A simulation study of passive inference of TCP rate and detection of congestion

Mouhamad Ibrahim
INRIA ENS-Lyon
France
Mouhamad.Ibrahim@ens-lyon.fr

Eitan Altman
INRIA Sophia Antipolis
France
Eitan.Altman@sophia.inria.fr

Pascale Primet
INRIA ENS-Lyon
France
Pascale.Primet@ens-lyon.fr

Giovanna Carofiglio
Alcatel-Lucent Bell Labs
France
Giovanna.Carofiglio@alcatel-lucent.com

Georg Post
Alcatel-Lucent Bell Labs
France
Georg.Post@alcatel-lucent.com

ABSTRACT

In this paper, we propose and implement a mechanism for network simulator-2 (ns-2) to estimate the individual rates of TCP flows and to detect their incipient congestion. Rate estimation as well as congestion detection are based on measurements collected passively by a monitoring node which is located on the intermediate paths between TCP source destination pairs. While estimating the rate of a given flow relies on measuring its interpacket times as well as the size of its received packets, the congestion detection sub-mechanism analyses the variability of its growth rate over a sliding window to decide on a incipient congestion. Our proposed method does not rely on estimating the RTTs to detect congestion, nor on ACK packets traversing the backward paths to collect measurements samples. Moreover, it does not impose any constraint regarding the location of the monitoring router with respect to bottleneck links. Throughout the various simulations that we have conducted, our method has shown high efficiency in detecting the occurrence of incipient congestion on the monitored flows over relatively short time periods. The method is useful for monitoring and controlling the rates of large TCP flows passing through an autonomous system.

Keywords

TCP, Network simulator ns-2, rate estimation, passive measurement, congestion detection

1. INTRODUCTION

TCP (Transmission Control Protocol) is the most commonly used transport protocol in a large number of applications: File transfer, web browsing, peer-to-peer applications, emails, ... A TCP connection, termed also as TCP flow, consists of a sequence of packets that share common identifiers such as common source and destination IP addresses and common TCP port number. Depending on the flow average size and its lifetime duration, the networking community has classified TCP connections as mice for short lifetime connections and as elephants for long lifetime connections. While most of the TCP connections in the Internet are mice with a small volume of transferred data (80% of flows), the small amount of elephant flows (5% of flows) is responsible for the largest amount of transferred data in terms of bytes (80% of the load) [7].

TCP connections are characterized as being elastic in the sense that their transmission rates are dependent on the congestion level of the network. Indeed, when a TCP source sends a burst of packets to the destination, it waits back for an acknowledgement (ACK) packet from the destination to decide whether to proceed with further transmissions. The time interval that elapses between the packet transmission instant and the instant when the sender receives back its ACK from the destination is defined as the round trip time (RTT) of the flow. The RTT consists of a fixed delay which represents the packet transmission time over the end-to-end path plus its queuing delay in the network. The queuing delay is variable and reflects the level of congestion of intermediate buffers on the path between the TCP source destination pair [14].

While congestion detection at a given TCP source is handled by the TCP protocol through missed ACKs, detecting congestion at a router is carried out by an active queue management (AQM) technique that operates directly on the router buffer by measuring and monitoring its average queue size. Meanwhile, a question arises on the possibility of an intelligent router to detect remotely an incipient congestion at another distant router over their shared connections. This is of interest for applications related to network monitoring, anomaly detection and remote active queue management. Detecting remote congestion through passive measurements is the aim of our current paper which proposes real time tools to carry it out and which can be implemented in a real router as well as in the ns-2 simulator. The passive measurements would be gathered by the monitoring router which is located somewhere on the forward paths of TCP

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GAMECOMM 2009, October 23, Pisa, Italy
Copyright © 2009 ICST 978-963-9799-70-7
DOI 10.4108/ICST.VALUETOOLS2009.7639

source-destination pairs.

Our method is designed to operate on large TCP flows (elephant flows) and can be seen as being composed of two sub-mechanisms. The first sub-mechanism deals with the rate estimation of the flow and the estimation smoothing, while the second sub-mechanism deals with negative trends detection based on the smoothed rate samples. Rate estimation as well as rate degradation detection are based on per flow packet measures which are collected by the monitoring router. In particular, rate estimation is based on measuring the interpacket times as well as the size of received packets, and it is carried out using only the data packets passing on the forward paths between the sources and destinations.

The implementation of our method as well as the data traces used for the validation of its various mechanisms were carried out using ns-2. Nonetheless, our implementation in ns-2 is restricted to the use of information that would be available at the router in a real system.

The main contributions of our paper are: i) proposal of a method to smooth the estimated rate samples of TCP flows to promote rate change detection; ii) development of a method to detect negative trends, and hence incipient congestions, using the smoothed rate samples; iii) implementation and validation of the proposed mechanisms in ns-2.

The remainder of the paper is structured as follows. Section 2 discusses some related works while Section 3 describes an existing mechanism to estimate individual TCP rates. Section 4 proposes a technique to smooth the estimated rate samples. Based on the smoothed samples, a non-parametric trend detection technique is presented in section 5 to detect incipient congestions of the monitored rates. Section 6 describes briefly the technical implementation of the method in ns-2 while Section 7 presents and discusses some of the simulation results. Last, Section 8 concludes the paper.

2. RELATED WORK

The study of passive measurement techniques to infer the characteristics of TCP connections passing through an intermediate monitoring router has been widely considered by the networking research community over the last decade. Most of these works focus on inferring and analyzing the variability of the RTTs and the congestion windows (*cwnd*) of TCP flows [12, 11, 10, 14]. However, none of these studies aims to detect incipient congestion on the passing flows based on the observed behaviors.

Katabi and Blake presented in [12] a passive measurement technique to detect the presence of a shared congested link upstream of the monitoring point as well as the bottleneck router link speeds. To detect the sharing of congestion, they have shown that the entropy of the aggregate interpacket times is lower when the flows are congested than if they do not share a congested link. Interpacket time is defined as the time from the beginning of one data packet to the beginning of the next data packet in the same connection (Hereafter, we will use the previous definition of interpacket time for the rest of the paper).

[11] proposed a simple methodology based on two techniques to measure the RTTs. The first technique is applicable to TCP caller-to-callee flows, and it relies on measuring the time interval during the three-way handshake messages at connection setup. The second technique is applicable to data transfer from the callee-to-caller when relevant information are available to the monitoring point. In particular, during the slow-start phase, the monitoring point can measure the time spacing between the first and second bursts of sent data packets to provide a rough estimation of the connection RTT. Nevertheless, the second technique is more subject to delay jitter in the network which would bias the RTT estimate.

In [10], the authors presented a passive measurement technique to infer and keep track of the congestion window of TCP senders as well as of their RTTs. The measurements are gathered by monitoring the data packets passing from the sender-to-receiver direction as well as the ACK packets passing on the reverse direction. To estimate the current sender *cwnd*, the technique makes use of observed receiver-to-sender ACKs sequence numbers. In addition, it needs to track the timeout events at the sender by monitoring the out-of-sequence transmissions. To estimate the RTTs, the monitoring point needs to measure at first step the trip delay from it to the receiver and then back to it. The RTT estimate will hence be the sum of the first trip delay plus the second trip delay from the monitoring point to the sender then back from the sender to the monitoring point. However, there are several subtle challenges that the method needs to cope with to handle accurate estimation of the *cwnds* and the RTTs.

In [14], Lance and Frommer presented a spectral analysis algorithm which estimates the RTTs of a given TCP connection by analyzing the sequence of interpacket times at a given monitoring point. The technique which is based on Lomb periodogram requires at least 256 interpacket times (257 packets) to generate an initial estimate of the connection RTT. The proposed method does not make any assumption regarding the position of the monitoring router, the flavor of TCP, and does not rely on ACK packets. A detailed list of these works can be found in [13] and the references therein.

In this paper, we are considering a different approach from the previous studies since we are attempting to detect incipient congestion of large TCP flows by detecting the degradation of their rates. Our method does not impose any constraint regarding the location of the monitoring router with respect to bottleneck links. Moreover, it does not rely on ACK packets traversing the backward paths to collect measurements samples. Note that accessing the ACK packets is not always possible for an intermediate router since the forward and backward paths of flows do not always use the same sequence of links. For instance, over a sample of $33,000$ TCP flows, it has been observed in [13] that the monitoring router was not able to see ACKs for 20% of the flows.

Our proposed method does not rely on RTTs estimate to detect congestion. Actually, acquiring a relatively accurate estimate of a flow RTT in the real Internet is difficult to achieve giving the various options of TCP flavors such as delayed and selective acknowledgments (SACKs) [2]. Furthermore, the variability of the measured RTTs due to congestion is difficult to detect unless a highly accurate estimator is employed. For instance, based on measurements made on traces gathered within Sprint Ip backbone, RTTs variation during a connection lifetime was shown to be less than one second for $75 - 80\%$ of the connections [10].

3. FLOW RATE ESTIMATION

The first step in our remote congestion detection algorithm is to estimate the individual rate of an elephant flow

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

arriving at the entry of the monitoring router. We assume that a TCP flow identified as being elephant and admitted for monitoring is already in progress after some elapsed time since its start time. This elapsed time would represent the time needed to identify the size of the flow. In practice, one would define an elephant (or large) flow as a flow that either has sustained over some predefined time period threshold or it has sent at least a burst of 10 data packets.

To carry out individual rate estimation, we opted to use a simple estimator which has been already proposed by Stoica el al. in order to estimate the rates of flows arriving at edge routers [15]. For a given flow i, its estimated rate at the arrival time of its n^{th} packet at the monitoring router is written as:

$$rate_i(n) = (1-exp(\frac{-T_n}{K}))\frac{PktSize_n}{T_n}+exp(\frac{-T_n}{K})rate_i(n-1) \quad (1)$$

where $PktSize_n$ and $T_n = t_n - t_{n-1}$ represent respectively the received packet size and the interpacket time computed at the current packet arrival time t_n. K is a time constant, in seconds, that determines the reactiveness of the filter.

Observe that the previous estimator is an exponential weighted moving average filter which assigns variable weights depending on the measured interpacket times through the function $(1 - exp(\frac{-T_n}{K}))$. For different values of the time constant K, Figure 1 illustrates the behavior of the function $(1 - exp(\frac{-T_n}{K}))$ when the interpacket time T_n varies in the range $[2.10^{-4}, ..., 0.2]$ seconds. For instance, for a packet size of 1500 Bytes, the lower value of 2.10^{-4} seconds would represent the interpacket time of two successive packets of the same flow received by the monitoring router on a link of 60 Mbps. The upper value of 0.2 seconds would represent a RTT of that flow.

As the curves of Figure 1 illustrate, when $K < 0.3$ seconds, the filter is highly reactive to the variations of the measured interpacket time T_n even when T_n takes small values. Consequently, the rate estimation will show high variability making later trend detection a difficult task. On the other hand, when $K > 0.5$ seconds, the reactiveness of the filter is slow which would result in poor tractability to the variations of the real rate. In the rest of the paper, we consider the previous filter of Equation (1) with a time constant K equal to 0.4 seconds.

Observe that a scalable approach to estimate the rate of Equation (1) would rely on sampling the interpacket times of a given flow. Evidently, this solution would be attractive since it will minimize the burden of measuring the interpacket time at each packet arrival. Our work on this direction is still in progress and the choice of a suitable sampling frequency for a given flow remains the main issue to solve.

4. SMOOTHING THE ESTIMATED RATE

Recall that our primary objective behind rate estimation is to detect potential incipient congestion on a given flow by examining the run time evolution of its estimated rate. However, as we will see later on in Section 7.2.1 (see Figures 4 and 5), giving a closer look at an estimated rate shows that its corresponding curve is made from a series of data points that are close to each others and which vary slowly at local scales. As a consequence, detecting rate variations at these scales is difficult to achieve. To cope with that, we have looked for a statistical tool that would represent the

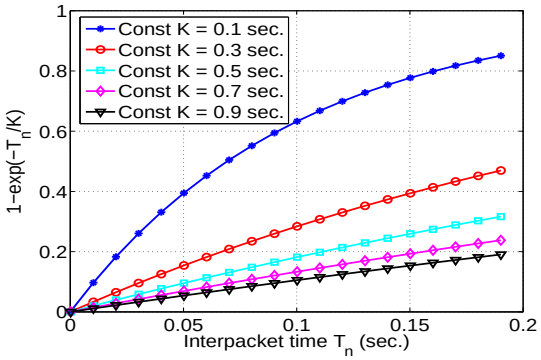

Figure 1: Variation of the filter weight $(1-exp(\frac{-T_n}{K}))$ as a function of the interpacket time T_n, and for different values of the time constant K.

estimated local rate intervals by equivalent points. A way to handle that is to use the kernel smoothing technique.

Kernel smoothing works as follows. Given a set of nearest neighborhood data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ taken over a smoothing window, the aim is to choose an x-coordinate x0 and then compute the fitted point (x_0, y_0). x_0 is usually set to be the middle of the window, while y_0 will represent the *weighted average* of the y-coordinate of the data points [9, Chapter 6]. The weights of the data points are computed according to a *kernel* K_λ which is a continuous and symmetric density function with a scale parameter λ. The kernel function adjusts the weight of each y_i according to the distance $|x_i - x_0|$ which separates its x-coordinate from x_0. For a given scale parameter λ, termed also as the smoothing window size, the weight of a given data point (x_i, y_i) is given by:

$$W_\lambda(x_i) = \frac{K_\lambda(x_0, x_i)}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)}$$

The fit at x_0 writes hence as:

$$y_0 = \sum_{i=1}^{n} W_\lambda(x_i) y_i \quad (2)$$

In its general form, the *kernel* $K_\lambda(x_0, x_i)$ is defined as:

$$K_\lambda(x_0, x_i) = D(\frac{x_i - x_0}{\lambda})$$

where $D(u)$ is a positive real valued density function where its values are generally decreasing for the increasing distances between the x-coordinate of the data points and x_0. There are different forms for the function $D(u)$ [9]. Due to its high smoothing capability, we have adopted the following *Epanechnikov* quadratic kernel given by:

$$D(u) = \begin{cases} \frac{3}{4}(1-u^2) & \text{if } |u| \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Returning back to our rate smoothing case, we have chosen to apply the kernel smoothing technique on the estimated rate samples rather than on the instantaneous rate ones [1]. Actually, applying the kernel smoothing on the less

[1]The instantaneous rate of a given flow at the arrival time of its n^{th} packet at the monitoring router is defined as the ratio of the last received packet size over its interpacket time.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

noisy samples of the estimated rates had resulted in appropriate smoothed curves.

Therefore, to proceed with the kernel smoothing over the estimated rate, we need to specify, in addition to the kernel function, the size of the smoothing window λ. In order to have sufficient number of estimated rate samples for subsequent smoothing, we agreed to set the size of the smoothing window to be at least the expected sum of two RTTs. By assuming that a standard RTT varies around 0.1 seconds, we chose hence to lower bound the smoothing window λ by a value equal to 0.25 seconds. Observe that when the smoothing window is set equal to 0.25 seconds, we will get one smoothed point for each time interval of 0.25 seconds of the estimated rate samples.

Even though in our work we have adopted fixed size of the smoothing windows, note that the size of the smoothing window can be variable, and the general form of the kernel function writes as [9, Chapter 6]:

$$K_\lambda(x_0, x_i) = D(\frac{x_i - x_0}{h_\lambda(x_0)})$$

where h_λ is a width function that determines the width of the smoothing window λ in the neighborhood of a given x_0. Although we did not consider this issue in the current work, we foresee to investigate in this direction in the future time. A potential reference to adapt the window size within the neighborhood of a given x_0 would be to consider the variance of the estimated rate points falling within that window.

5. NON-PARAMETRIC DETECTION OF RATE DEGRADATION

Given that the smoothed rate of a given flow is known at this stage, our objective hence is to detect degradations over the smoothed rate amplitude. These degradations are considered as signs of congestion of the monitored TCP flows. In this section, we introduce a non-parametric detection mechanism which operates over a transformation of the smoothed rate signals to detect potential negative trends (i.e. amplitude decreases). The proposed detection mechanism works as follows.

It starts by computing at run time the slopes of the smoothed rate points. The slope is simply obtained by taking the difference of two successive smoothed points $(Sx_{m-1}, Sy_{m-1}), (Sx_m, Sy_m)$ over the size of the smoothing window. Dividing the latter ratio by the packet size returns a rate in terms of packets per seconds, denoted as $Z(m), m > 1$. Here, m corresponds to the index of the current smoothing window where the slope computation is being done. In the following, we use the term *packet growth rate* to denote the rate $Z(m)$. The latter rate is an indicator of the number of packets that the flow was able to inject or to drain over the last time interval of the smoothing window. For instance, if the packet growth rate $Z(m) = x$ is positive, this indicates that the flow was able to inject x packets/s in the network over the last smoothed time window. On the opposite, if the rate $Z(m) = x$ is negative, this indicates that either the flow was not able to inject any more packets in the network and the network buffers were simply draining, or its rate was severely decreasing since the previous smoothed window. A similar idea to detect congestion of a given flow by investigating the number of its packets that would be buffered in the network over an RTT is presented in [1].

To discover an incipient congestion of a given flow, our proposed detection mechanism will take as input a window of recent samples of its packet growth rate $Z(m)$. To track their variability, it will slide the window by one element at each computation to include the most recent computed growth rate sample. The variability of $Z(m)$ will depend on whether the TCP flow is in the slow start phase or in the congestion avoidance phase, but also, on the level of congestion in the network during each phase.

In fact, based on the analytical model presented in [3], the authors have shown that the rate of a TCP flow that is experiencing congestion will present different growth behavior depending on whether that flow is in the slow start or the congestion avoidance phase. Within the slow start phase, its rate will show a fast linear increase instead of its expected exponential increase. When the flow is in the congestion avoidance phase, its rate will show a sub-linear increase instead of its expected linear increase.

These analytical results have been confirmed by the observations that we got through simulations. Indeed, when the flow is in the congestion avoidance phase, the *average* of its packet growth rate taken over a window will admit a certain steady-state value till either one of the two events occurs. If a congestion occurs, the *average* of its packet growth rate will flip to a lower steady-state value. This transition is what we termed before as a negative trend of the signal $Z(m)$, and it constitutes the key target of our trend detector. If the congestion level diminishes, the *average* of its packet growth rate will flip to a higher steady-state value (a positive trend). On the other hand, when the flow is in the slow start phase, the variability of its packet growth rate was complicated to describe. This is due in one part to the rapid increase of the growth rate during that phase, but also, to the fact that the smoothing windows where the growth rate samples were obtained are not related to the real values of the flow RTTs. Consequently, detecting an incipient congestion during the slow start phase is not achievable by our proposed mechanism. Still, the mechanism is able to detect a congestion upon transitions from slow start phase to congestion avoidance phase.

To proceed with negative trend detection of the signal $Z(m)$, we have used a Kendall-τ trend detector [8]. Our trend detector is similar to the one presented in [17] to detect trends in the measured signal from a smoke sensor with the exception that here we are simply interested in detecting only the negative trends over our signal $Z(m)$.

Given that the packet growth rate is positive for the last obtained smoothed rate sample (Sx_m, Sy_m), the idea is to consider a window of the last M samples of the packet growth rate till index m, and then to compute the normalized average number of difference in amplitude over that window. The latter measure is termed as the trend factor, $\tau(m)$. The difference in amplitude between two samples of the signal $Z(m)$ is computed according to the following sign function $sign(diff)$ defined as follows:

$$sign(diff) = \begin{cases} 1 & diff > 1 \\ 0 & -1 \leq diff \leq 1 \\ -1 & diff < -1 \end{cases} \quad (4)$$

where $diff = Z(i) - Z(j)$ represents the difference between two samples of the packet growth rate. Equation (4) translates that when the difference between two samples of the

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

packet growth rate is less than one packet, then the detector considers that there was no significant difference in amplitude, and hence it is not an indication of a change due to congestion. On the other hand, if the difference exceeds the threshold of one packet, the sign function takes the value of $+1$ to indicate a positive increase while it takes the value of -1 when the difference goes down below the threshold of one packet. The trend detector is hence formulated as

$$T(m) = \sum_{i=0}^{M-2} \sum_{j=i+1}^{M-1} sign(Z(m-i) - Z(m-j)) \qquad (5)$$

where $Z(k)$ is simply the packet growth rate at the smoothed rate sample with coordinate (Sx_k, Sy_k). Here, $Z(m)$ represents the last obtained packet growth rate sample over the current detection window of size M. The trend factor $\tau(m)$ is simply the normalized value of trend detector obtained by taking the ratio of $T(m)$ over the total possible number of sign changes over a window of length M:

$$\tau(m) = \frac{2T(m)}{M(M-1)} \qquad (6)$$

Subsequently, the values of $\tau(m)$ will vary within the range $[-1, +1]$. For our detection mechanism, we consider that a negative trend occurs whenever the trend factor reaches a value lower than -0.5. In other words, for a given detection window, when the number of negative differences in amplitude is more than the half of the possible number of sign changes, there is a high potential that the signal is undergoing a decrease as a consequence of an incipient congestion. The choice of this value as a congestion indicator has been confirmed throughout the various simulations that we have conducted (please refer to Section 7.2.3).

6. NS-2 IMPLEMENTATION OF RATE ESTIMATION AND RATE DEGRADATION

In this section, we describe how to add observation points in ns-2 source code in order to perform the rate estimation and the rate degradation detection. The choice of implementing these mechanisms in the source code of ns-2 was motivated mainly by the need to gather several relevant information which we were not able to collect by using TCL commands. In addition, implementing these mechanisms in the source code of ns-2 had resulted in relatively small execution times.

Both mechanisms for rate estimation as well as for rate degradation have been implemented in the function *QueueMonitor::in* which is defined in the file *./ns-version/tools/queuemonitor.cc*. This function will be called upon the arrival of a packet on an monitoring router interface for which a *queue monitor* or a *flow monitor* command was already defined in the TCL script. Therefore, upon a packet arrival, the monitoring router identifies first its flow Id, and then, it proceeds to estimate and smooth the rate of the flow. A special attention needs to be taken if more than one *queue monitor* or *flow monitor* commands were to be defined in the TCL script. In this case, the function *QueueMonitor::in* will be called upon a packet arrival at each interface for which we had set a monitor object. To cope with that, one needs to use an additional command to pass from the TCL script the identity of the queue object representing its monitoring router.

Note that in order to identify the identity of the current queue monitor object at run time, we made use of a function called *name()*, defined in ./tclcl/tracedvar.h, which returns a pointer to the current queue-monitor object when it is used with the command *this*.

Another technical problem we dealt with was related to the acquisition of the current congestion window of a given TCP source upon the reception of one of its packet. This information is needed simply to compute, upon a packet arrival, the real rate of a given TCP source for later comparison to its estimated rate. On the opposite to TCL code, obtaining the congestion window of a TCP source within the function *QueueMonitor::in* was not evident. The way we handled that was to introduce a new variable in the TCP header of the packets which stores the last congestion window seen by the TCP source. The definition of this variable was implemented in the function *TcpAgent::output* defined in the file *./ns-version/tcp/tcp.cc*. Note that the last value of RTT seen by a TCP source can be obtained within the function *QueueMonitor::in* from an existing variable already defined in the TCP header of the packet.

7. SIMULATION RESULTS AND DISCUSSIONS

In this section, we introduce the two simulation scenarios that we have considered, in particular, a single-bottleneck dumb-bell topology and a single-bottleneck parking-lot topology. Next, we present the simulation results that we have conducted in ns-2 to evaluate the various mechanisms proposed in Sections 3, 4 and 5.

7.1 Simulation scenarios

To set up a network scenario, one needs first to define a network topology, a traffic model and the corresponding performance metrics of interest. Even though that recent efforts have been done to provide a standardized test suite to evaluate the various proposals of TCP [16, 5, 6, 4], such a suite does not exist for the moment. Therefore, for the present simulation scenarios, we have considered two commonly studied network topologies, namely a single-bottleneck dumb-bell topology (designed henceforth as scenario A) and a single-bottleneck parking-lot topology (designed as scenario B) [16]. Regarding the traffic model, we have considered long-lived FTP connections that start at random or deterministic times and last for the simulation duration.

7.1.1 Scenario A: single-bottleneck dumb-bell topology

For Scenario A, we consider three links with equal capacity of 100 Mbps and 1 ms delay. The bottleneck link has a capacity of 50 Mbps and 20 ms delay. There are three FTP sources located at $S1$, $S2$ and $S3$ that start their transmissions randomly within the interval $[0, 5]$ seconds and then stop simultaneously at time 100 seconds. The buffer size of node $R1$ is limited to 100 packets while those of nodes $S1$, $S2$ and $S3$ are limited to 50 packets. Packet size is set to 1 KBytes. Table 1 provides a summary of the scenario parameters and the corresponding network topology is shown in Figure 2.

7.1.2 Scenario B: single-bottleneck parking-lot topology

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

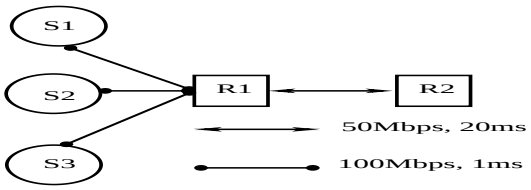

Figure 2: Scenario A: single-bottleneck dumb-bell topology.

Table 1: Simulation parameters for scenario A.

TCP variant	NewReno
TCP window size	8000
Number of connections	3
Simulation duration	100 seconds
Queue management	Drop tail
Packet size	1000 Bytes
Filter time constant	0.4
Queue monitoring	$R1 - R2$
Buffer size R1-R2	100 Packets
Buffer size Si-R1	50 Packets

Scenario B consists of four TCP sources, noted as Si which start transmitting at random or at deterministic times to their respective destinations Di. The TCP flow originating from node $S1$ to node $D1$ is going through several links, including a bottleneck link and it is crossed by the other TCP flows that share some of these links. For the current work, we are considering that all the sources start transmitting simultaneously at time 0. There is a single bottleneck link between nodes $R2$ and $R3$ with a capacity of 50 Mbps and 20 ms delay. Links capacity (resp. delay) between nodes $R1$ and $R2$, and between nodes $R3$ and $R4$ is set to 100 Mbps (resp. 20 ms delay). Access links capacity (resp. delay) between the various nodes and the routers is set to 70 Mbps (resp. 5 ms delay). Packet size is set to 1.5 KBytes. Table 2 provides a summary of the scenario parameters and the network topology is shown in Figure 3.

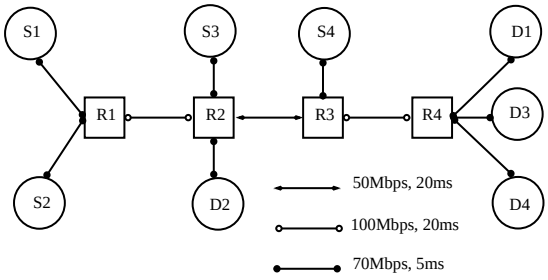

Figure 3: Scenario B: single-bottleneck parking-lot topology.

7.2 Simulation results

7.2.1 Flow rate estimation

Table 2: Simulation parameters for scenario B.

TCP variant	NewReno
TCP window size	512
Number of connections	4
Simulation duration	50 seconds
Queue management	Drop tail
Packet size	1500 Bytes
Filter time constant	0.4
Queue monitoring	$R3 - R4/R1 - R2$
Buffers size Ri	200 Packets
Buffers size Si, Di	50 Packets

Recall that our main objective of rate estimation is to come up with an estimator that is accurate enough in terms of tracking the variability of the real rates of TCP sources for later detection of amplitude decrease.

Figure 4 compares the real rate of TCP source $S1$ to the estimated rate under scenario A. The real rate, used simply for comparison purpose, is plotted by taking the ratio of the current congestion window size of the TCP source (stored in the TCP header of the packet - refer to Section 6 to see how to implement it in ns-2 source code) over its current RTT. The estimated rate is obtained by setting the estimator according to Equation (1) with a time constant K set equal to 0.4. The monitor router is placed at the entry of link $R1 - R2$ (see Figure 2). The x-axis on the figure corresponds to packet arrival times at the entry of router $R1$. As the figure illustrates, the estimated rate follows well the *trend* of the real rate even on chunks where the real rate shows high variability. However, the estimator fails to track the real rate when the flow is in the slow start phase (beginning of the simulation), mainly due to its low reactiveness with respect to the prompt increase of the real rate during that phase.

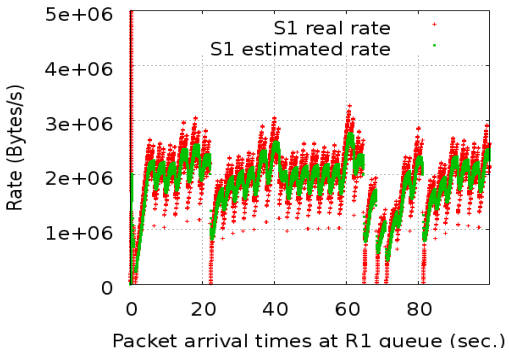

Figure 4: Scenario A: rate estimation for single-bottleneck dumb-bell topology.

Similar results are obtained under scenario B. Figure 5 compares the real rates of flows $S1 - D1$, $S3 - D3$, and $S4 - D4$ to their estimated rates. The monitor router is placed at the entry of link $R3 - R4$, downstream of the bottleneck link (see Figure 3). The x-axis on the figure corresponds to the packet arrival times at $R3$. As the curves

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

illustrate, the estimated rates follow well the *global trend* of the real rates over large time scales. However, alike scenario A, the estimator fails to track the evolution of real rates over some parts when a flow is in its slow start phase. Over these chunks, the estimator underestimates the real rates due to its relative low reactiveness in following the prompt variabilities of the real rates. Even though, the estimator performs sufficiently well in tracking the global trends of the various rates.

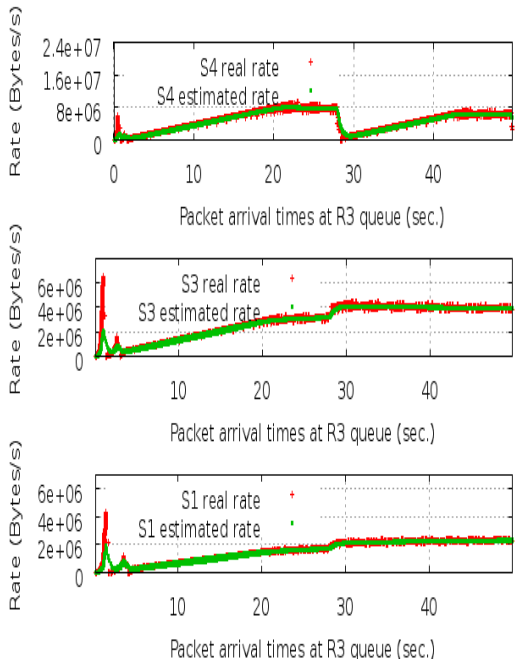

Figure 5: Scenario B: rates estimation for parking-lot topology.

As a side remark, observe that placing the monitor router downstream or upstream of the bottleneck link has no impact on the performance of rate estimation. Indeed, we got similar estimate for flow $S1 - D1$ when the monitor router was located upstream of the bottleneck link at the entry of link $R1 - R2$.

We have also simulated scenario B in the presence of cross-traffic short lived TCP connections. The considered scenario is a follows. In addition to the elephant connection that lasts till the simulation ending time, at each source S_i, a new TCP connection arrives according to a Poisson process with a rate of $\frac{1}{0.05}$ connection/seconds. Each one of these short connections generates random data size according to a Pareto distribution with a mean of $10KBytes$ and a shape of 1.5. As a result, the rates of some of the elephant flows have changed due to the occurrence of different congestion levels over the various buffers. However, the performance of the rate estimator did not altered in the presence of short connections.

7.2.2 Smoothing the estimated rate

Recall from Section 4 that in order to apply the kernel smoothing on the estimated rate samples, we need to specify a kernel function and to set a size of the smoothing window. As we mentioned already, in this work we have chosen to use the *Epanechnikov* quadratic kernel given by Equation (3). Smoothing windows are of fixed size and independent from each others. Their size is set equal to 0.25 seconds. In other terms, the kernel function will generate one smoothed rate point for each set of estimated rate samples taken over a time interval of 0.25 seconds. Note that the chosen value of smoothing window size is not unique and other values can be taken instead. Nevertheless, the lower bound of these values should be at least the sum of two RTTs of the considered flow in order to have sufficient samples of the estimated rate. As for their upper bound, a value corresponding to the sum of four RTTs would be sufficient in order to maintain a good smoothing representation of the estimated rate. A rough estimation of RTT can be taken to be equal to one of the largest interpacket time seen by the monitoring flow when it starts monitoring the flow.

Once the size of the smoothing window is specified, the kernel smoothing method proceeds with the first window, determines the x-coordinate of the target point x_0 which is set equal to be the middle of the window, from which it computes its fit y_0 using Equation (2). The computation of the fit of a given window is carried on line following a new estimation of the rate corresponding to the arrival of a new packet. When a new packet arrival falls outside the current window, the method proceeds to the next smoothing window, repeats the computations again, and continues so on over the estimated samples of the flow.

Figure 6 (resp. Figure 7) plots the smoothed rate (shown in blue stars) over a chunk of the estimated rate of flow 1, scenario A (resp. scenario B). As both figures illustrate, the points obtained with the kernel smoothing represent well the various intervals of the estimated rate making hence easier the detection of change over them (see next subsection).

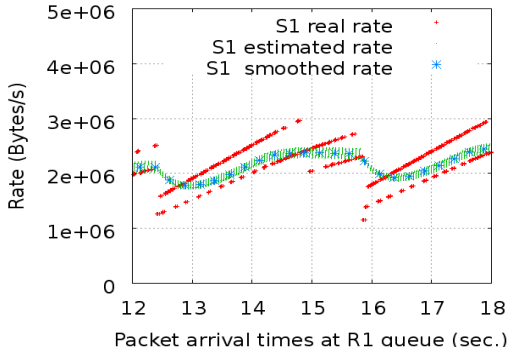

Figure 6: Kernel smoothing of the estimated rate of flow 1, scenario A, over the time interval $[12, 18]$ seconds.

7.2.3 Non-parametric detection of rate degradation

To apply the non-parametric detection mechanism, the algorithm starts by setting the size of the detection window. The computation of the trend factor over a detection window will be carried whenever that window is filled with M samples of packet growth rate $Z(m)$. By that time, and following the computation of the trend factor, the mecha-

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

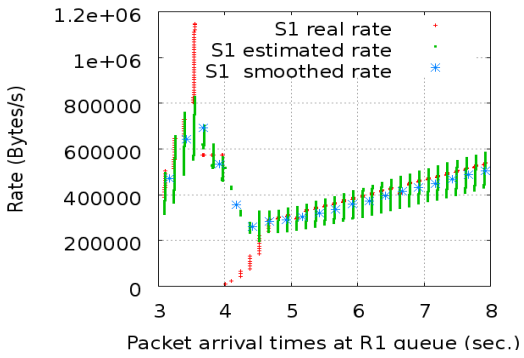

Figure 7: Kernel smoothing of the estimated rate of flow 1, scenario B, over the time interval $[3, 8]$ seconds.

nism decides on whether the monitored flow is suffering a congestion or not. Then, the detection window is slided one element to include the next sample of the packet growth rate as long as its value is positive. When that value is negative, we assume that the flow is draining. Consequently, we stop the computation of the trend factor, and we reset all the elements of the current detection window to zero. Filling the detection window will restart upon the acquisition of a positive sample of packet growth rate.

Observe that the size of the detection window constitutes a trade-off between fast change detection versus false alarm of change detection. In particular, when the detection window size is small, for instance in the order of four samples of $Z(m)$, there is a high potential of false change detection. On the opposite, when the detection window is large, for instance more than twelve samples, there is a high potential of not detecting the change due to the attenuation of the impact of last samples. Moreover, when the window size is large, the detection time will be consequently longer.

Throughout the conducted simulations, we have observed that a detection window with a size varying between eight to twelve samples of packet growth rate constitutes a good trade-off to compute the trend factor. For smoothing windows with size 0.25 seconds, this corresponds to a minimum detection time varying between 2.25 to 3.25 seconds (e.g. the method needs nine smoothed rate samples, and hence nine smoothing window intervals, to complete a detection window with eight growth rate samples). Observe that the form of trend detector that we considered in Equation (5) gives more weights to the last included samples of the window over the previous ones, which results consequently in a high reactiveness to change and minimizes hence the detection time. In our simulations, we have set the size of the detection window equal to eight.

Even though during our simulations we had not to deal with false alarms of change detection, note as a side remark that decreasing the probability of having such events can be achieved by various ways. For instance, the method would compare the trend factors obtained over two successive detection windows and look whether they confirm the negative trend of $Z(m)$. Another approach would be to lengthen the detection window size upon a potential change detection to include the next sample of the packet growth rate within the computation of the trend factor.

Figure 8 plots the growth rate of flow 1 under scenario A. As the estimation of its rate shows (refer to Figure 4), the flow is suffering from a continuous congestion during its slow start and congestion avoidance phases. The congestion is mainly due to buffer overload at router $R1$. As a result, the growth rate of that flow is highly variable with the appearance of a continuous and high number of negative trends. This is shown clearly on Figure 9 which plots a closer view of the growth rate of that flow over a shorter time interval $[25, 40]$ seconds. Figure 10 plots the corresponding trend factors obtained by applying our proposed detection mechanism on the growth rate of Figure 8. As the figure illustrates, the observed negative trends have been detected sufficiently well by our trend detection mechanism with a detection window of eight (e.g. detecting three out of five negative trends over the interval $[25, 40]$ seconds, see Figure 11). Note that the detection times correspond to the instants when the trend factor takes a value less than -0.5.

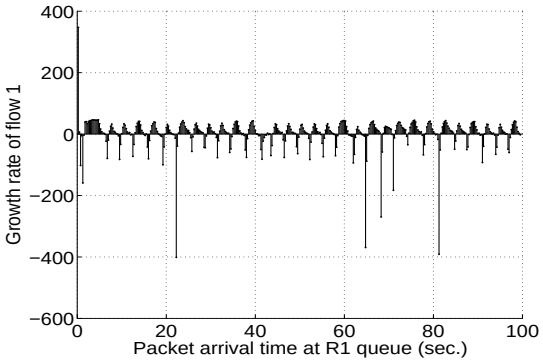

Figure 8: Packet growth rate $Z(m)$ of flow 1, scenario A, over the simulation time.

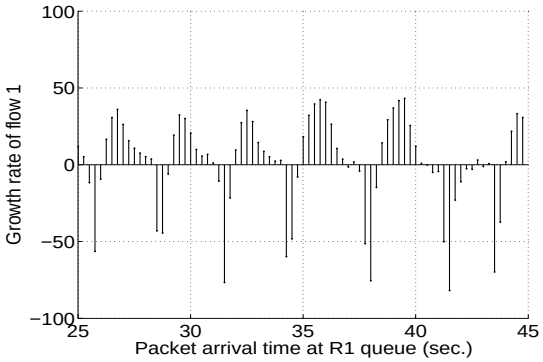

Figure 9: A view of the packet growth rate $Z(m)$ of flow 1, scenario A, over the time interval $[25, 45]$ seconds.

For the case of scenario B, things are a little bit simpler. For instance, considering flow 1 in that scenario (bottom curve on Figure 5), we observe that the flow suffers during its slow start phases from two severe drops. Following that, the flow enters in the congestion avoidance phase and its rate will increase linearly at a constant rate till instant 22 seconds. By that instant, a congestion occurs, and its

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

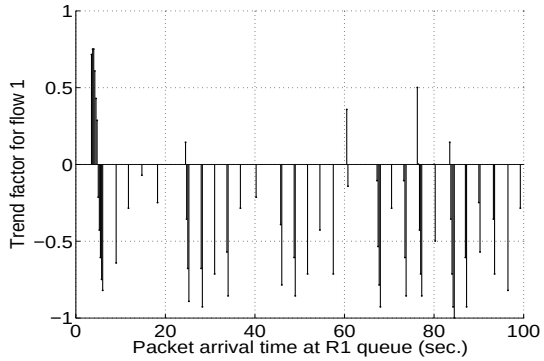

Figure 10: Trend factor computed over the growth rate samples of flow 1, scenario A.

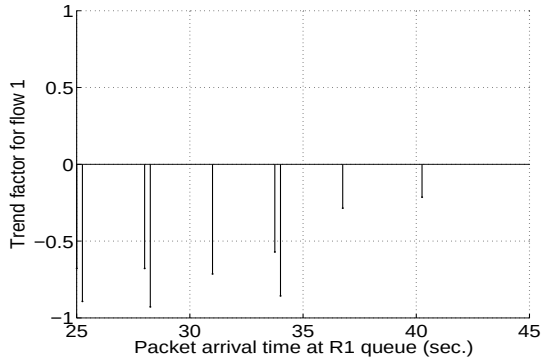

Figure 11: Trend factor computed over the growth rate samples of flow 1, scenario A, over the time interval $[25, 45]$ **seconds.**

rate starts to increase sub-linearly till instant 28 seconds where the congestion vanishes resulting consequently in an increase of its rate. However, the eclipse of the congestion is short and a congestion reoccurs around instant 30 seconds, resulting again in a decrease of its rate. Figure 12 plots the growth rate of flow 1, scenario B, while Figure 13 replots a closer view of the growth rate of that flow over the time interval $[10, 35]$ seconds.

The appearance of the two negative trends during the congestion avoidance phase (around instants 22 and 30 seconds) has been discovered rapidly well by our trend detector. This is illustrated on Figure 14 which draws the trend factors corresponding to the growth rate of flow 1 over the interval $[10, 35]$ seconds. As it is shown, the trend factor will have a value less than -0.5 around instants 22 and 30 seconds.

8. CONCLUSIONS

In this paper, we proposed and implemented in ns-2 a mechanism that detects incipient congestion on elephant TCP flows by estimating and tracking their rates. The proposed mechanism is based on measurements collected passively by a monitoring node which is located on the intermediate paths between TCP source-destination pairs. Given a particular flow, its rate estimation requires measuring its interpacket times as well as the size of its received packets.

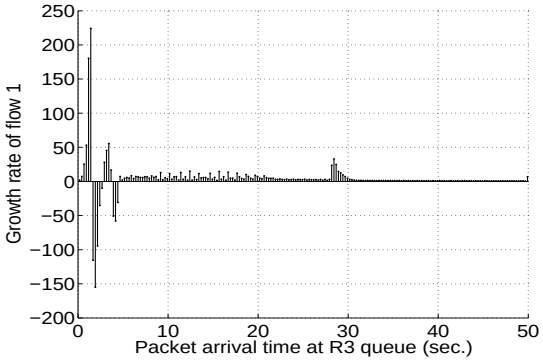

Figure 12: Packet growth rate $Z(m)$ **of flow 1, scenario B.**

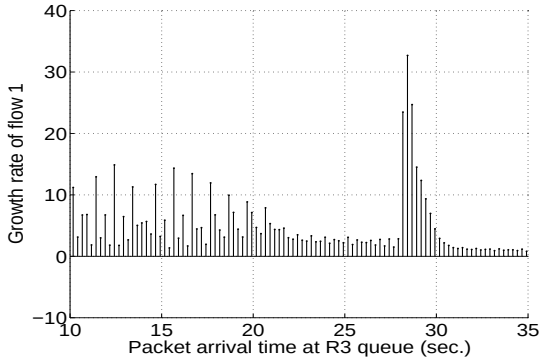

Figure 13: A view of the packet growth rate $Z(m)$ **of flow 1, scenario B, over the time interval** $[10, 35]$ **seconds.**

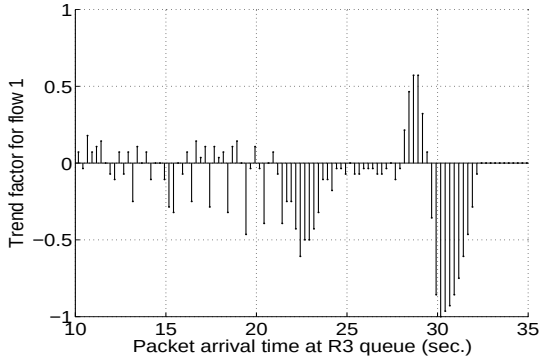

Figure 14: Trend factor computed over the growth rate samples of flow 1, scenario B, over the time interval $[10, 35]$ **seconds.**

Congestion detection operates consequently on the smoothed rate samples by using a sliding window and a trend detector.

To implement the mechanism, we had to develop various tools that are useful for estimating and detecting the congestion when one analyzes real traces as well as in the analysis of simulation traces. These tools were programmed in ns-2 so as to get real time results during the simulations. Our

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639

goal is to simulate a real behavior of a network in which a remote detection mechanism would allow a monitoring router to take measure against the congestion in real time. The tools that we developed are easy to use and to transfer to other applications.

While our proposed method operated very well in detecting incipient congestion on a large number of simulated flows under different scenarios, our work is sill in progress, and the method has some improvements to be done. Our future work will focus on estimating and smoothing the rate by using samples of interpacket times instead of gathering these information upon each packet arrival. We expect that a digital-signal-processing implementation, with very relaxed sampling times per flow, is possible and can be useful to manage the dominant flows in a network under congestion. Another secondary concerns would be to propose mechanisms to adapt dynamically the size of the smoothing and the detection windows.

Acknowledgements

The authors would like to thank Paulo Goncalves from INRIA ENS-Lyon, France, and Julio Rojas from the university of Avignon, France, for fruitful discussions on the paper.

This work was done in the framework of the INRIA and Alcatel-Lucent Bell Labs Joint Research Lab on Semantic Networks.

9. REFERENCES

[1] C. Albuquerque, T. Suda, and B. Vickers. Network border patrol. In *Proc. of IEEE Infocom*, Tel Aviv, Israel, 2000.

[2] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. RFC 2581, April 1999.

[3] E. Altman, F. Boccara, J. Bolot, P. Nain, P. Brown, D. Collange, and C. Fenzy. Analysis of the TCP/IP flow control mechanism in high-speed wide-area networks. In *Proc. of the 34th IEEE Conference on Decision and Control*, New Orleans, Louisiana, USA, December 1995.

[4] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, and I. Rhee. Towards a common tcp evaluation suite. In *Proc. of PFLDnet*, March 2008.

[5] L. L. H. Andrew, S. Floyd, and W. Gang. Common TCP evaluation suite, July 2008.

[6] S. Floyd and E. Kohler. Tools for the evaluation of simulation and testbed scenarios. Internet Draft, http://tools.ietf.org/html/draft-irtf-tmrg-tools-05, February 2008.

[7] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE Network*, 17:6–16, 2003.

[8] J. D. Gibson and J. L. Melsa. *Introduction to nonparametric detection with applications*. Academic Press, 1975.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning - Data Mining, Inference and Prediction*. Springer, 2001.

[10] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. of Infocom*, 2004.

[11] H. Jiang and C. Dovrolis. Passive estimation of tcp round-trip times. *Sigcomm Computer Communication Review*, 32(3):75–88, 2002.

[12] D. Katabi and C. Blake. Inferring congestion sharing and path characteristics from packet interarrival times. Technical Report MIT-LCS-TR828, 2001.

[13] R. Lance. *Network state estimation via passive traffic monitoring*. PhD thesis, University of Maryland, College Park, 2005.

[14] R. Lance and I. Frommer. Round-trip time inference via passive monitoring. *Sigmetrics Performance Evaluation Review*, 33(3):32–38, 2005.

[15] I. Stoica, S. Shenker, and H. Zhang. *Core*-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Transactions on Networking*, 11(1), February 2003.

[16] G. Wang, Y. Xia, and D. Harrison. An ns2 tcp evaluation tool. Internet Draft, http://tools.ietf.org/html/draft-irtf-tmrg-ns2-tcp-tool-00, April
2007.

[17] S. Wang and Z. Dou. Exclusive trend detector with variable observation windows for signal detection. *Electronics Letters*, 33(17):1433–1435, August 1997.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7639
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7639