Transmission Control Protocol Based on Statistic Process Control

1,2Weifeng Zhao, 3Dawei Gao, 4Yandong An, 5Yantai Shu, 6M. Y. Sanadidi

1 First Author
School of Computer Science and Technology, Tianjin University, Tianjin, P.R.China, fgwy_zhao@163.com
2School of Computer and Information Engineering, Tianjin Institute of Urban Construction, Tianjin, P.R.China, fgwy_zhao@163.com
3,4School of Computer Science and Technology, Tianjin University, Tianjin, P.R.China, naokicn@hotmail.com, nuanran199@yahoo.com.cn
5Corresponding Author
School of Computer Science and Technology, Tianjin University, Tianjin, P.R.China, ytshu@tju.edu.cn
6Computer Science Department, UCLA, Los Angeles, USA medy@cs.ucla.edu

Abstract
In this paper, we study the distribution of RTT, and propose a new TCP congestion control algorithm that is based on Statistical Process Control - TCP SPC. Through simulation, measurement and theoretical analysis, we show that the RTT distribution can be approximated by a normal distribution in the case of heavy network load. Using RTT statistics, TCP SPC predicts the network load, and improves the performance of TCP. Compared with TCP Reno, TCP SPC in effect predicts the current load more accurately, distinguishes wireless channel packet loss from network congestion packet loss, overcomes the impact of wireless packet loss, and renders the adjustment of congestion window more timely and more appropriate. TCP SPC is transparent to routers and destinations, and thus is easy to deploy. We implemented TCP SPC in Qualnet as well as in Linux. The simulation and experimental results show that TCP SPC can be used in wired and wireless network with a wide range of packet loss rates, providing higher throughput as well as ensuring fairness and friendliness.

Keywords: Congestion Control, RTT Distribution, TCP, SPC, Wireless Network

1. Introduction
The Transmission Control Protocol (TCP) is one of the most important means of end-to-end congestion control in the Internet [1]. It provides reliable connections for most of today’s network applications. TCP relies on the Additive Increase and Multiplicative Decrease (AIMD) strategy to adjust the sender congestion window, and it uses packet loss as an indication of network congestion. As a result TCP Reno performs well in wired networks with negligible link error rates. But it has a lower performance in wireless networks, because it cannot distinguish between wireless link packet loss and network congestion packet loss. Further, AIMD is more appropriate for short paths. However, under high bandwidth-delay product paths, AIMD becomes inefficient, and often causes severe fluctuation of the source sending rate and the router queue size [2]. Such fluctuation degrades bandwidth utilization and produces a larger amount of congestion packet loss.

In the past decades, researchers attempted to improve the performance of TCP when facing wireless links with high random loss rates [3-7]. At the sender, there are two types of solutions. One is a cross-layer optimization tactic [3] wherein the sender adjusts the congestion window according to information collected by the MAC layer. And the other is an intra-layer optimization tactic: adding Loss Discrimination Algorithm in traditional TCP algorithms to discriminate wireless link packet loss and improve network performance [4].

The importance of RTT distributions in congestion control was recognized a while back, and researchers studied the subject to determine such distributions behavior. For example: [8] shows that RTT obey different distributions under different network load conditions. However, the measurements reported were obtained for internal networks only, and many years ago. The Internet has materially changed in topology and load since then.
In this paper, we use SPC methods [9] to propose a new TCP sender congestion avoidance algorithms — TCP SPC [10]. Some researchers have indeed improved TCP performance via SPC ideas. [11] uses an SPC approach to calculate a more appropriate RTO. NewReno-FF [12] also uses an SPC based sender side algorithm. In these early attempts, an EWMA filter and the SPC method were both deployed, and the assumptions of these two methods appear to be in conflict.

Compared with the traditional algorithms which use packet loss as signal of congestion, TCP SPC uses RTT statistics and a SPC method to predict the changes in a defined set of network states. TCP SPC adjusts the sender side congestion window depending on the current network state. The innovation in this algorithm is in its use of RTT statistics, rather than the “Smoothed RTT” values currently maintained and used in traditional TCP. TCP SPC does not use any cross-layer information. It is easy to deploy because TCP SPC requires the modification of only a small amount of code and only at the sender. Simulation and experimental results show that, compared to TCP Reno, TCP SPC can improve network performance in wireless network, and also works fine in a wired network.

This remainder of the paper is organized as follows. First a study of RTT distribution is presented in Section 2, and following that, we introduce the TCP SPC congestion control algorithm in Section 3. The implementation of TCP SPC in Linux is discussed in Section 4. In Section 5 we present the performance evaluation of TCP SPC, and our conclusions can be found in Section 6.

2. Analysis of RTT distributions

2.1. Experimental measurement

Since a TCP sender is not concerned with the specific route to the TCP receiver, the entire network can be viewed as a black box or alternatively the cloud shown in Figure 1. The input into the cloud is a stream of data packets that are sent to the TCP receiver, and the output is information regarding packet loss and RTT. This model is shown in Figure 1.

We relied on the Ping facility to measure RTT. A “pinger” sends a series of Internet Control Message Protocol (ICMP) Echo Request(s) to a destination host. The latter returns to the pinger a corresponding series of ICMP Echo Reply(s). The pinger host then calculates the RTT(s) according to information in the Reply packets.

Given the black box model of Figure 1, regarding destination host selection, we need to be concerned only with the geographic location and the host type. The selection criteria include: 1) a variety of host types: commercial host, academic host, etc; 2) a single host, instead of a host group; 3) diverse geographic locations: domestic host, foreign host. We use a host in Tianjin University (TJU) as the “pinger”. It connects to the internet by a 100Mbps full-duplex Ethernet. The destinations are located in Beijing, Hong Kong, the United States and other locations, and include commercial as well as academic hosts.

Figure 2 presents the traffic through a router over a 24 hours period. As expected, the network load changes consistent with the user’s daily work and rest rules. After reaching a peak around 23:00, we observe a decline to a minimum near 5:00am. Subsequently load increases gradually, and reaches another peak at 12:00 noon. In the afternoon the load reduces slightly. Then it starts climbing again over the next 24 hour cycle. There are two peaks which appear at the time of users rest time. One is at lunch break, and the other is after dinner.
Therefore, we divide the level of network load into three categories: heavy load, light load, and moderate load, which are consistent with the user’s internet time (from 11:00 to 14:00 and from 17:00 to 23:00), rest time (from 23:30 to 08:30) and transition time respectively.

The statistical results show that the RTT histogram collected from each destination has three kinds of shapes, similar to [8]. For example, Figure 3 is shows the three histograms for RTT data collected from TJU to Chinese University of Hong Kong (CUHK), for the three load levels respectively.

![Figure 3. RTT frequency histogram](image)

Thus we remark that the load level affects the shape of the RTT distribution, and in particular, the RTT distribution for the heavy load case is similar to a normal distribution. We use one-sample K-S test of non-parametric tests in SPSS software to verify if the 45000 RTT samples continuously under heavy load in two hours obey the normal distribution. The Sig. (both sides) is .000. Therefore the RTT samples do not obey a Normal distribution.

However, we found that it is a small proportion of RTT samples which cause the departure from the Normal distribution. So we removed the outliers, 2% of the samples that are greater than 100ms and less than 80ms. Repeating the K-S test on the remaining 98% of the RTT samples, we find that the Sig. (both sides) is .388, which is larger than 0.05. Therefore without the outliers, the remaining 98% of the original set do obey the Normal distribution. This shows that only a small percentage of the RTT data deviate from a Normal distribution. Therefore, we will proceed having determined that the distribution of RTT under heavy load is Normal, once we ignore the outliers above.

### 2.2. Simulation verification

We simulate using Qualnet 3.7 as an alternative approach for confirming whether RTT values obey a Normal distribution in a wireless network. In the simulation, we distribute the nodes randomly, separated in a way to ensure only adjacent nodes can directly transmit data to one another. We selected two topologies as detailed below.

Simulation 1: Five nodes compose a linear topology as in Figure 4 to transfer One FTP stream. We use a single-sample K-S test to validate the Normal distribution of the RTT samples. The results show that the proportion of RTT samples which obey a Normal distribution is 89%.

In Simulation 2, we increase the number of nodes to seven laid out in the dumbbell topology shown in Figure 5. Two FTP flows share the same intermediate nodes, but do not share the sending and receiving nodes. Otherwise, the simulation setup is the same as in Simulation 1. The results here show that the proportion of RTT samples obeying a Normal distribution is 86%.
2.3. Theoretical analysis

RTT mainly include two parts: 1) propagation times: which are mostly constant. 2) queuing time: which varies, and therefore is the main factor introducing variance in RTT. According to the model of Figure 1, from the perspective of a transport layer source, all the routers over which a packet passes can be viewed as one virtual router. Assume that the buffer size of the virtual router is \( n \) Byte, and each packet service time obeys the exponential distribution. When the network is under heavy load, the forwarding rate is lower than the sending rate at the source, and one can assume that the virtual router buffer space is completely occupied. Then, the “Time in system” of each packet can be modeled as the sum of \( n+1 \) independent exponential service times. The time in system therefore follows the Gamma distribution. According to the central limit theorem, when \( n \) is large, the Gamma distribution can be approximated as a normal distribution.

3. Principles of TCP SPC

3.1. Establishment of control chart

We sample \( N \) consecutive RTT values to calculate the mean \( m \) and the standard deviation \( \sigma \) according to formulae (1) and (2) below.

\[
\begin{align*}
    m &= \frac{\sum_{i=1}^{N} RTT_i}{N} \quad (1) \\
    \sigma &= \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (RTT_i - m)^2} \quad (2)
\end{align*}
\]

Then, we calculate the following control values of Control Charts.

- **UCL**(Upper Control Limit) = \( m + 3\sigma \)
- **LCL**(Lower Control Limit) = \( m - 3\sigma \)
- **UWL**(Upper Warning Line) = \( m + 2\sigma \)
- **LWL**(Lower Warning Line) = \( m - 2\sigma \)
- **UFL**(Upper Foco Line) = \( m + \sigma \)
- **LFL**(Lower Foco line) = \( m - \sigma \)
- **CL**(Centre line) = \( m \)

A Control Chart, as the one shown in Figure 6 can be drawn after calculating the control value.

![Figure 6. Control Chart](image)

With RTT values following a normal distribution, the probability that RTT samples fall in \([m-\sigma, m+\sigma]\), \([m-2\sigma, m+2\sigma]\) and \([m-3\sigma, m+3\sigma]\) are 0.6827, 0.9545 and 0.9974 respectively. If the RTT samples found in \([m-3\sigma, m+3\sigma]\) are randomly distributed over that interval, it indicates that the
network is in stable state. On the other hand if they exhibit a specific trend, it indicates that the network status has changed, and we must adjust the congestion window.

3.2. Monitoring the network load

We divide the RTT trends into four categories which are named network load criterions:

(1) Congestion state
   - Over-UCL: one sample locates above the UCL
   - Near-UCL: more than one samples locate between UCL and UWL
   - Jitter-Trend-Up: distance of 7 consecutive samples growing larger and larger

(2) Over-load state
   - Up-Excursion: more than one samples are consistently greater than CL
   - Point-Trend-Up: the value of 7 consecutive samples are more and more larger

(3) Under-load state
   - Over-LCL: one sample locates under the LCL
   - Near-LCL: more than one samples locate between LCL and LWL
   - Down-Excursion: more than one samples are less than the CL continuously
   - Jitter-Trend-Down: the distance of 7 consecutive samples getting smaller and smaller
   - Point-Trend-Down: the value of 7 consecutive samples getting smaller and smaller

(4) Chart invalidation
   - Stratum: 13 consecutive samples locate between UFL and LFL

After sampling a RTT, we estimate the network status in accordance with the following order (1) congestion state (2) over-load state (3) under-load state (4) Control Chart invalidation.

3.3. Congestion window adjustment scheme

TCP SPC congestion window adjustment is also divided into two phases: the slow start phase and congestion avoidance phase. In the slow start phase, TCP SPC uses the same method with TCP Reno to adjust the congestion window. In the congestion avoidance phase, TCP SPC uses a different strategy to adjust the congestion window:

(1) When the congestion state is identified, TCP SPC immediately halves the congestion window, so that the network can leave the congestion state as soon as possible. After resizing the window, TCP SPC remains in congestion avoidance and redraws the Control Chart.

(2) When conditions for the over-load state are met, TCP SPC reduces the congestion window to 3/4 of current window to avoid the network entering into a congestion state. After resizing the window, TCP SPC remains in congestion avoidance and redraws the Control Chart.

(3) When the under-load state is encountered, we increase the congestion window by 1 to take advantage of the unused network bandwidth.

(4) If 13 continuous RTT samples do not meet any of the above criterions, we believe that the network is in steady state. In order to make full use of idle bandwidth, TCP SPC increases the congestion window by one packet.

4. Implementation of TCP SPC

We implement TCP SPC in the Ubuntu 9.04 Linux with kernel version 2.6. We selected this version because it is suitable for comparing TCP SPC with other congestion control algorithms; e.g. wireless network protocols like TCP Westwood, delay based protocols like TCP vegas; and high bandwidth-delay protocols like HSTCP, STCP, TCP Bic and TCP Cubic.

The main struct, functions and interfaces of TCP SPC implementation are as follows:

- Define a struct "struct tcp_spc" to save the parameters, such as 4σ cordon, control limits.
- Modify the array size to 18. This array which is in “inet_connection_sock” struct is used to stored protocol data.
- Define three interfaces: ssthresh, init, and cwnd_event.

The main process of implement is shown in Figure 7.
5. Performance evaluation of TCP SPC

5.1. Scenarios and parameters

We evaluated the performance of the new algorithm using Qualnet 3.7 simulation software as well as a measurements testbed. Two scenarios in each methodology were used as follows:

Simulation scenario 1: 8 wireless nodes compose a linear topology to transfer One FTP traffic. Each wireless node is 200 meters away to ensure that only the adjacent two nodes can transfer packet directly.

Simulation scenario 2: 49 nodes arrange into a 7*7 grid structure shown in Figure 8. The traffics are in different directions.

Experimental scenario 1: 3 notebooks compose a two-hop topology to transfer One FTP traffic.

Experimental scenario 2: 4 notebooks compose a topology in Figure 9. Node 1 and node 2 send packets to node 4 through an intermediate node 3. There is no interference between node 1 and node 2.
5.2. Selection of sample number N

Collecting more RTT samples can increase the parameters accuracy. But the more samples, the more time it takes to calculate these parameters. Therefore, we have to balance the accuracy and sensitivity.

In simulation scenario 1, we studied throughput for different numbers of samples N, and different number of hops. The results of Figure 10 show that the throughput decreases with the increase of the hop number and the max throughput is achieved at N=13 regardless of the number of hops.

![Figure 10. Relationship of throughput and sample number N](image)

(a) 1 hop       (b) 3 hops        (c) 5 hops       (d) 7 hops

5.4. Effect of loss rate to throughput

We simulate the throughput under different packet error rate in simulation scenario 1. The result is shown in Figure 11.

![Figure 11. Relationship of loss rate and throughput](image)

From the results we can find that the throughput of TCP SPC is better than other protocols. Compared to the throughput of TCP Lite, TCP SPC has 58.5% improvement on average.

We studied the impact of non-congestion loss. We used experimental Scenario 1 with and without interference. Strong interference was generated by microwave. We recorded the throughput every 3 minutes. The results are shown in Figure 12 and Figure 13.

![Figure 12. Throughput without interference](image)

![Figure 13. Throughput with interference](image)
The two figures show that: 1) The throughput of the two protocols are generally similar under no interference, as it should be. 2) The throughput of TCP SPC is better than TCP Reno under wireless microwave interference. A large number of experiments show that the throughput of TCP SPC is higher than TCP Reno by 10 to 30%. 3) The throughput deteriorates still under interference and this is because lost packet retransmissions, which consumes a lot of bandwidth.

5.5. Effects of traffic flows

We studied throughput and fairness of TCP SPC and TCP Reno in simulation scenario 2. We collected aggregate throughput of all flows, and follow the fairness index in [13]: Fairness index = \( \frac{\sum x_i^2}{n \sum x_i^2} \).

Our simulations are run with 4, 6, and 8 traffic flows. The results are shown in Figure 14 and Figure 15.

![Figure 14. Relationship of flows and throughput](image1)

![Figure 15. Relationship of flows and Fairness](image2)

We can see from Figure 14 that TCP SPC achieves higher total throughput. Compared to TCP Lite, the average aggregate throughput under TCP SPC shows a 56.8% improvement. With the increase of number of flows, the aggregate throughput of TCP SPC is excellent compared to other protocols. So TCP SPC can adapt to the challenge of a large number of flows. Figure 15 shows that the higher throughput of TCP SPC is obtained without sacrificing fairness since all protocols exhibit equivalent fairness behavior.

5.6. Friendliness

In experimental scenario 2, we use TCP Reno to continuously send packets from node 1 to node 4. And we send packets from node 2 to node 4 by TCP Reno and TCP SPC interchangeably. Figure 16 is the throughput trends of link 1 when link 2 uses different protocols.

In Figure 16, the two curves are alternating up and down. In general, there is no obvious good or bad. So we believe that using TCP SPC does not affect the performance of other protocols.

![Figure 16. Friendliness](image3)
6. Conclusion

In this paper, we studied the distribution of RTT values and showed that RTT values rather obey the normal distribution in the case of heavy network load. We proposed a new TCP congestion control algorithm that uses SPC methods to adjust a TCP sender congestion window. The simulation and experimental results show that TCP SPC has better throughput in wireless network compared with TCP lite, and the throughput of the new algorithm is better than Reno under wireless interference. At the same time, the new protocol maintains good fairness behavior, and in addition is friendly to TCP Reno.

7. Acknowledgment

This research was supported in part by the National Natural Science Foundation of China (NSFC) under grant No. 61072063 and 61172063.

8. References