Some Results on State-based Analysis of TCP

Tuan-Anh Trinh, Mihály Krizsán and Sándor Molnár
High Speed Network Laboratory
Department of Telecommunications and Telematics
Budapest University of Technology and Economics
H-1117, Magyar tudósok körútja 2, Budapest, Hungary

E-mail: {tuan,krizsan,molnar}@ ttt-atm.ttt.bme.hu

***Abstract*--This paper reports some results on state-based analysis of TCP connections. During a connection, TCP stays in either of the following states: Slow Start, Congestion Avoidance, Fast Retransmit (Recovery) and Time Out. We use simulation to examine TCP state by state. We have developed a tool to investigate the behavior of TCP in different periods. At this stage, our tool can detect the beginning and the end of each period, collects some useful statistics for our state-based model. We also report some interesting observations while investigating TCP behavior at states, such as the evidences and the causes of the difference between the congestion window and the number of packets outgoing in the network.**

***Index terms*—TCP, modeling.**

I. INTRODUCTION

TCP modeling can be found in two main approaches: packet level and fluid level. One of the motivations for the packet level approach is the possibility of applying existing discrete-time models [1], [2], [5], [10]. Respectively, the motivation for fluid level model is the possibility of applying existing continuous-time models [3], [6], [7], [8], [9]. In both approaches, good points have been addressed and important, subtle results have been achieved. In [6], T. Ott *et al* used stochastic differential equations to model TCP behavior and first suggested the *square-root formula*. J. Padhye *et al* in [1] extends the model in [6] to capture Time Out. This model is widely accepted as one of the most accurate models for TCP Reno (in the case of bulk data transfer). We can also mention here the chaotic nature of TCP as suggested and examined in [3]. However, as TCP modeling is application-sensitive, a general purposed TCP model that is precise, yet simple, is still unavailable. This makes the TCP modeling task still very challenging. Another issue of TCP modeling is the types of modeling: black-box modeling and white-box modeling.

Black-box modeling approaches usually start from a theoretical model while white-box modeling approaches try to mimics TCP inherent operations based on some statistics. An example of white-box modeling is the well-known ON/OFF model for voice traffic. It has two states: IDDLE and SPEAK. If the speaker speaks, then it is in SPEAK state, and it is in IDDLE state otherwise. A natural question arises then: How about TCP? Our state-based model of TCP follows white-box modeling approach. We model TCP by its states. During a connection, TCP stays in either of the following states: Slow-Start, Congestion Avoidance, Fast Recovery, Exponential Back-off. TCP can jump from one state to another state in response to external events such as packet loss or Time Out. We consider how much time TCP stays in each state and the distribution of time elapsed at each state. We then consider the jumping probability from one state to another state. From the statistics, we can build a model to estimate TCP throughput. We have developed our technique in a tool called **TCP_ASD** (**TCP A**utomatic **S**tate **D**etection) that automates state analysis of TCP connections. This paper reports some results achieved from the tool and simulations of TCP.

The remainder of the paper is organized as follows. In Session 2 we describe the simulation setup that we use throughout this paper. In Session 3 we describe our state-based model of TCP in more detail. Session 4 describes the TCP_ASD tool. Session 5 presents some results from our analysis. Session 6 concludes the paper.

II. SIMULATION SETUP

For our analysis, we used a simple simulation setup as above. The access link bandwidth was set to 8 Mb/s, the access link delay was set to 1 ms. The bottleneck link bandwidth was set to 800 Kb with 100 ms delay. The packet size was set to 1000 bytes.

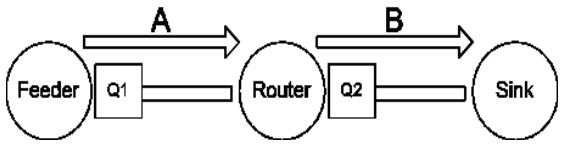

Figure 1. Simulation Setup

The receiver advertised window is set to 45 (big enough in order not to infer to our simulation). In this paper, we only examine TCP performance with Drop-Tail router.

III. A STATE-BASED MODEL OF TCP

We consider the dynamics of TCP state by state. After the hand-shake period, TCP starts in Slow-Start. In this period, the congestion window is increased in an exponential manner. From Slow-Start, TCP can jump into either of the following states: Fast Recovery, Congestion Avoidance and Exponential Back-off, depending on the external events. If the congestion window gets the slow start threshold without packet loss, then TCP will enter Congestion Avoidance. We call the probability that TCP jumps from Slow-Start to Congestion Avoidance by p_{sc} If packet loss happens then on the arrival of the third duplicate ACK, TCP will retransmit the lost packet and enter Fast Recovery. We call the probability that TCP jumps from Slow-Start to Fast Recovery by p_{sf} . Otherwise it enters Time Out. We call the probability that TCP jumps from Slow-Start to Exponential Back-off (or Time Out) by p_s . Let's suppose now that TCP is in Fast Recovery.

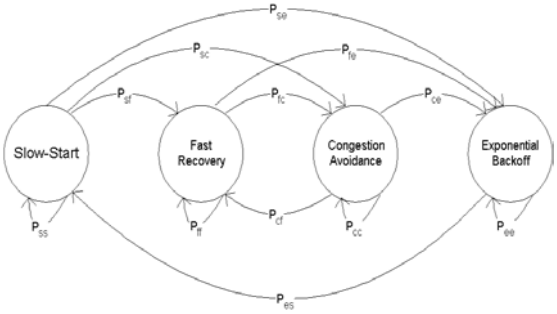

Figure 2. The model

Similarly, from Fast Recovery, TCP jumps to Congestion Avoidance with probability p_{fc} and to Exponential Back-off with probability p_{fe} . From Congestion Avoidance, TCP jumps to Exponential Back-off with probability p_{ce} and to Fast Recovery with probability p_{cf} . From Exponential Back-off, the only possibility for TCP is to jump to Slow-Start (with p_{es}). We also consider the distribution of time elapsed in each state. If the time spent at each state is exponentially distributed, then we have a Markov chain. The states of the Markov chain are the states of TCP itself. From the Markov chain we can derive the stationary behavior of TCP, such as throughput of TCP, as well as the exact dynamics of TCP.

IV. TCP AUTOMATIC STATE DETECTION TOOL

Our tool has two major parts. The first part is responsible for running simulation and preprocessing the data. This part also collects useful information from the simulations of TCP connections such as the congestion window and slow start threshold (*cwnd_* and *ssthresh_*). The second part is responsible for actually producing the results. We have implemented (in C++) a number of our algorithms [4] to illustrate a number of TCP metrics such as number of out-going packets, number of forward-going packets. The TCP_ASD tool uses the dynamics of the congestion window to detect state changes of a TCP connection, collects some statistics such as sojourn time at states, bytes sent at states for our state-based model. We also examine different TCP versions such as TCP Tahoe, Reno, New-Reno, and SACK. Finally the MATLAB scripts part is responsible for illustrating the dynamics of the mentioned processes.

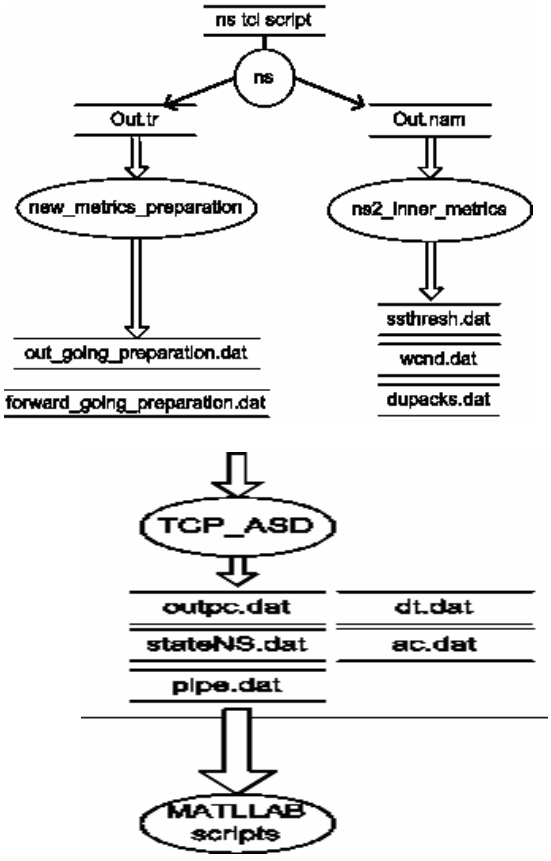

Figure 3. Flow diagram of our tool

V. SOME RESULTS

5.1 State detection

Figure 4 illustrates how state change is detected. The connection starts with a Slow Start. After the third duplicate ACK arrives, the congestion window is halved and TCP enters Fast Recovery. The receipt of the recovery ACK gets TCP out of Fast Recovery. At this

point there are two possibilities: Congestion Avoidance or Time Out. If only one packet is lost in the window of packets, then Congestion Avoidance follows. If multiple losses occur, then Time Out (or Exponential Back-off) follows. .

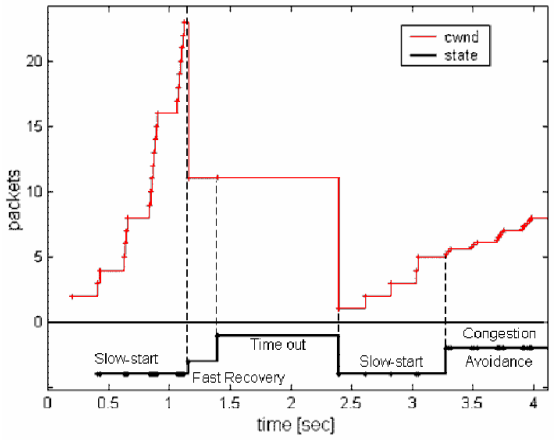

Figure 4. State detection of TCP

When the timer expires, the congestion window is set to 1 and Slow Start begins and the congestion window is increased in an exponential manner. If the congestion window gets the slow start threshold, TCP enters Congestion Avoidance.

5.2 Sojourn times

The first thing to mention here is that TCP state at Congestion Avoidance period in most of its time in our simulations. Since New Reno and SACK provides mechanisms to handle multiple losses in a window, there is no Time Out there. But it is not without cost.

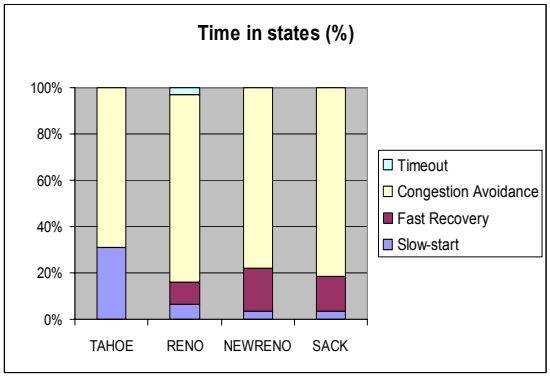

Figure 5. Sojourn times

As we can see in the Figure 5, the time New Reno and SACK stayed at Fast Recovery is significantly bigger that Reno. And we know that in Fast Recovery, there is not much effective data transmission (goodput). In our simulations with TCP Tahoe, no Time Out occurred. This is because TCP Tahoe is not so aggressive. Every single packet loss results in an empty pipe and Slow Start again.

5.3 Bytes Sent

We observe that Congestion Avoidance carried most of the data in a TCP connection. An important observation here is that New Reno and SACK did indeed improve in Fast Recovery (more data sent during this period). Since every single loss in Tahoe causes the TCP Slow Start again, the bytes sent in Slow Start in Tahoe is significantly bigger than other versions.

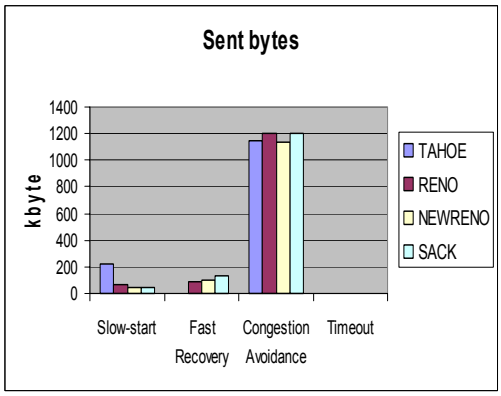

Figure 6. Sent bytes

5.4 Rates

Now let us consider the rate of sending data at each state.

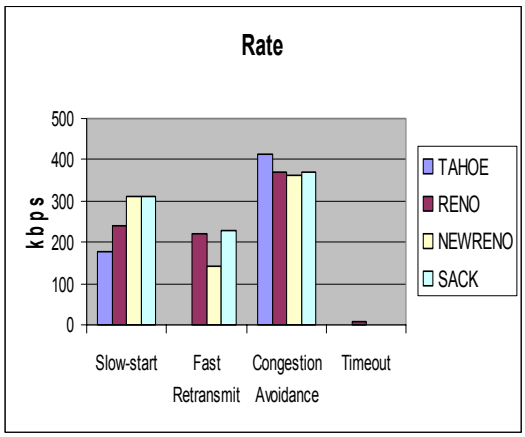

Figure 7. Rates

We observe that in Congestion Avoidance, TCP Tahoe outperformed other versions of TCP. It is not without cost. It performed worst in Slow Start.

5.5 An important observation on the number of out-going packets

The measurement of the number of out-going packets (*the pipe*) is quite straightforward: upon each acknowledgment at the sender, the pipe equals the number of packets sent but not yet acknowledged till that time, including the packets that are sent just after receiving that acknowledgement. If a packet is dropped at

the router, we should deliberately subtract the pipe by one since one packet has left the pipe. There are two kinds of slow start to consider. The first slow start is the slow start right after the handshake period and slow start after time out. Slow-Start in the first period appears to behave normally in this period (see Fig. 8). The lower figure shows how packets are sent and ACKs are received at the sender. The upper figure shows the dynamics of the *cwnd* and out-going packets. We have two things to mention here. First, the congestion window increases in an exponential manner after each round trip time as long as TCP stays in Slow-Start.

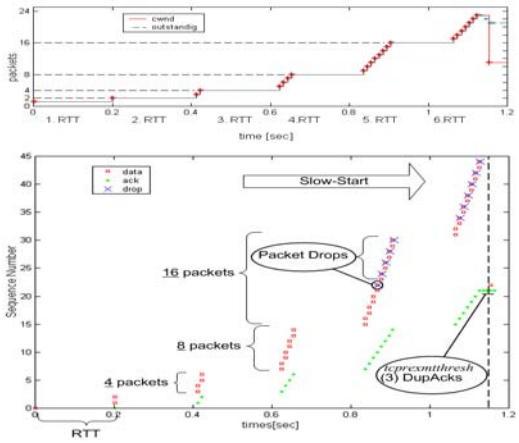

Figure 8. Normal slow start

Second, in this period, the *cwnd* and out-going packets are indeed the same because they are integers (1,2,4,8,16 until TCP sender senses packet loss). The conclusion (observation) for this part is that the dynamics of the number of out-going packets *exactly* reflects the dynamics of the congestion window process. This period ends when the *cwnd* reaches the threshold (*ssthresh_*) or a packet loss occurs. In our simulation, this slow start ended after spent 5RTT+ and when 3 dup ACKs received. The second type of slow start we consider in this paper is the slow start after time out. This slow start, by specification, should behave like normal slow start. However, in presence of interesting coming ACKs from the receiver, it behaves somewhat differently.

In this slow start, unlike the first slow start, there is a difference between the *cwnd* and the actual number of packets in the network (packets in pipe). Our tool tries to specify this number and give us the evidence and reason for this difference. The evidence of the difference is clear from the Figures 9 and 10. We find that the reason for the difference is due to the packets that already left the network and are cached at the sender's buffer but are not yet processed because of loss packets in between, we call them the "holes". We can also observe "ACK jump" effect in this period. Notice that this is not delayed ACK, since we deactivated this option in our simulations. This is because the sender sends the packets that are already cached at the receiver buffer. In this Slow Start the congestion window evaluation is 1, 2, 3, 5, and so on, after each round trip time. The actual number of out-going packets are 0,2, 4, 5, and so on. Clearly, the increase is not in an exponential manner. There is a difference between this metric and the congestion window. The sender, however, doesn't have that knowledge, and resends these packets using slow start mechanism, causing the difference (Fig. 10). Now let us take a closer look at our case. The following packets are dropped at the router: 22 24 26 28 30 34 36 38 40 42 44. The following packets are cached but not yet processed at the receiver: 23 25 27 29 31 32 33 35 37 39 41 43. After receiving 3 duplicated ACKs for packet 22, the sender resent packet 22 (Fast Retransmit) and halves its congestion from 22 to 11 and enters Fast Recovery. However, it does not receive enough additional duplicated ACKs to inflate the window. The arrival of ACK number 23, the sender exits Fast Recovery and the *cwnd* becomes 11,0909, but by this time, the pipe is already empty. TCP sender enters Time Out. As the Timer expires, the TCP sender exits Time Out and sends packet number 24 (loss packet) entering Slow Start phase. The last sent packet was packet number 44. The first thing to be observed here is on "needlessly resent packets". Upon receiving ACK number 25, the sender sends two packets: 26, 27. However the packet 27 was already cached at the receiver, so we called it "needlessly resent packet". We can observed in Fig. 9 that packets number 29, 31, 32, 33, 35, 37, 39, 41, 43 are all "needlessly resent packets". The second thing to be observed here is the ACK "jump". As we can see in Fig. 9 , packet 24 is not ACKed. Instead, two ACKs arrive for packets. This is not because of delay ACK as we mentioned above. It is also due to packets cached at the receiver. The third thing to be observed here is "3 dupACKs but NO retransmission", which seems to be not "normal" in TCP performance. This is, again, can be traced back to the fact that some packets are resent but were already cached at the receiver.

The conclusion for this part is two fold: First, in Slow Start phase, for the first Slow Start the *cwnd* perfectly follows the number of out-going packets. For Slow Start after Time Out, the *cwnd* does not increase in exponential manner and does not always match the number of out-going packets. However, the difference is minor. We also presence observations in this Slow Start and gave explanation for them. Basically, all can be traced back to the fact that, TCP needlessly resends the packets that are already cached at the receiver.

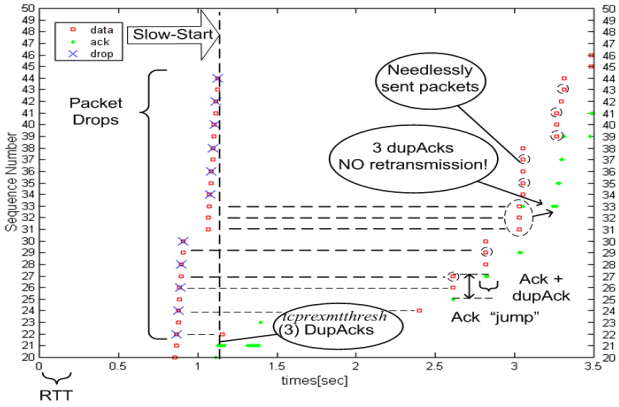

Figure 9. Slow start after Time Out

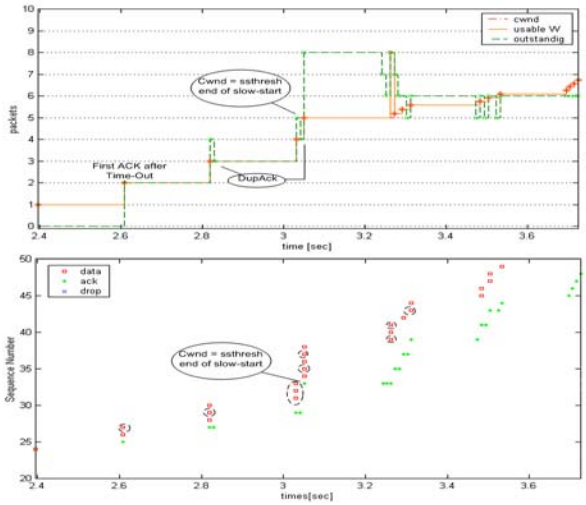

Figure 10. Out-going pakets vs. cwnd

VI. CONCLUSION

We have presented a state-based analysis of TCP through simulations. We have developed a tool for our analysis. We used the tool to collects useful statistics for our state-based model of TCP. Finally, we showed an important observation of TCP dynamics. Analysis and validation of our state-based model is left as our future work.

VII. REFERENCES

[1] Jitendra Padhye *et al*, Modeling TCP Reno Throughput: A Simple Model and its Empirical Validation, SIGCOMM'98.

[2] A. Kumar, " Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link", IEEE/ACM Transactions on Networking, 1998.

[3] A. Veres, M. Boda, The Chaotic Nature of TCP Congestion Control, INFOCOMM 2000, Tel Aviv, Israel.

[4] T. A. Trinh, T. Éltető, On Some Metrics of TCP, LCN 2000, Tampa, Florida, USA, November 2000.

[5] N. Cardwell, S. Savage, T. Anderson, Modeling TCP Latency, INFOCOMM 2000.

[6] T. Ott, J.H.B. Kemperman, M. Mathis, The Stationary Behavior of Ideal TCP Congestion Avoidance, Bell Lab Technical Report, 1996.

[7] V. Misra, W. Gong and D. Towsley. A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED, SIGCOMM 2000.

[8] Schwefel, Behavior of TCP-like Elastic Traffic at a Buffered Bottleneck Router, INFOCOMM 2001.

[9] Steven Low *et al,* Dynamics of TCP/RED and a Scalable Control, INFOCOMM 2002.

[10] Peerapol Tinnakornsrisuphap, Armand Makowski, Limit Behavior of ECN/RED Gateways Under a Large Number of TCP Flows, INFOCOMM 2003.