BOTTLENECK-QUEUE BEHAVIOR: How much can TCP know about it?

L. Mamatas[1], V. Tsaoussidis[1] and C. Zhang[2]

[1] Dept. Of Electrical and Computer Engineering, Demokritos University of Thrace, Greece
[2] Dept. Of Computer Science, Florida International University, USA

Abstract

We start off with an analysis on the effect of queuing delay on throughput. Departing from there, we claim that RTT alone is a better estimator than throughput. We investigate the potential of RTT-based metrics to approach the bottleneck queue behavior from TCP and we characterize the accuracy of our estimation.

Keywords

TCP, RTT, Queue

1. Introduction

End-to-end congestion avoidance and control as well as fair network resource management would have great benefit, had the TCP sender known the behavior of the bottleneck queue. Several methodologies have been developed to estimate bandwidth and bottleneck queue based on temporary measurements of throughput, inter-packet gap, or RTT. For example, TFRC [Handley, M., Floyd *et al.* (2003)] calculates throughput via a throughput equation that incorporates the loss event rate, round-trip time and packet size. TCP-Vegas [L. Brakmo and L. Peterson (1995)] estimates the level of congestion using throughput-based measurements. TCP-Vegas demonstrates that measurement-based window adjustments is a viable mechanism, however, the corresponding estimators can be improved. In TCP-Westwood [C. Casetti, M. Gerla *et al.* (2002)], the sender continuously measures the effective bandwidth used by monitoring the rate of returned ACKs. TCP-Real [V. Tsaoussidis and C. Zhang (2002)] uses wave patterns: a wave consists of a number of fixed-sized data segments sent back-to-back, matching the inherent characteristic of TCP to send packets back-to-back. The protocol computes the data-receiving rate of a wave, which reflects the level of contention at the bottleneck link. Bimodal congestion avoidance and control mechanism [P. C. Attie, A. Lahanas *et al.* (2003)] computes the fair-share of the total bandwidth that should be allocated for each flow, at any point, during the system's execution. TCP-Jersey [K.Xu, Y.Tian *et al.* (2004)] operates based on an "available bandwidth" estimator to optimize the window size when network congestion is detected. The Packet-Pair technique [S. Keshav (1991)] estimates the end-to-end capacity of a path using the difference in the arrival times of two packets of the same size traveling from the same source to the same destination. Recently, several investigations have been also carried out regarding the accuracy of the proposed estimators, like [M. Jain and C. Dovrolis (2004)].

Initially, we do not attempt to introduce a new estimator but rather to characterize how accurately TCP estimators can approach the queue behavior during communication. We

discuss pitfalls of throughput estimation and we emphasize further on behavior-estimation methodologies based on RTT measurements. We attempt to address the following issues:
- *Can TCP sender detect flow synchronization?* Synchronized flows do not allow for end-to-end detection of queue size for all participating flows. We estimate *packet position* per flow. We note that the prescribed behavior of TCP sources leads to a prescribed packet position in the queue when synchronization is present, depending on the number of participating flows in the system.
- *Can we estimate accurately the queue behavior in the absence of synchronization*? When the participating flows utilize large windows and packets are presumably (due to lack of synchronization) interleaved, the max-packet-delay per window may give sufficient feedback on the current queuing delay.
- *Does the accuracy depend on the window size and the scale of packet interleaving?* When the number of participating flows increases, windows are becoming smaller and samples do not suffice to select a value for max-packet-delay which is close to the real queuing delay. Sampling technique needs then to be adjusted to a predetermined number of packets. The more we increase the sample, the better accuracy we achieve but the less responsive a protocol can be due to further abstracting of the sampling scale.

2. Throughput vs RTT

First we extend the analysis model of [D.-M. Chiu and R. Jain (1989)] by taking into account the role of bottleneck queue. Consider a simple network topology shown in figure 1, in which the link bandwidth and propagation delay are labeled. In our scheme, *n* TCP flows share a bottleneck link with capacity of *bw*, and the round trip propagation delay is $RTT_0 = 2 * (src\_delay + delay + sink\_delay)$. Since our focus in this subsection is on the overall system behavior, we define the aggregated congestion window size at time *t* as:

$$cwnd(t) = \sum_{i=1}^{n} cwnd_i(t) \quad (1)$$

where $cwnd_i(t)$ is the window size of the i^{th} flow. Consequently, the system throughput at time *t* can be given by the following equation:

$$throughput(t) = \frac{cwnd(t)}{RTT(t)} = \frac{cwnd(t)}{RTT_0 + qdelay(t)} \quad (2)$$

where *qdelay(t)* is the queuing delay at the bottleneck router. As can be seen from (2), the throughput is not only a function of the congestion window, but also a function of the queuing delay, which was not incorporated into the analyses in [D.-M. Chiu and R. Jain (1989), J. Padhye, V. Firoiu *et al.* (1998)].

Assume all flows are in the additive increase stage. First consider the case where *cwnd(t)* is below the point *knee* [D.-M. Chiu and R. Jain (1989)]:

$$cwnd_{knee} = RTT_0 \cdot bw \quad (3)$$

Then there is no *steady* queue build-up[1] in R_1 (i.e. $RTT(t) = RTT_0$), and according to (2), the throughput grows in proportion to *cwnd*. The bottleneck capacity is not fully utilized until *cwnd* increases to $cwnd_{knee}$.

[1] There could be *temporary* queue build-up in this scenario, due to the traffic burstiness. This is neglected to simplify our analysis.

If *cwnd(t)* increases further beyond $cwnd_{knee}$, however, the system displays different dynamics. The bottleneck queue starts to build up, after the bottleneck capacity is saturated. Rewrite *cwnd(t)* as:

$$cwnd(t) = cwnd_{knee} + \Delta w(t) \quad (\Delta w(t) > 0) \quad (4)$$

Since the bottleneck link can transmit at most $cwnd_{knee}$ packets in one RTT_0 (see (3)), $\Delta w(t)$ packets will linger in the queue. Hence the steady queuing delay at the bottleneck will be:

$$qdelay(t) = \Delta w(t) / bw \quad (5)$$

Intuitively, the system throughput is bounded by the physical capacity *bw*, in spite of the increase of *cwnd(t)* beyond the knee, because *qdelay(t)* in the denominator of (2) grows as well. This is confirmed by the following computation:

$$\begin{aligned} throughput(t) &= \frac{cwnd_{knee} + \Delta w(t)}{RTT_0 + qdelay(t)} \\ &= \frac{RTT_0 \cdot bw + qdelay(t) \cdot bw}{RTT_0 + qdelay(t)} \\ &= bw \end{aligned} \quad (6)$$

The system dynamics can be continuously described by equations (4) – (6), until the queue length $\Delta w(t)$ reaches the maximum buffer size, i.e. when *cwnd* touches the point *cliff* [2]

$$cwnd_{cliff} = (RTT_0 + \max qdelay) \cdot bw \quad (7)$$

TCP senders then multiplicatively decrease their congestion window, after packet losses due to buffer overflow are detected.

Although the above analysis is based on a simplified model, it provides useful insights into the dynamics of congestion control, and a guideline for protocol improvement. Such guidelines can be found in [V. Tsaoussidis and C. Zhang (2005)]. The computation of (6) demonstrates that increasing *cwnd* beyond the knee does not enhance further the system throughput, but only results in increasing queuing delay. Can we use throughput to monitor network conditions? One can detect, perhaps, the knee threshold. Experience with TCP-Vegas has shown that the level of flow throughput at the time the threshold was crossed cannot be uniformly detected by all flows. Therefore, adjustments based on this information may lead to unfair bandwidth sharing. However, while estimators using throughput for measuring congestion may fail, estimators based on the RTT measurements could provide more accurate indication of the network condition.

3. ESTIMATING BOTTLENECK QUEUE SIZE

We focus on the estimation of the bottleneck queue size at the sender. We claim that using an appropriate RTT-based mechanism, the sender can measure with sufficient accuracy at what percentage the bottleneck queue is full. Such information may enhance the sophistication of the transport protocol regarding:

- the detection and avoidance of flow synchronization.
- its capability to distinguish congestive from non-congestive errors.
- its flexibility to implement a corresponding back-off or keep-on strategy, when the queue is getting full or remains unexploited, accordingly.
- the possibility to apply an adaptive scheme relevant to the detected network conditions.

[2] The intuitive concept of knee and cliff was first introduced in [D.-M. Chiu and R. Jain (1989)]. Here we give an analytical expression.

Our analysis on RTT-based estimation of queue behavior departs from a three-case categorization of network-traffic. Our main target is to investigate if, how and when specific packet-RTT measurements could correspond to the level of queuing delay experienced in the system. Maximum queuing delay corresponds to the max queue length. Naturally, our categorizing criterion is statistically oriented: What is the level of packet interleaving in the system? Increased interleaving leads to higher probability of experiencing max delay by all participating streams. On the contrary, synchronization leads to wrong estimation of max queuing delay. For example, a flow can have always its packets in the start of the queue and consequently follow a wrong estimation of an empty queue. We claim that this situation can be predicted when packet position in the queue follows the pattern of TCP window adjustments. That is, the expected position under conditions of flow synchronization would be a certain position in the queue. Given the window increase strategy of TCP, the expected position would follow a corresponding increase as well.

We examine different sampling techniques for measuring RTT. We show that measuring every RTT (ERTT) to estimate the queue dynamics may fail when synchronization exists; however, synchronization itself can be occasionally detected. Window-based sampling (WRTT) of largest RTT fails for small windows; and a predetermined number of packets works only when the link is fully utilized.

Otherwise, when there are periods of unutilized queue, a technique based on measuring *MaxRTT* for a predetermined number of packets (PRTT) may fail. *In conclusion, our sampling methodology corresponds to three types of congestion: (i) persistent congestion where packets are interleaved, (ii) transient congestion with small windows and synchronized flows and (iii) transient congestion with unsynchronized flows.*

Our estimators are currently based on the following assumptions:

- When *sample RTT* reaches *maximum RTT*, we assume that the bottleneck's buffer is full (when the measured *maxRTT* is taken out of a sufficient sample[3]).
- Similarly, when *sample RTT* approaches the *minimum RTT*, we assume that the bottleneck's buffer is Empty.

We introduce initially the Occupied Queue Size (OQS) index. The OQS index records the fraction of the bottleneck queue, which is full, based on the current packet position[4] in the queue:

$$OQS = (Sample\_RTT - MinRTT) / (MaxRTT - MinRTT)$$

The OQS index gets values between 0 and 1 since $minRTT<maxRTT$ if the number of packets per window $n>1$, packets following the same route.
When OQS gets the value 0, we estimate an empty buffer (*sample RTT=MinRTT*). Similarly, OQS=1 indicates a full buffer (*sample RTT=MaxRTT*) and hence OQS=EstimatedQueueLength.
For the sake of the experiments, when the bottleneck-queue length is known, we use the following metric:

[3] We investigate here what is a *sufficient sample*.
[4] For which the RTT has been measured

PP= Estimated_Queue_Length x (Sample_RTT – MinRTT) / (MaxRTT – MinRTT) =>

$$PP = EQL \times OQS$$

The Packet Position (PP) metric is therefore the product of the OQS index and the bottleneck-queue's length. The PP metric reveals the position of the last acknowledged packet in the bottleneck queue. We use this metric in contrast to the real position of the packet, in order to initially evaluate our estimator. However, we note that the estimated packet position is always equal to or less than the real packet position. We also estimate the mean and maximum deviation from the real position, categorizing the distinctive effects of different scenarios to deviation.

4. EXPERIMENTAL METHODOLOGY

4.1 Evaluation Plan

We have implemented our evaluation plan on the ns-2 network simulator. The network topology used as a test-bed is the typical single-bottleneck *dumbbell*, as shown in Figure. 1. The link's capacity at the receivers (bw_3) is 1Mbps. We used equal number of source and sink nodes. The simulation time was fixed at 60 seconds, a time-period deemed appropriate to allow protocols to exploit their dynamics.

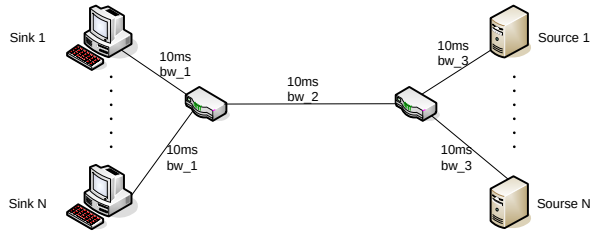

Figure 1. Simulation topology

In our scenarios, ftp flows are entering the system within the first two seconds. All flows are fixed, during the rest 58 seconds.

	Bw_1	**Bw_2**	**Bw_3**
Scenario 1	1 Mbps	100 Mbps	1 Mbps
Scenario 2	100 Mbps	10 Mbps	1 Mbps

Table 1. Experiments

We carried out experiments using the above two scenarios (Table 1). We adjusted values for Bw_1, Bw_2 in order to move the bottleneck in different queues (queue 3 or queue 1). We traced all the packets of the last flow (the 10^{th}) and record all the positions they got in all available queues. We use these results for the evaluation of the OQS and PP estimators. All queues have 50 packets length.

In order to evaluate how efficiently our mechanisms can work, we have also experimented with dynamic scenarios with graduated contention increase/decrease and different queuing

algorithms (like RED [S. Floyd, and V. Jacobson (1993)]). The results were similar with the ones reported.

5. EVALUATION RESULTS

5.1 Scenario 1 (bw_1=1 MBps, bw_2=100Mbps)

In the following figure (figure 2), we observe the behavior of the three queues in contrast to our estimation mechanism. In this experiment, we measure every RTT (ERTT sampling). We show that this estimation mechanism works very well, under the given network conditions. In fact, the estimation mechanism follows queue 3. In figure 3, we show the per-flow mean and max deviation of our estimation to the actual position of the packets. Consequently, we show that our estimation works effectively by the viewpoint of all ten flows. We got similar results using a RED queue.

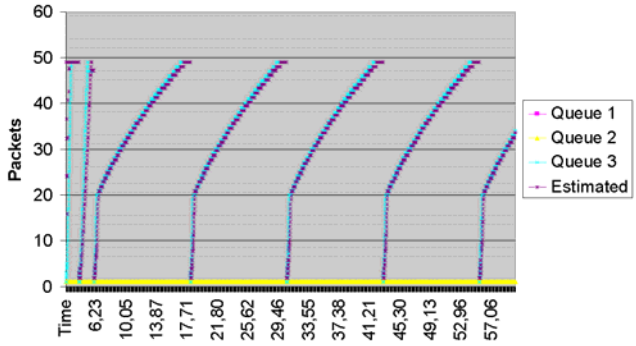

Figure 2. DropTail Queue

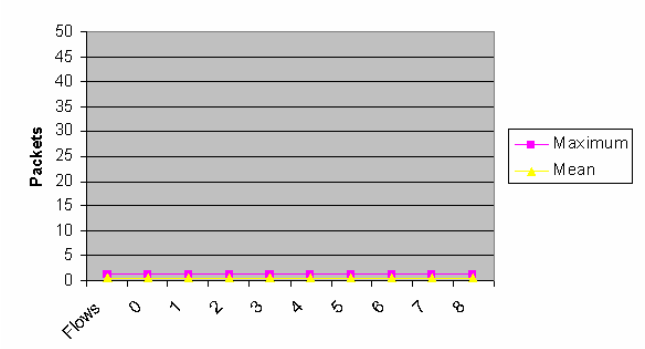

Figure 3. Maximum & Mean Deviation

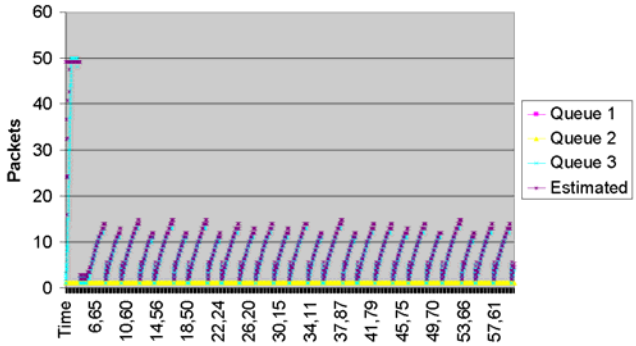

Figure 4. RED Queue

5.2 Scenario 2 (bw_1=100 MBps, bw_2=10Mbps)

When we used a first link of 100Mbps and a second of 10Mbps, we noticed that our estimation follows the first queue (using both DropTail and RED – Figures 5, 7). The mean deviation of our estimation (figure 6) is under five packets. The scenario with graduate contention decrease (figure 8) points out that the measurement gets refined with less number of flows.

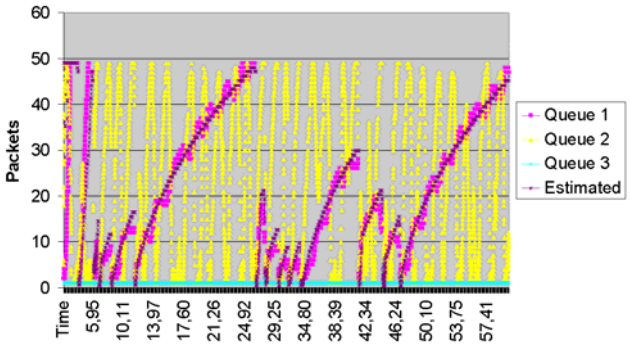

Figure 5. DropTail Queue

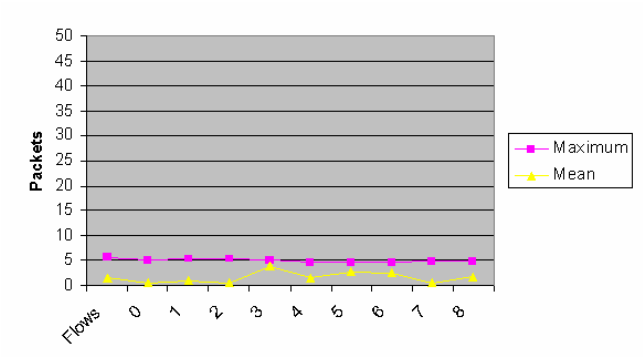

Figure 6. Maximum & Mean Deviation

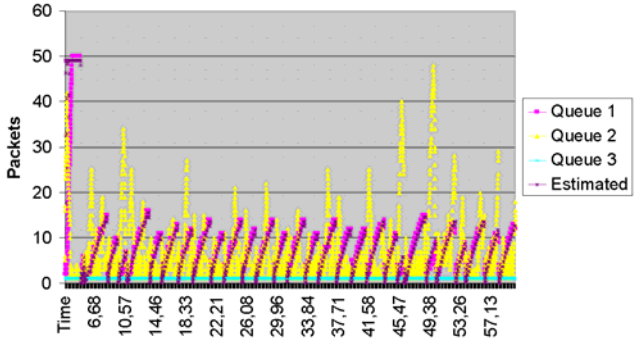

Figure 7. RED Queue

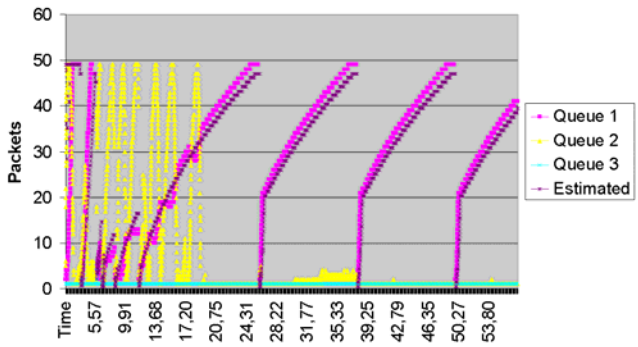

Figure 8. DropTail Queue with Contention Decrease

6. CONCLUSIONS & ONGOING WORK

We investigated the capability of different estimators to discover the bottleneck queue behavior. We conclude that RTT-based estimators occasionally fail to capture the queue dynamics. Depending on the conditions, one estimator may outperform another. The results call for a new design where the estimator adapts its strategy to detected network conditions. We intent to graft such an estimator to TCP sender.

7. REFERENCES

P. C. Attie, A. Lahanas and V. Tsaoussidis (2003), *"Beyond AIMD: Explicit fair-share calculation"*, In Proceedings of ISCC 2003, June, 2003.

L. Brakmo and L. Peterson (1995), *"TCP Vegas: End to end congestion avoidance on a global Internet"*, IEEE Journal on Selected Areas in communication, 13(8):1465--1480, October 1995

C. Casetti, M. Gerla, Saverio Mascolo, M.Y. Sansadidi, and Ren Wang (2002), *"TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks"* In Wireless Networks Journal 8, 467-479, 2002

D.-M. Chiu and R. Jain (1989), *"Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks"*, Computer Networks and ISDN Systems, 17(1):1-14, 1989.

S. Floyd, and V. Jacobson (1993), *"Random Early Detection gateways for Congestion Avoidance"*, IEEE/ACM Transactions on Networking, 1(4):397-413, August 1993.

Handley, M., Floyd, S., Pahdye, J., and Widmer, J (2003), *"TCP Friendly Rate Control (TFRC): Protocol Specification"*, RFC 3448

M. Jain and C. Dovrolis (2004), *"Ten Fallacies and Pitfalls in End-to-End Available Bandwidth Estimation"*, ACM Internet Measurements Conference (IMC), Sicily Italy, October 2004.

S. Keshav (1991), *"Congestion Control in Computer Networks"*, PhD Thesis, UC Berkeley, published as UCB TR 91/649

J. Padhye, V. Firoiu, D. Towsley, and J. Kurose (1998), *"Modeling TCP Throughput: A Simple Model and its Empirical Validation"*, In Proceedings of ACM SIGCOMM '98, August 1998

V. Tsaoussidis and C. Zhang (2002), *"TCP-Real: Receiver-Oriented Congestion Control"*, Computer Networks Journal (Elsevier), Vol. 40, No. 4, November 2002.

V. Tsaoussidis and C. Zhang (2005), *"The dynamics of responsiveness and smoothness in heterogeneous networks"*, IEEE Journal on Selected Areas in Communications, (JSAC) Special issue on Mobile Computing and Networking, March 2005

K.Xu, Y.Tian and N.Ansari (2004), *"TCP-Jersey for wireless IP communications"*, IEEE JSAC, vol.22, no.4, pp.747-756, 2004.