mb s^{-1}, much lower than the desired 40 Mb s^{-1}. Another observation concerns how RTT affects performance. When RTT

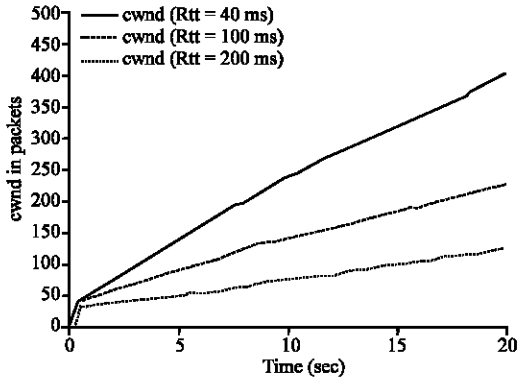

Fig. 1: cwnd dynamic during the startup phase

Table 1: Newreno utilization during first 20 sec (Bandwidth = 40 Mb s^{-1})

RTT (ms)	20	50	100	150	200
Utilization (%)	95.60	71.80	23.20	11.0	7.20

Table 2: Newreno utilization during first 20 sec (RTT = 100 ms)

Bandwidth (Mbps)	10	20	40	80	150
Utilization (%)	77.10	45.90	23.20	13.00	6.90

increases, the ideal window grows too. On the other hand, because cwnd increases one packet per RTT during congestion avoidance, longer RTT means slower cwnd growth, resulting in even lower utilization. The results in Table 1 show the drastic reduction in utilization as RTT increases (Katabi *et al*., 2002).

Consider now the impact of bottleneck bandwidth on utilization during startup stage (Jacobson, 1988). With the increase of bottleneck bandwidth, the packet transmission speeds up, but the sender still has to wait for ACKs to increase cwnd. Thus, after prematurely exiting slow-start, cwnd grows with almost the same rate (one packet per RTT) regardless of bandwidth. With larger bottleneck capacity, more bandwidth is left unused, which leads to lower utilization. Table 2 shows the relation between utilization and bottleneck bandwidth during the startup stage. The utilization drops to 4.7% with 200 Mb s^{-1} bottleneck bandwidth.

MCCP: CONGESTION CONTROL PROTOCOL

In MCCP, a sender continuously monitors ACKs from the receiver and computes its current Available Bandwidth Estimate (ABE). ABE relies on an adaptive estimation technique applied to the ACK stream. The goal of ABE is to estimate the connection eligible sending rate with the goal of achieving high utilization. When a connection initially begins or restarts after a coarse timeout, MCCP adaptively and repeatedly resets the Slow-start threshold (ssthresh) based on the available bandwidth. After a packet loss indication, which could be

due to either congestion or link errors, the sender uses the estimated available bandwidth to properly set the congestion window and the slow-start threshold.

Available Bandwidth Estimation (ABE): The Available Bandwidth Estimate is determined using a time-varying coefficient, exponentially weighted moving average (EWMA) filter, which has both adaptive gain and adaptive sampling. Let t_k be the time instant at which the k_{th} ACK is received at the sender. Let $\hat{S}_k$ be the ABE sample and $\hat{S}_k$ be the filtered estimate of the ABE at time t_k Let α_k be the time-varying coefficient at t_k. The ABE filter is then given by:

$$\hat{S}_k = \alpha_k \hat{S}_{k-1} + (1-\alpha_k) S_k \quad (1)$$

Where,

$$\alpha_k = \frac{2\tau_k - \Delta t_k}{2\tau_k + \Delta t_k}$$

and τ_k is a filter parameter which determines the filter gain and varies over time adapting to RTT and other path conditions. In the filter formula, the ABE sample at time K is:

$$s_k = \frac{\sum_{t_j > t_k - T_k} d_j}{T_k}$$

where, d_j is the number of bytes that have been reported delivered by the jth ACK and T_k is an interval over which the ABE sample is calculated. ABE uses a continuously adaptive sampling interval T. The more severe the congestion, the longer T should be. ABE provides an adaptive sampling scheme, in which the time interval T_k associated with the Kth received ACK is appropriately chosen between 2 extremes depending on the network congestion level. The sampling interval ranges between T_{min} and T_{max}. T_{min} is the ACK inter arrival time, while T_{max} is set to RTT.

To determine the network congestion level, the ABE estimator compares the Estimated Available Bandwidth with the instantaneous sending rate obtained from cwin/RTT_{min}. The difference between the instantaneous sending rate and the achievable rate, clearly feeds the bottleneck queue, thus revealing that the path is becoming congested. The larger the difference, the more severe the congestion and the larger the new value of T_k should be.

When the K_{th} ACK arrives, the estimator first checks the relation between ABE estimates $\hat{S}_{k-1}$ and the current cwin value. When $\hat{S}_{k-1}$ * $RTT_{min} \geq$ cwin, indicating a path without congestion, T_k is set to T_{min}. Otherwise, T_k is set to:

$$T_k = RTT * \frac{cwin - (\hat{S}_{k-1} * RTT_{min})}{cwin} \quad (2)$$

Or upon rearrangement:

$$T_k = RTT * (\frac{cwin}{RTT_{min}} - \hat{S}_{k-1}) / \frac{cwin}{RTT_{min}} \quad (3)$$

In Eq. (3), cwin/RTT_{min} is the expected sending rate, while S_{k-1} is the estimated rate the network allowed. After T_k is chosen, the ABE sample associated with K_{th} received ACK is then expressed by:

$$s_k = \frac{\sum_{t_j > t_k - T_k} d_j}{T_k} \quad (4)$$

we evaluate the accuracy of ABE estimates, both in the presence of network congestion and in the presence of link errors (Jain and Dovrolis, 2003).

Adapting to RTT and network instability in ABE: The EWMA (Exponentially Weighted Moving Average) filter

$$\hat{S}_k = \alpha_k \hat{S}_{k-1} + (1-\alpha_k) S_k$$

places more importance on recent data and discounts older data in an exponential manner. The value of the parameter α_k, dictates the degree of filtering. The smaller α_k the more agile the filter and the larger α_k the more stable the filter. Basically, when τ_k is larger, α_k will be larger and the filter tends to be more stable and less agile.

In ABE, we propose that the parameter τ_k adapts to network conditions to dampen estimates when the network exhibits very unstable behavior and react quickly to persistent changes. A stability detection filter can be used to dynamically change the value of τ_k. We measure the network instability U with a time-constant EWMA filter.

$$U_k = \beta U_{k-1} + (1-\beta) |S_k - S_{k-1}| \quad (5)$$

In Eq. (5), S_k is the K_{th} rate sample and β is the gain of this filter, which is set to be 0.6 in our experiments. When the network exhibits high instability, the consecutive observations diverge from each other, as a result, U_k increases. Under this condition, increasing the value of τ_k makes the ABE filter, as in Eq. (1), more stable. When a

TCP connection is operating normally, the interval between the consecutive acknowledgements are likely to vary between the smallest the bottleneck capacity allows and one RTT. Therefore, τ_k should be larger than one RTT, thus τ_{min} = RTT. We set τ_k to be:

$$\tau_k = RTT + N*RTT\frac{U_k}{U_{max}} \quad (6)$$

The value of RTT in the expression above is obtained from the smoothed RTT estimated in TCP. The factor N is set to be 10 in our experiments, which gives good performance under various scenarios. U_{max} is the largest instability in the ten most recent observations. The sampling time interval T_k is fixed to the interval of the last 2 ACKs in this set of experiments.

Performance evaluation of ABE: TCP behavior in Fig. 2 reveals the reason behind the performance degradation of NewReno and the robustness of MCCP with small buffer capacity. A buffer overflow occurs when the cwin exceeds 46 packets, which is the sum of the pipe and the buffer size. Upon the detection of a packet loss, the NewReno sender (Fig. 2a) sets the new cwin and ssthresh to 23 packets (half of the old cwin). This value is much lower than the pipe size needed to fully utilize the link. Thus, setting cwin and ssthresh to half causes the router to be idle and the link under utilized. On the other hand, Fig. 2b shows that MCCP ABE reduces its cwin and ssthresh according to the available bandwidth estimate (Fig. 2c), which is very close to the optimal value, i.e., equal to the pipe size.

We have also assessed relation of the throughput gains to the End-to-End propagation time and to the bottleneck link transmission speed. Results show that, when compared to NewReno, ABE is able to maintain relatively robust performance with the increase of RTT. Further, ABE is more effective than NewReno in utilizing bottleneck bandwidth (Jagannathan, 2002), especially when the bandwidth is higher.

Fast probing mechanism: Fast Probing uses available bandwidth estimate to adaptively and repeatedly reset ssthresh. During fast probing, when the current ssthresh is lower than available bandwidth estimate, the sender resets ssthresh higher accordingly and increases cwnd exponentially. Otherwise, cwnd increases linearly to avoid overflow. In this way, fast probing probes the available network bandwidth for this connection and allows the connection to eventually exit slow-start close to an ideal window corresponding to its share of path bandwidth. The pseudo code of the algorithm, executed upon ACK reception, is as follows:

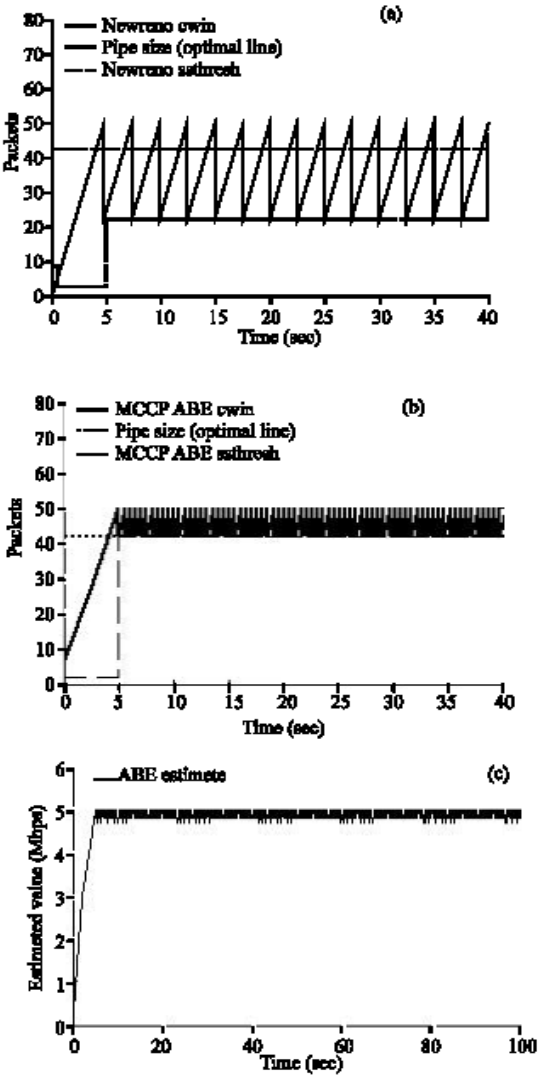

Fig. 2: Illustration of MCCP ABE and NewReno behavior with small buffer capacity, (a) Newrwno: cwin and ssthresh, (b) MCCP ABE: cwin and ssthresh, (c) MCCP ABE: estimate

```
if (DUPACKS are received)
switch to congestion avoidance phase;
else (ACK is received)
if (ssthresh < (ABE * RTT_min)
ssthresh = (ABE * RTT_min) /* reset ssthresh */
endif
if (cwnd > = ssthresh) /*linear increase phase*/
increase cwnd by 1/cwnd;
else if cwnd < ssthresh)
 /*exponentially increase phase*/
increase cwnd by 1;
endif
endif.
```

By repeating cycles of linear increase and exponential increase, cwnd adaptively converges to the desired window in a timely manner, enhancing link utilization in slow-start.

Non-Congestion Detection (NCD): In this study, we present a NCD mechanism that aims at detecting extra available bandwidth and invoking fast probing accordingly. In congestion avoidance, a connection monitors the congestion level constantly. If a MCCP sender detects non-congestion conditions, which indicates that the connection may be eligible for more bandwidth, the connection invokes fast probing to capture such bandwidth and improve utilization.

cwnd/RTTmin indicates expected rate in no congestion and ABE is the achieved rate. To be more precise, ABE is the achieved rate corresponding to the expected Rate 1.5 times RTT earlier. Thus, we must use in a comparison, the corresponding expected rate, that is (cwnd-1.5)/RTT_{min}. ABE tracks the expected rate in noncongestion conditions, but flattens, remaining close to the initial expected rate (ssthresh/RTT_{min}) under congestion. We define the congestion boundary as:

$$\text{Congestion Boundary} = \beta * \text{Expected Rate} + (1-\beta) * \text{Initial Expected Rate, where } 0 < \beta < 1$$

ABE may fluctuate crossing above and below the congestion boundary. To detect noncongestion, we use a noncongestion counter, which increases by one every time ABE is above the congestion boundary and decreases by one if ABE is below the congestion boundary. A pseudocode of the noncongestion detection NCD algorithm is as follows:

```
if (in congestion avoidance except for the initial 2 RTT){
if (ABE > Congestion Boundary){
no_congestion_counter++;
else if (no_congestion_counter > 0){
no_congestion_counter--;
if (no_congestion_counter > cwnd){
restart fast probing;
}else
{ no_congestion_counter = 0;
}
```

If the parameter β is greater than 0.5, the Congestion boundary line gets closer to expected rate. We can make this algorithm more conservative by setting $\beta > 0.5$. Even if the NCD algorithm can accurately detect noncongestion, there is always the possibility that the network becomes congested immediately after the connection switches to fast probing phase. Many of the TCP connections may decrease their cwnd after a buffer overflow and congestion is relieved in a short time period. The NCD in some connection may detect noncongestion and invoke fast probing. However, the erroneous detection is not a serious problem. Unlike exponential cwnd increase in slow-start phase of NewReno, the TCP connection adaptively seeks the fair share estimate in fast probing mode. Thus, if the network has already been congested when a new fast probing begins, the fast probing connection will not increase cwnd much and will go back to linear probing soon enough.

SIMULATION RESULTS

Premature exit from slow-start: Figure 3 shows cwnd dynamics under random packet loss during slow-start. The bottleneck link bandwidth is 100 Mb s^{-1}, 2 way propagation delay is 100 ms and the bottleneck buffer is equal to the BDP. When cwin/RTT_{min} reaches 2 Mb s^{-1}, a packet is dropped (assumed to be random loss, which may happen in the early stage of a connection over satellite or wireless links). The connection exits slow-start phase and enters congestion avoidance. Without NCD, cwnd increases slowly, one packet every RTT, requiring more than 60 sec for cwnd to reach BDP. With the help of NCD, the MCCP connection detects persistent noncongestion within a few seconds and then starts a new fast probing again.

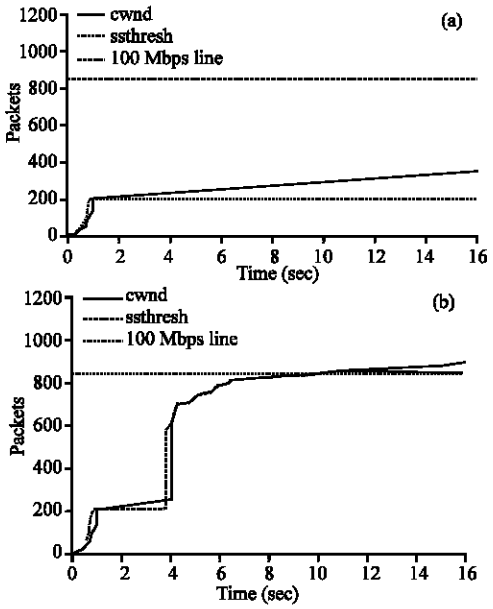

Fig. 3: cwnd dynamic under a random packet loss at slow-start phase, (a) TCP (without NCD), (b) MCCP (with NCD and fast probing)

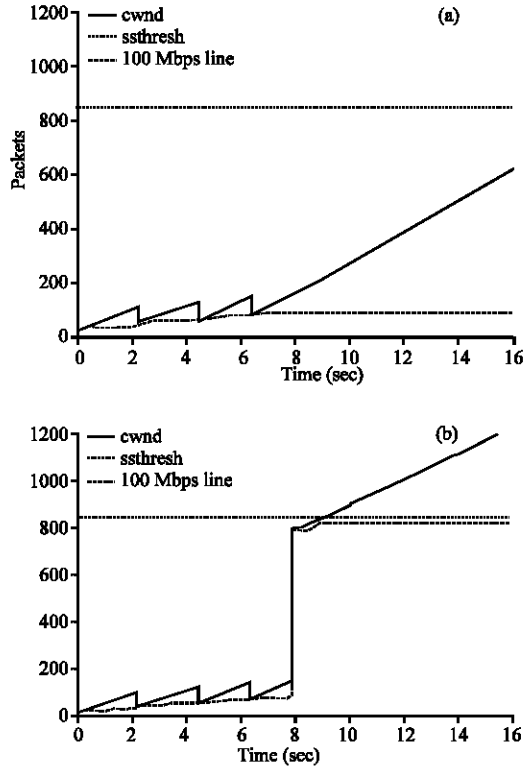

Fig. 4: cwnd dynamic when dominant flows are gone from the bottleneck router, (a) TCP (without NCD), (b) MCCP (with NCD and Fast Probing)

The throughputs of these 15 sec simulations are 30.3 Mb s^{-1} without NCD and 88.8 Mb s^{-1} with NCD and fast probing.

Dynamic bandwidth: To illustrate how MCCP behaves under dynamic bandwidth, Fig. 4 shows cwnd dynamics when nonresponsive UDP flows are gone from the path, causing extra bandwidth to become available. The bottleneck link bandwidth is 100 Mb s^{-1}, 2-way propagation delay is 100 ms and the bottleneck buffer is set to BDP. The nonresponsive UDP flows have disappeared from the path around 50 sec and the remaining flow is eligible to use the newly materialized bandwidth. Without NCD, the connection needs 60 sec to reach BDP. On the other hand, NCD detects the unused bandwidth within a few seconds and a new fast probing phase makes instant use of this unused bandwidth. Note that dynamic bandwidth due to other reasons stated in Introduction will induce similar behavior that helps to improve utilization.

CONCLUSION AND FUTURE WORK

In this study, we presented MCCP, a TCP modification that uses sender side intelligence to address the issues of highly dynamic bandwidth, large delays and random loss to support multimedia applications. Besides basic TCP scheme, MCCP incorporates 2 new mechanisms: fast probing and NCD. Fast probing enhances probing during slow-start and whenever noncongested conditions are detected. Fast probing converges to more appropriate ssthresh values thereby making better utilization of large pipes and reaching cruising speeds faster, without causing multiple packet losses. NCD is shown to be effective in detecting persistent noncongestion conditions, upon which MCCP invokes fast probing. The combination ensures that during Congestion Avoidance, MCCP can make quick use of bandwidth that materializes because of dynamic loads among other causes. In the future, we will evaluate MCCP further in terms of expanded friendliness, random loss, complex topologies and interaction with active queue management schemes.

REFERENCES

Allman, M.S. Floyd and C. Partridge, 2002. Increasing TCP's Initial Window, RFC 3390.

Floyd, S., 2003. High speed TCP for large congestion windows, Internet draft (Online). Available: Draft-ietf-tsvwg-highspeed-01.txt.

Hoe, J.C., 1996. Improving the startup behavior of a congestion control scheme for TCP. In: Proc. ACM SIGCOMM, pp: 270-280.

Jacobson, V., 1988. Congestion avoidance and control. ACM Comput. Commun. Rev., 18 (4): 314-329.

Jagannathan, S., 2002. End to End Congestion Control of Packet Switched Networks. In: Proc. IEEE Int. Conf. Control Appl., pp: 519-524.

Jain, M. and C. Dovrolis, 2003. End-to-End Available Bandwidth: Measurement Methodology, Dynamics and Relation with TCP Throughput. IEEE/ACM Trans. Networking, 11 (4): 537-549.

Katabi, D., M. Handley and C. Rohrs, 2002. Internet congestion control for future high bandwidth-delay product environments. In: Proc. SIGCOMM, pp: 89-102.