PERFORMANCE ANALYSIS OF TCP-RENO AND TCP-SACK: THE SINGLE SOURCE CASE

JEAN-LUC DOREL † MARIO GERLA

UCLA COMPUTER SCIENCE DEPARTMENT
3732H BOELTER HALL
LOS ANGELES CA 90095, USA
{JLD,GERLA}@CS.UCLA.EDU

ABSTRACT

This paper explores the behavior of both TCP-Reno and TCP-SACK under a simple scenario. In this scenario a single TCP source transmits continuously its packets over a single bottleneck node characterized by its queue size, bandwidth and propagation delay. The analysis allows to derive the performance of TCP. When the bandwidth x propagation delay pipe becomes very large the utilization tends to 75% of the bottleneck throughput. When the queuing delays are predominant the utilization tends to 100% since the queue is never empty. In the transient analysis we show how the initial phase of the session can degrade the performances. These results are assessed through simulation.

Key words: TCP flow control, TCP-Reno, TCP-SACK.

1 INTRODUCTION

The current Internet is a complex network to analyze since it carries out traffics obeying to many different resource allocation schemes (such like: TCP-Tahoe, TCP-Reno, TCP-SACK, TCP with explicit congestion information, UDP, RSVP...) under a wide range of network parameters (i.e., router's queue size, link throughput, propagation delay). This paper is an attempt to analyze a subset of what is the current Internet to provide results for more complex and complete analysis. More precisely this paper analyzes the behavior of both TCP-Reno and TCP-SACK when operating over a single bottleneck node in the case of a single source.

† This work was supported by French MOD/DRET contract 96.398.

This paper is organized as follow: We begin with a presentation of the TCP-Reno and TCP-SACK algorithms. Next we described the model of the system in section 3 and analyzed it in section 4. A comparison with simulation results follows in section 5. Similar studies are compared in section 6 and conclusions arising from this study are given in section 7.

2 ALGORITHMS

TCP-Reno is basically built on five flow control algorithms that adapt the throughput of TCP by modifying its sending window. First the sending window is set to 1 and the source starts in the so-called *Slow-Start* algorithm [1] in which the window is increased by one each time an acknowledgment is received until a threshold window is reached. When this threshold is reached TCP enters the *Congestion Control* algorithm [1] and stay in this mode until the retransmission timer expires. This event triggers the Slow-Start mode by setting the window to one. In the Congestion Avoidance mode the window is increased by one once all packet acknowledgments have been received. It is important to note that every packet is acknowledged by the receiver and a source emits a packet only if an acknowledgment has been received: this is the *Self-Clocking* algorithm of TCP [1]. This algorithm makes that acknowledgments are roughly transmitted at the rate packets arrive at the receiver so that the rate at which the sender transmits is approximately the rate of the bottleneck node.

When the window is too large, losses occur in the congested node. Losses are detected through the *Fast Retransmit* algorithm when a small number of duplicated acknowledgments (in general three) are received by the sender. After a loss, Reno does not resume with a Slow-Start as in Tahoe. Rather, Reno wants to keep advantage of the self-clocked acknowledgments in the pipe. However as a loss has occurred it is necessary to reduce the window size W, reached before loss was detected. For that purpose the source resumes with a new window size of $W/2$ after acknowledgments in excess have been received and discarded (this is the *Fast Recovery* algorithm [2], [3]). The problem with this algorithm is that if multiple losses occur during the same window it is very hard to know when start the $W/2$ train of self-clocked acknowledgments. This is because the number of remaining

acknowledgments is unknown and the number of loss is not the assumed one packet loss. This leads to desynchronization, time-out and poor performance as shown in [4]. The *Selective Acknowledgment* option [5] has been designed to overcome this problem of Reno. With SACK, when the receiver detects holes in the sequence of packets it advises it to the sender by selectively acknowledging blocks of packets that have been received correctly. With this procedure the sender knows exactly the number of acknowledgments in the pipe and thus can synchronize with the W/2 window size.

As stated in [6] : "adding SACK to TCP does not change the basic underlying congestion control algorithms." SACK has to be regarded as a more robust response to a regime of high successive losses rather than a modification of the basic performance characteristics of Reno. As consequence the subsequent analysis does not focus on the SACK mechanism and assumes that the synchronization after loss detection is perfect i.e., the sender is always able to recognize the number of holes in the train of acknowledgments. This assumption is true for Reno as far as the number of loss is at most one for a given window, and it is always true for TCP-SACK as far as acknowledgments are not lost.

3 SYSTEM MODEL

The system (i.e., the source, router, links and receiver) is analyzed through the following model:

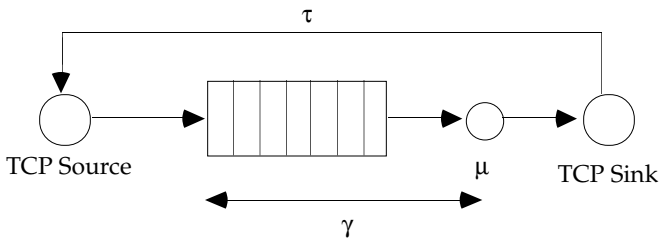

Figure 1: System Model

The chain of routers and links from the source to the destination is modeled through its bottleneck node (i.e., the node connected to the slower link). The throughput of the node is μ and its queue capacity is γ (including the packet being transmitted). The scheduling policy is FIFO and the dropping policy is drop-tail. The delays other than bottleneck transmission time and queuing are

modeled through a single deterministic parameter τ. These parameters are without units meaning that e.g., if μ is expressed in packets/s then τ is in seconds γ in packets.

We assume that the source has always packets to transmit and that the window advertised by the receiver is always greater than the congestion window. In other words the performance bottleneck is the network rather than receiver host.

Taking into account that all parameters are known and that the source transmits packets continuously one can study this system through a deterministic analysis.

4 THROUGHPUT ANALYSIS

4.1 SYSTEM BEHAVIOR

The source starts with a window of one in the Slow-Start mode. After the threshold window is reached the Congestion Avoidance algorithm applies. The window is regularly increased and fills the bandwidth x propagation delay pipe ($\mu\tau$ in our model). When the $\mu\tau$ window is reached packets are accumulated in the queue. When the window reached $\varphi=\mu\tau+\gamma$ the next increase provokes a congestion and a packet loss. These two phases are illustrated in figure 2.

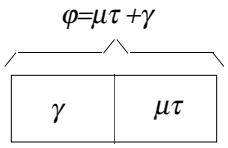

Figure 2: Network Pipe

In our model there is only one loss because after the packet in excess has been dropped, following packets arrive at the bottleneck rate of $1/\mu$ (this is the self clocking effect of TCP) and overflow does not occurred since a packet is de-queued every $1/\mu$. The loss will be detected after a round-trip time by the reception of three duplicate acknowledgments (c.f., the Fast Retransmit algorithm). The window is then divided by 2 to prevent further congestion and the Congestion Avoidance mode resume by increasing the window by one each time the current window has been completely

acknowledged. This process lasts until the window reaches φ and a new increase provokes an overflow of the queue.

The system appears to be a succession of *cycles* each one consisting in an increase of the window from $\varphi/2$ to φ (in the Congestion Avoidance mode) followed by a loss and a recovery through the Fast Retransmit and Fast Recovery algorithms. These cycles are preceded, at the beginning of the session, by a single Slow Start phase. In the *steady state analysis* we assume that the session is long enough and consider this initial phase as negligible. Thus the mean of throughput of the session is equivalent to the mean throughput of a single cycle. The assumption is relaxed in the *transient analysis*.

4.2 STEADY STATE ANALYSIS

We define the throughput utilization λ as the number of packets n transmitted during a cycle divided by duration d of the cycle. This ratio is expressed as portion of the raw bandwidth of the server (i.e., λ is a percentage of μ).

As the window increases from $\varphi/2$ to φ the number of packet transmitted during a cycle is[1]:

$$n = \sum_{\varphi/2}^{\varphi} i = \frac{3\varphi^2 + 6\varphi}{8}. \qquad \text{(Eq.1)}$$

Concerning the duration of the cycle, we have to distinguish two cases whether the queue is empty or not when the source resumes after a loss detection. The queue is empty if the window $\varphi/2$ is not large enough to fill the $\mu\tau$ pipe, which happened when $\mu\tau \geq \gamma$. When $\gamma > \mu\tau$ the window is large enough to fill the $\mu\tau$ pipe (i.e., $\varphi/2 > \mu\tau$) meaning that there are still packets in the queue and the time to transmit packets is longer than if the queue is empty. More precisely when the queue is empty a given window is transmitted in a round-trip time ($rtt = 1/\mu + \tau$) and when the queue is not empty a given window is transmitted in a round-trip time plus the time to transmit the packets in the queue.

[1] assuming the lost packet is negligible

In the case where $\mu\tau \geq \gamma$ the situation is as follows:

Window	Duration
$\varphi/2$	rtt
$\mu\tau$	rtt
$\mu\tau+1$	$rtt+1/\mu$
$\mu\tau+\gamma$	$rtt+\gamma/\mu$

Table 1: Window transfer times when $\mu\tau \geq \gamma$

Thus the duration for an entire cycle (from window $\varphi/2$ to φ) is:

$$d = (\varphi - \frac{\varphi}{2})rtt + \frac{1}{\mu}\sum_{\gamma} i$$
$$= \frac{\varphi}{2} rtt + \frac{\gamma^2 + \gamma}{2\mu}. \quad \text{(Eq.2)}$$

By combining Eq.1 and Eq.2, the throughput utilization is:

$$\lambda \equiv \frac{n}{\mu d} = \frac{3\varphi^2 + 6\varphi}{4(\varphi(1+\mu\tau) + \gamma^2 + \gamma)},\ \mu\tau \geq \gamma. \quad \text{(Eq.3)}$$

In the case where $\gamma > \mu\tau$, window $\varphi/2$ starts with R packets remaining in the queue:

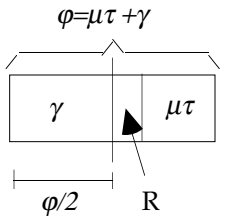

Figure 3: Remaining packets when $\gamma > \mu\tau$

R is defined as: $R = \frac{\varphi}{2} - \mu\tau = \frac{\gamma - \mu\tau}{2}$. The window transfer times are as follows:

Window	Duration
$\varphi/2(>\mu\tau)$	$rtt+R/\mu$
$\varphi/2+1$	$rtt+R/\mu+1/\mu$
φ	$rtt+R/\mu+\varphi/2\mu$

Table 2: Window transfer time when $\gamma>\mu\tau$

The duration of the cycle can then be defined as:

$$d = \frac{\varphi}{2}(rtt + \frac{\gamma - \mu\tau}{2}) + \frac{1}{\mu}\cdot\sum_{\varphi/2} i, \qquad \text{(Eq.4)}$$

and the throughput utilization obtained by combining Eq.1 and Eq.4 is:

$$\lambda = \frac{3\varphi^2 + 6\varphi}{3\varphi^2 + 6\varphi} = 100\%,\ \gamma > \mu\tau. \qquad \text{(Eq.5)}$$

This result is not surprising since the queue is never empty. However if we consider the lost packet is not negligible, by subtracting 1 to Eq.1, Eq.5 becomes:

$$\lambda = \frac{3\varphi^2 + 6\varphi - 8}{3\varphi^2 + 6\varphi},\ \gamma > \mu\tau. \qquad \text{(Eq.6)}$$

4.3 BEHAVIOR AT THE LIMITS

From Eq.6 it is clear that the throughput utilization tends to 100% when γ is very large.

When $\mu\tau$ becomes very large, the throughput utilization based on Eq.3 is:

$$\begin{aligned}\lim_{\mu\tau\to\infty}\lambda &= \lim\frac{3\varphi^2 + 6\varphi}{4(\varphi(1+\mu\tau) + \gamma^2 + \gamma)} \\ &= \lim\frac{3(\mu\tau)^2}{4(\mu\tau)^2} \\ &= 75\%.\end{aligned} \qquad \text{(Eq.7)}$$

This result is easy to understand considering that the window evolves from $\mu\tau/2$ (queue is negligible) up to $\mu\tau$ at a linear increase every rtt: the resulting mean window $\frac{3}{4}\cdot\mu\tau$ is sent in a round trip time, which corresponds to a 75% utilization of μ.

4.4 TRANSIENT ANALYSIS

The importance of the transient behavior increases as the quantity of information to transmit decreases since small duration implies that the first Slow-Start becomes more significant. We study the impact of this first Slow-Start by evaluating its contribution in terms of number of packets transmitted and duration to the whole session throughput.

In [6] the duration of the Slow-Start phase for the case where $3\gamma > \mu\tau$ is defined as $rtt \cdot (\log_2 \varphi / 2)$ and the number of packet transmitted during this phase is φ. Assuming the user wants to transfer an information of size Q, the time to transfer Q-φ packets at the steady state throughput is $d_s = \frac{\mu(Q-\varphi)}{\lambda}$, and the resulting throughput utilization as a function of Q is:

$$\lambda(Q) = \frac{Q}{\mu(d_s + rtt \cdot (\log_2 \varphi / 2))}. \qquad \text{(Eq.8)}$$

5 SIMULATION RESULTS

Simulations have been made using the ns package [7] in the topology illustrated in figure 1. First we compare the results obtained for TCP-Reno and TCP-SACK with those predicted by Eq.3 for various queue capacities (μ=100, τ=1).

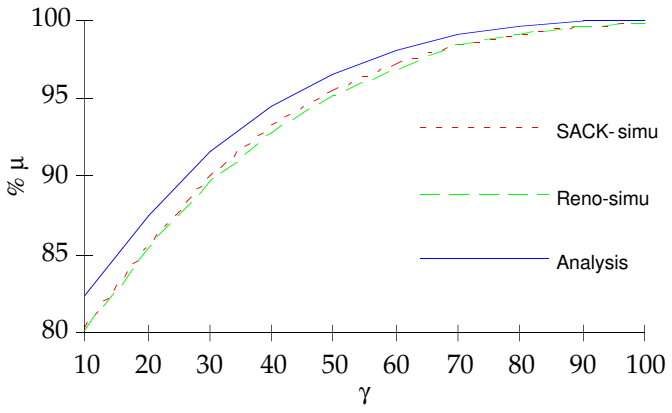

Figure 4: Throughput utilization vs. γ

Figure 4 shows the very good correlation between the analysis and the simulation. The difference is within 2.5%. These results also illustrate that the utilization tends to 100% when the buffer size is very large while it decreases below 80% for relatively large $\mu\tau$, which is consistent with the limits obtained in section 4.3.

One could also see the very close performance between TCP-SACK and TCP-Reno. This is due to the very long simulation duration (10000 u.t.) that hide the Slow-Start transient phase at the beginning of the session and during which SACK outperforms Reno in managing multiple losses. Simulation presented in [8] confirms that multiple losses mainly occur during the Slow-Start phase. The impact of this first Slow-Start is illustrated in figure 5 where the duration of the simulation is set to 100 u.t. (μ=100, τ=1).

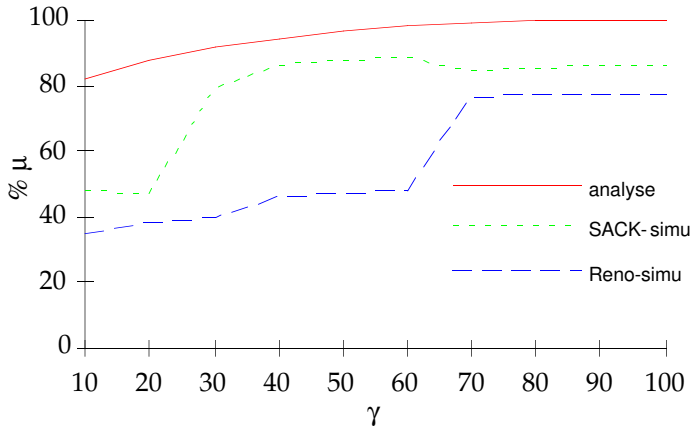

Figure 5: Throughput utilization for short simulation

When the simulation duration is shorter the effect of the first Slow-Start is more visible and the ability of TCP-SACK to better handle losses is clear.

The following figure illustrates the impact of Q (μ=100, τ=1 and γ=50) for a SACK connection.

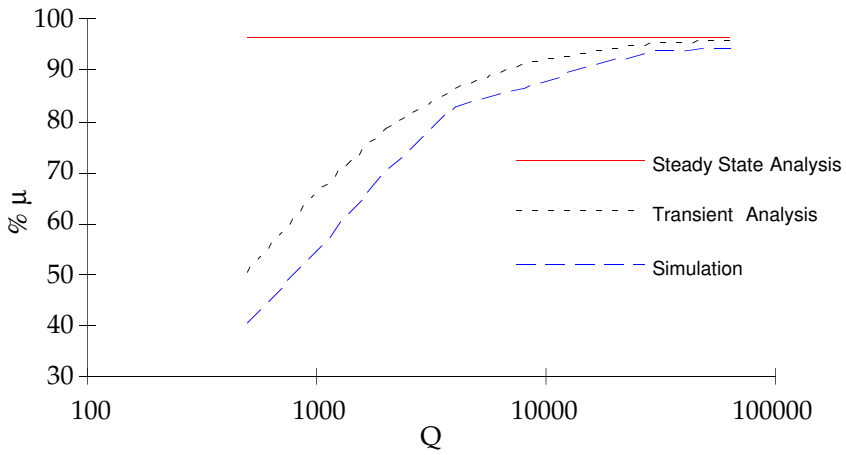

Figure 6: Influence of Q

The resulting throughput utilization clearly improves as the quantity of information to transmit increase. The difference between transient analysis and simulation is within 10% and the analysis catches well the evolution of the system as Q increase. When the quantity of information to transmit is very large the performances are those of the steady state analysis given by Eq.3.

6 RELATED WORK

The analysis in [6] uses the same model as us. However the analysis of Reno is quiet different. First they analyzed the amount of packets transmitted as a function of the duration of a cycle, which imply a more complicated expression than ours in Eq.1. The duration of a cycle for the case where $\mu\tau>\gamma$ is itself split into (i) the duration to fill the $\mu\tau$ pipe and (ii) the duration to fill the queue. Our analysis combines both and gives a single expression (Eq.2). The match between the analysis in [6] and simulation is within 2%, which is similar to our (2.5%). The case where $\gamma>\mu\tau$, the limit behavior and the transient behavior are not analyzed in [6].

7 CONCLUSION AND WAY FORWARD

This paper has proposed an analysis for TCP on a bottleneck node with a single source. The main conclusion arising are:

1) The analytic expression (Eq.3) of the throughput of TCP appears to be very close to the simulation results (the difference is within than 2.5%).

2) When the bandwidth x propagation delay pipe becomes very large the utilization tends to 75% of the bottleneck throughput as a result of the linear increase of the window from $\varphi/2$ to φ.

3) When the queuing delays are predominant ($\gamma > \mu\tau$), the utilization is 100% since there always packets in the queue.

4) Under this model SACK and Reno behaves similarly since only one loss occurs when the window reach φ.

5) However the is not true during the transient initial phase where SACK is clearly better in managing multiple loss. We showed how this initial phase affects the performance as a function of the quantity of data to transmit.

This analysis aims at been used in more complex models that analyze the performance of TCP-Reno and TCP-SACK with multiple sources.

8 REFERENCES

[1] V. Jacobson : "Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM' 88*, Standford, USA, August 1988.

[2] V. Jacobson : "Modified TCP congestion avoidance algorithm", message to the end2end working group, April 1990.

[3] W. R. Stevens : "TCP/IP illustrated Volume 1 : The Protocols", **Addison-Wesley**, 4th printing, 1994.

[4] K. Fall and S. Floyd : "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", to appear in Computer Communication Review.

[5] M. Matthis et al : “TCP Selective Acknowledgment Options”, RFC 2018, April 1996.

[6] T.V. Lakshman, U. Madhow : "Performance Analysis of Window-based Flow Control using TCP/IP : Effect of High Bandwidth-Delay Products and Random Loss", *HPN 94*, Grenoble, June 1994.

[7] LBNL’s Network Simulator (ns) is available via: http://www-nrg.ee.lbl.gov/ns/.

[8] S. Floyd : “Simulation Tests”, LBNL report, October 1996.