Estimating packet loss rate in the access through application-level measurements

Estimating Packet Loss Rate in the Access Through Application-Level Measurements

Simone Basso
Nexa Center for Internet & Society
Dept. of Control and Computer Engineering
Politecnico di Torino, Turin, Italy
simone.basso@polito.it

Antonio Servetti
Dept. of Control and Computer Engineering
Politecnico di Torino, Turin, Italy
antonio.servetti@polito.it

Michela Meo
Dept. of Electronics and Telecommunications
Politecnico di Torino, Turin, Italy
michela.meo@polito.it

Juan Carlos De Martin
Nexa Center for Internet & Society
Dept. of Control and Computer Engineering
Politecnico di Torino, Turin, Italy
juancarlos.demartin@polito.it

ABSTRACT

End user monitoring of quality of experience is one of the necessary steps to achieve an effective and winning control over network neutrality. The involvement of the end user, however, requires the development of light and user-friendly tools that can be easily run at the application level with limited effort and network resources usage. In this paper, we propose a simple model to estimate packet loss rate perceived by a connection, by round trip time and TCP goodput samples collected at the application level. The model is derived from the well-known Mathis equation, which predicts the bandwidth of a steady-state TCP connection under random losses and delayed ACKs and it is evaluated in a testbed environment under a wide range of different conditions. Experiments are also run on real access networks. We plan to use the model to analyze the results collected by the "network neutrality bot" (Neubot), a research tool that performs application-level network-performance measurements. However, the methodology is easily portable and can be interesting for basically any user application that performs large downloads or uploads and requires to estimate access network quality and its variations.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operation—*Network Monitoring*; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement, Performance

Keywords

TCP, Network neutrality, Mathis model, Packet loss rate estimation

W-MUST'12, August 17, 2012, Helsinki, Finland.

1. INTRODUCTION

The debate on network neutrality is becoming more and more relevant in economic, technical and political environments. As a result, a number of active and passive tools have been developed to assess network neutrality and make access networks more transparent. Nowadays, many of the active tools that need, for performance reasons, to be as close as possible to end users are available through Measurement Lab (M-Lab) [5], a distributed world-scale server platform, which hosts the server side component of the tools, facilitates server discovery and data collection[1]. Still, a common problem, for active and passive tools, is that many of them perform only spot measurements, and large scale continuous monitoring is instead needed to better inform the debate on network transparency, characterization of ISP behavior and network neutrality [23] [16].

In turn, the necessity of large scale and continuous monitoring fosters the implementation of measurement tools at application level, using simple and nonintrusive methodologies, to incentive Internet users to install them on their personal computers. Typically, application level tools run TCP transfers, eventually emulating different protocols, and measure transfer speed and other application level metrics. However, to get clear clues into network neutrality and make monitoring effective, the implications of traffic discrimination onto TCP connections need to be investigated and then exploited.

The starting point to study these implications is the observation that, in a nonneutral network, packets belonging to discriminated TCP flows are more likely to be delayed or dropped. The application level effect of these discriminating policies is that the user sees a reduced *goodput*, which is the application level speed measured at the receiver. More specifically: (i) a multiplicative increase of delay yields to a multiplicative decrease of speed, because TCP goodput is inversely proportional to the delay; (ii) a lost packet will at least cause TCP to halve its sending rate, with nearly instantaneous effects on goodput.

However, since goodput depends on both delay and losses, a

[1]M-Lab is a joint initiative of New America Foundation's Open Technology Institute, Google, and academic researchers. Its 70+ servers are either installed at Internet Exchange Points, or hosted by academic institutions and partner companies. In addition to hosting, M-Lab also provides support services, such as automatic data collection and publishing. Hosted measurement tools include NDT [7], Glasnost [3], NPAD [10], as well as Neubot [8], the tool we developed and maintain.

methodology that aims not only at detecting discrimination but also at understanding the type of discrimination needs to estimate the key parameters that control goodput: the *round trip time* (RTT), and the *packet loss rate* (PLR). Once average RTT and PLR have been estimated for different protocols, statistical analysis can be used to decide whether there is a significant difference in the RTT and PLR typically experienced by the protocols.

In this paper, we propose a model to estimate PLR of a network path by measuring RTT and goodput during a 10-second TCP bulk transfer, and we compare the model results with TCP analysis of packet traces captured during the transfer. The 10-second requirement is imposed to the model to account for the fact that most application level tools run their tests for a fixed number of seconds, typically 10, to achieve their scalability, continuous monitoring and low-overhead goals.

Among all the possible uses of the model, we are particularly interested in using it to postprocess and analyze the results of the "network neutrality bot" (Neubot), the tool that we developed and maintain [8]. This tool, already described in our previous works [18] [12] [14] [13], is an open source daemon that runs in the background and periodically performs HTTP and BitTorrent performance tests with instances of itself, installed on the servers provided by M-Lab. During the tests, the tool measures goodput and RTT at application level, and stores the results on M-Lab servers[2].

The model presented in this paper represents our effort to shift Neubot test results analysis from goodput alone (see for example our qualitative analysis of results in the Turin area [13]) to, as said by Bauer et al., "a more nuanced characterization [...] of quality" [15] that includes also RTT and estimated PLR. This new characterization brings practical advantages both for access network quality and protocol discrimination studies, because the cause of different performance (different goodput) is made explicit by differences in delay and estimated loss rate. This information provides insights on how the Internet access service can be improved and/or on the cause of protocol discrimination, allowing researchers to prepare and deploy, as said by Palfrey and Zittrain, "specially tailored" tests [23].

The remainder of this paper is organized as follows. In section 2 we derive our model from the well-known Mathis equation, and we discuss the model strengths, weaknesses, and application level requirements. In section 3, we evaluate the model, both in a testbed environment, to characterize its behavior in a wide range of network conditions, and in controlled Internet experiments, where we add random losses to an existing connection and we check whether the model can detect that. Finally, in section 4 we draw the conclusions, and we report about current and future research efforts.

2. MODEL

In this section we describe our model and discuss its strengths, weaknesses and limitations.

2.1 Model description

The model for packet loss rate estimation at the application level presented in this paper is based on the well-known Mathis model, which is one of the simplest renewal-theory-based TCP model. The Mathis model describes the macroscopic behavior of the congestion avoidance algorithm and provides a simple formula to predict the goodput of a sustained steady-state TCP connection [20]. More complex and advanced models exist, e.g.: PFTK, which, unlike the Mathis model, also models timeouts and window limitation [22]; Cardwell et al., which extends PFTK to model TCP latency and the effects of slow start [17]; Guillemin et al., which also models congestion avoidance in presence of losses that span more than one RTT [19]. However, for this first study we decided to use the Mathis formula, because its parameters (goodput and RTT) are easily measurable at application level. Leaving the use of more complex models for future investigations, if needed.

Because of its simplicity, Mathis formula builds on a number of assumptions, and here we report only the ones that are more relevant to our model:

1. SACKs are implemented, so that multiple losses in a RTT are indication of a single congestion event;
2. the receiver window is big enough and the sender has always data to send;
3. the connection never times-out and always recovers from a loss using fast retransmit and fast recovery;
4. the connection is long enough to reach steady state;
5. the RTT is constant over the path.

Note that, if assumption 1-3 are not met, the result is a reduced goodput, while, if assumption 4 is not met, the result may be a higher goodput. This happens when slow start overestimates the available capacity, and congestion avoidance starts from an operating point far from the actual equilibrium.

Under these assumptions the Mathis formula is:

$$goodput = \frac{MSS \cdot C}{RTT \cdot \sqrt{PLR}}, \tag{1}$$

where:

(a) *goodput* is given by the size of delivered data over the delivery completion time;

(b) *MSS* is the maximum segment size, that is typically 1460 bytes per packet (even if other sizes are possible);

(c) C is a constant that incorporates the loss model and the acknowledgment strategy. When the loss model is random and TCP uses delayed ACKs, the value of this constant is 0.93;

(d) RTT is the round trip time;

(e) *PLR* is the number of congestion signals per acknowledged packet.

In our model we assume $MSS = 1460, C = 0.93$, and we derive the packet loss rate (PLR) from (1) to obtain:

$$PLR = \left(\frac{MSS \cdot C}{goodput \cdot RTT} \right)^2 \tag{2}$$

Using (2), PLR can be estimated by a simple measure of goodput and RTT.

In the next subsection we discuss how to compute goodput and RTT at the application-level and what are the limits of the model.

[2]Test results are automatically published on Neubot and M-Lab websites [9] [2], under the terms and conditions of Creative Commons Zero license, so anyone is free to reuse them for for research purposes.

2.2 Implementation requirements

The proposed model is particularly interesting because it can be easily implemented at the application level. In fact, it only requires the measures of connection goodput and RTT, which are both easy to estimate with a long-enough TCP connection, especially when both endpoints are under control, as happens for Neubot and other measurement tools. However, it should be possible to use this model also when just one endpoint is under control: for example, one can employ a modified *wget* [4] or *curl* [1] that estimates the PLR of an arbitrarily long download every 10 second.

Goodput can be estimated at the receiver by dividing the number of received bytes by the data transfer completion time. It should not be computed at the sender, because the sender might underestimate the completion time. In particular, the measure of the completion time at the sender may not take into account the time required for emptying the pipe and delivering all the data on the fly.

RTT can be estimated by measuring the time required for the *connect()* to complete, which is an upper bound of the three-way handshake time. A more robust strategy, which operates during the connection life time, consists in sending small requests that elicit immediate small responses and measuring the elapsed time between the request and the response. This is similar to what TCP does, when evaluates the time elapsed from a packet transmission and the reception of the associated ACK. However, whether this strategy can be implemented or not depends on the design of the application and on the specification of the application protocol.

2.3 Model discussion

The model limits derive from the assumptions implied in the Mathis formula, listed in Section 2.1. First, the model assumes a constant RTT and it trusts a RTT measurement performed before the actual transmission (or after it). So, the model is less accurate in networks with high RTT variations, such as wireless networks. Moreover, if the *SYN* or *SYN|ACK* segment is lost, *connect()* time is not a valid estimation of RTT, because it incorporates at least one retransmission timeout, as Mellia and Zhang's model for short lived connections shows [21].

Second, the model assumes random losses and fast recovery, but typically losses are correlated on the Internet. In particular, when the bottleneck implements drop tail queue management, a burst of packets (or part of it) can be lost, and the sender may receive less than three duplicate ACKs in the next RTT, as documented by PFTK model [22]. When this happens, the sender cannot trigger fast recovery and stops until the transmission timeout expires. In turn, this reduces the goodput and bumps the estimated PLR.

Third, the model assumes that the connection is long enough to reach the equilibrium. When this is not true, the estimated PLR may be lower than the real PLR. In particular, it is not the absolute elapsed time that matters, but, rather, the number of *rounds* (periods of duration equal to the RTT that correspond, in congestion avoidance, to the time needed to increase the window by one segment) experienced by the connection. When the PLR is small, slow start may overestimate the available bandwidth, and a short connection does not run for enough rounds to average that effect. Similarly, when the PLR is high, the connection may not run for enough rounds to average the effect of one (or few) lucky lossless rounds.

Fourth, the model assumes the receiver window is big enough and does not limit the transfer rate. When this assumption is not met, the model does not estimate the real PLR, but rather the "virtual" PLR imposed by the receiver buffer. This can be easily detected at the packet trace level, checking whether the connection experienced any loss event, while online detection at application level seems more problematic. Still, educated guesses are possible, since the behavior of major operating systems with respect to automatic receiver buffer tuning is known, and it is straightforward to query for operating system name and version at the application level.

Fifth, the model assumes that the sender has always data to send. In other words, this methodology can be used to estimate the PLR experienced by continuous single-connection TCP data transfers only. Whether this is possible or not, is something that clearly depends on the design of the application and on the specification of the application protocol.

Thus, while application-level measurements are very appealing and simpler both to implement and to run than low level ones, a number of factors need to be taken in account in the data analysis process to ensure a reliable interpretation of the collected data. Then, in the remainder of this paper we evaluate the model and analyze the measurements taking care of the remarks presented in this section.

3. MODEL EVALUATION

The model is evaluated using a testbed environment as well as controlled Internet experiments. The testbed environment is used to evaluate the model in a wide range of network conditions: *netem* is used to emulate paths with different RTTs and (random) PLRs[3]. Controlled experiments are performed to check whether the model can detect the injection of extra (random) losses into a real Internet path. In this case too, losses were generated using *netem*.

Both in testbed and in controlled experiments, goodput and RTT samples are collected at the application level, during a TCP bulk transfer test. RTT is estimated by measuring the time *connect()* takes to complete[4]. Goodput is computed by dividing the amount of bytes received over the elapsed time. The test duration is controlled by the sender, who stops sending after 10 seconds and waits for the receiver to close the connection[5]. We have collected 16 samples for each different network condition, i.e. for each (RTT, PLR) couple.

3.1 Evaluation using testbed

The model was evaluated using the testbed that emulates a wide range of network conditions. The testbed was composed by three Linux 3.0.0 virtual machines, running on VirtualBox and connected using internal network emulation. The sender and the receiver machines were attached to two different virtual networks. In turn, the two networks were connected by the third machine, which routed traffic and performed network emulation using *netem*. Overall, when *netem* did not kick in, the setup could sustain a TCP bulk transfer rate of more than 300 Mbit/s, and the RTT overhead was negligible.

Figure 1 shows the measured goodput as a function of the imposed RTT. Each point is the result of a single transfer test for different *netem*-imposed RTT and PLR conditions. RTT ranges from a minimum of 0.02 s to a maximum of 0.3 s, while PLR ranges from 1e-5 to 0.03. Points with different PLRs are given different levels of gray, to show more clearly that the higher the PLR is, the

[3]Netem [6] is a Linux kernel module that allows to emulate characteristics of wide area networks (WAN), by adding losses and/or delays to network interfaces. The basic usage allows to add a fixed delay and a random loss probability, but more advanced configurations are possible, allowing to specify delay distribution and loss correlation. Other emulated WAN properties include packet reordering, duplication and corruption.

[4]This is what Neubot does, for simplicity.

[5]The test runs for 10 seconds, regardless the path characteristics, to model the behavior of Neubot and other application level tools.

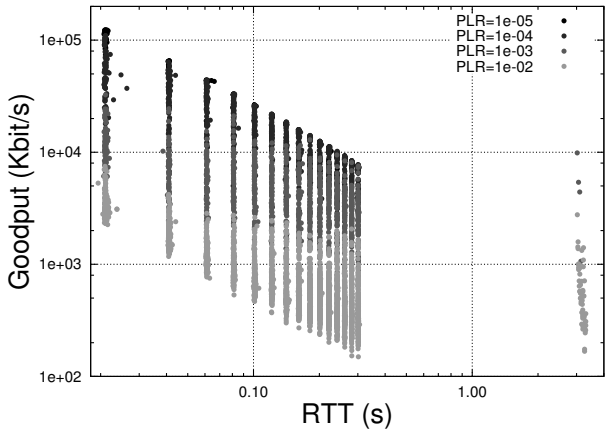

Figure 1: Goodput as a function of the RTT, in the testbed environment. Points are grouped by the magnitude of the imposed PLR.

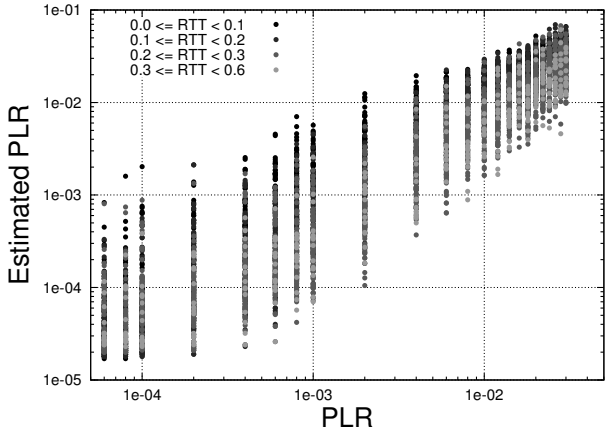

Figure 2: Estimated PLR, computed using Eq. (2), as a function of the PLR imposed using *netem*. Points are grouped by the magnitude of RTT.

lower the goodput is. Outliars on the bottom right of the plot are caused by lost SYN or ACK segments. Indeed, the measured RTT is three seconds (which is the default initial value of the retransmission timeout) plus roughly one RTT. These points have been omitted in subsequent plots and have not been used to compute PLR estimations.

Figure 2 shows the estimated PLR, computed using (2), as a function of the PLR and RTT imposed by *netem*. Each point is the result of a single transfer test and points with different RTTs are given different levels of gray. Although there is a lot of variation, the data pattern seems to suggest that points with smaller RTTs are more likely grouped at the top of the plot. Similarly, points with higher RTTs tend to group at the bottom.

To better investigate this behavior, we have averaged points having the same *netem* PLR and the same RTT order of magnitude. The results are shown in Figure 3: average curves follow loosely the bisection of the graph, which is the line that a perfect model would plot. More in detail, the figure leads to the following remarks:

(i) the PLR predictions are, in general, quite good;

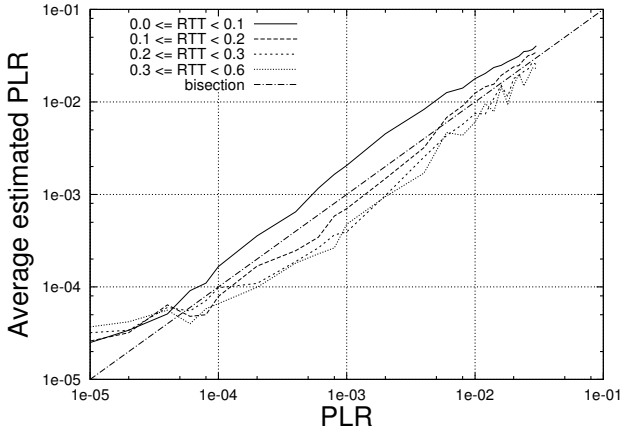

Figure 3: Averaged estimated PLR, computed using Eq. (2), as a function of the PLR, emulated with *netem*. Samples are grouped in buckets of size 0.1 s and then averaged.

(ii) the average of points with RTT lower than 0.1 s is above the bisection line, while all the other averages are below it: when RTT is small the model slightly overestimates PLR, the opposite is true for large values of RTT;

(iii) as expected, the model works best in the region where the PLR is not too small, namely, bigger than 10^{-4}.

To gain further insights into the model behavior, we analyze with *tcptrace*[6] the packet traces captured at the sender: Figure 4 shows the traces for two cases with significantly different RTT values and equal PLRs. Both connections exit from slow start with approximately the same outstanding window, but, since test duration is fixed, connections with higher RTT run for less rounds. So, they receive less congestion signals and cannot adjust their congestion window as quickly as the connections with smaller RTTs.

When PLR is very low or very high, TCP operating point is not the one assumed by Mathis model. In particular, when the PLR is very low, all connections are buffer limited. In this condition, connections with lower RTTs run for more rounds and deliver more bytes. Thus, their goodput is higher, and the estimated PLR is lower.

On the contrary, the combined effect of high PLR and high RTT makes the estimated PLR more noisy. Indeed, retransmission timeouts are more likely (since PLR is high) and TCP runs for fewer rounds (since RTT is high). Thus, the measured goodput varies a lot, depending on whether TCP experienced a timeout, and, in turn, the estimated PLR oscillates, as shown in Figure 3.

3.2 Evaluation using Internet experiments

Besides testbed experiments, the model was evaluated using Internet experiments, in which we estimate the default PLR of one ADSL and one WAN connection, and then we inject extra losses with *netem*, to check whether the model detects that. In both cases, the *netem* router was a Linux 3.0.0 workstation, connected via LAN

[6]Tcptrace [11] is a packet trace processing tool that performs several different types of TCP-level analysis. In particular, in this paper we have exploited two tcptrace features: the capability of counting the number of retransmission events that occurred during the lifetime of a TCP connection; the capability of measuring the amount of data sent but not acknowledged, also know as "outstanding window", which is a good approximation of the congestion window.

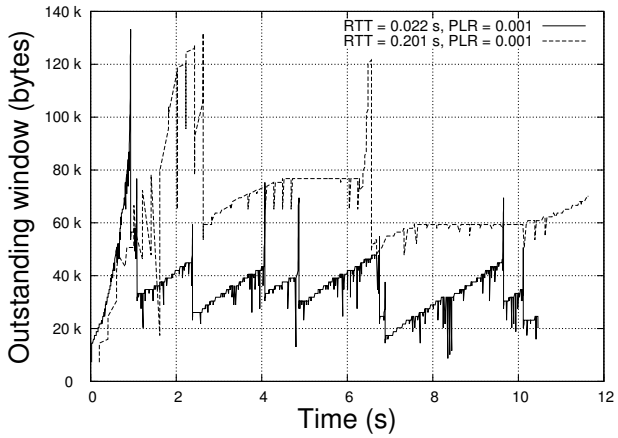

Figure 4: Outstanding window, computed using *tcptrace*, for two testbed connections, with equal PLR and significantly different RTT.

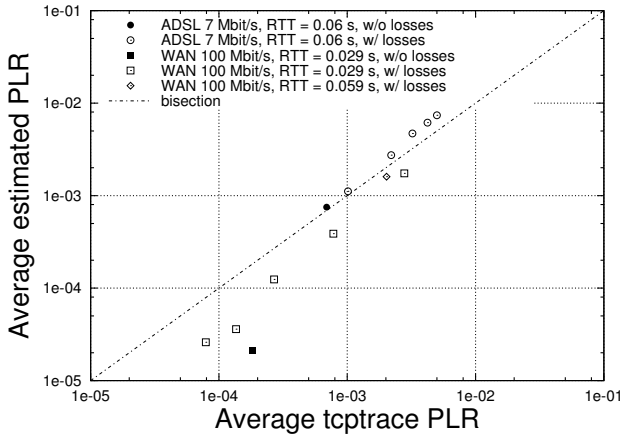

Figure 5: Average PLR of controlled Internet experiments, estimated using Eq. (2), as a function of average PLR, computed using *tcptrace*.

to our campus network and to a Linux 3.0.0 sender. In the ADSL case, the receiver was a MacOSX 10.6 laptop, the nominal downstream speed was 7 Mbit/s, and the average RTT during the tests was 0.06 s. In the WAN case, the receiver was a Linux 2.6 server, located in Germany, attached to 100 Mbit/s, and the average RTT during the tests was 0.029 s.

Figure 5 shows controlled experiment results. Each point is the average of 16 transfer tests, performed in similar network conditions. To estimate actual PLR on the x-axis, we run *tcptrace* on packet traces captured at the sender and divide *rexmt data pkts*[7] by *actual data pkts*[8]. Analysis of *time sequence graphs* for selected traces confirms that *rexmt data pkts* is a good approximation of the number of congestion signals, therefore we believe our low-level PLR estimate to be reasonably accurate (even if there is one exception, investigated and explained later).

Results show that the model can predict quite reliably the PLR of an ADSL connection. In particular, the prediction is accurate

[7]This variable is the count of all TCP segments with at least one data byte in the payload.

[8]This variable is the count of all TCP segments that *tcptrace* believes to be retransmissions.

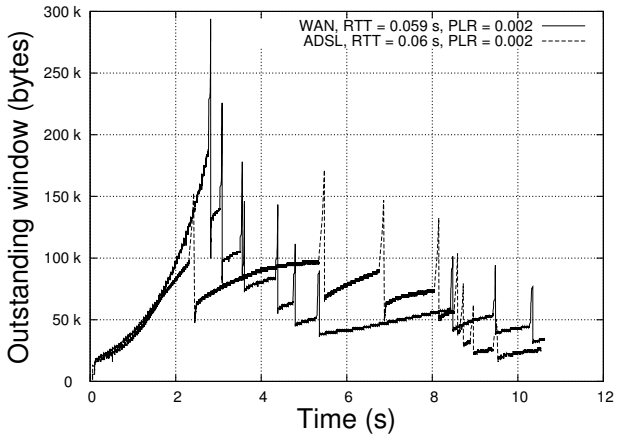

Figure 6: Outstanding window, computed using *tcptrace*, for WAN and ADSL connections, with similar imposed PLR, and where artificial delay was added to the WAN connection to make its RTT comparable with the ADSL one.

for PLRs around 10^{-3}, with an error of about 10%, and, while it is less accurate for higher PLRs, the error is always below 50%. Low-level analysis confirms that the error increases because the probability of timeout is higher, with higher PLRs (the window is smaller so it is more likely that TCP does not receive three duplicate ACKs). Incidentally, since our model is based on the Mathis formula, which does not model timeouts, it is reasonable for the error to increase slightly in presence of timeouts.

The case of the WAN connection is more problematic. On the one hand, the model detects the injection of extra losses. But, on the other hand, there are some inconsistencies, and low level analysis provides useful insights to help understanding and explaining them.

First, WAN-connection *tcptrace* PLR is lower when we impose a small extra amount of random losses with *netem*. This happens because the typical behavior of the connection is to saturate a drop tail bottleneck. So, *tcptrace* counts more retransmissions because more than one packet is lost per round. For example, during the worst run, *tcptrace* reports that 50 segments were retransmitted, but the time sequence graph shows that they correspond to five congestion signals only, producing two slow starts and one fast recovery. On the contrary, when we impose a small extra (uniformly and randomly distributed) PLR, the bottleneck queue is under control, there are less catastrophic congestion events, and the PLR estimated with *tcptrace* is closer to the actual number of congestion signals.

Second, the estimated-PLR error of WAN connection increases with low imposed PLRs. Low-level analysis with *tcptrace* shows that this happens because the connection "fills the pipeline" and the congestion window neither grows nor halves (and this is not because TCP is buffer limited, since there is plenty of space at the receiver, according to *tcptrace*). When this happens, TCP delivers more bytes per RTT than when it behaves as predicted by the Mathis model, hence the estimated PLR is lower than expected.

Third, the model reports that the WAN connection is always better than the ADSL, in terms of PLR. This is somewhat expected, and, to better understand what happens in terms of the model, we have run one extra experiment, where we impose the WAN connection exactly the same extra-PLR and RTT experienced by one ADSL connection experiment. This point is shown separately in Figure 5, and *netem* was employed to artificially bump the WAN connection delay to 0.06 s.

This experiment shows us that, even if we strive to put the WAN and the LAN connection on similar conditions, imposing the same RTT and PLR[9], there is always a small bias in favor of the WAN connection. What happens is that, whenever there is a random fluctuation and a cycle without imposed losses ensues, connections run at their native speed, and the WAN connection, which is 10x faster, is able to deliver more bytes.

Indeed, as shown in Figure 6, which plots the outstanding window of the best run of each connection, the WAN outstanding window is free to grow much higher during slow start. Moreover, both connections have a three-second loss-free period, and, in this period, the WAN congestion window increases linearly, while the ADSL one seems to saturate, probably because the pipeline is already full, and the connection is just filling the home router buffer.

4. CONCLUSION

This paper introduces a simple model, based on the Mathis formula, to estimate, at the application level, the packet loss rate (PLR) experienced by a TCP connection. To perform the estimation, the application needs to (i) measure the goodput of a bulk TCP transfer that runs for a fixed amount of time (10 seconds) and (ii) estimate the round trip time (RTT), before or after the transfer. Testbed experiments show that the model predicts the imposed PLR value with a residual (small) dependence on the RTT. Controlled Internet experiments show that the model can reliably predict the PLR imposed to an ADSL connection. The model can detect the injection of losses on a WAN connection as well. However, since the estimation is less precise in this case, our research efforts are currently focused on refining the model for this scenario.

5. REFERENCES

[1] cURL and libcurl. *http://curl.haxx.se/*.

[2] Data from M-Lab Tools. *http://measurementlab.net/data*.

[3] Glasnost: Test if your ISP is shaping your traffic. *http://broadband.mpi-sws.org/transparency/bttest-mlab.php*.

[4] GNU Wget. *http://www.gnu.org/software/wget/*.

[5] Measurement-Lab | M-Lab. *http://measurementlab.net/*.

[6] netem | The Linux Foundation. *http://www.linuxfoundation.org/collaborate/workgroups/networking/netem*.

[7] Network Diagnostic Tool (NDT). *http://www.internet2.edu/performance/ndt/*.

[8] Neubot | The network neutrality bot. *http://neubot.org/*.

[9] Neubot data | Neubot. *http://neubot.org/data*.

[10] NPAD - Network Path and Application Diagnosis project. *http://www.ucar.edu/npad/*.

[11] tcptrace - Official Homepage. *http://www.tcptrace.org/*.

[12] Basso, S. and Servetti, A. and De Martin, J.C. Rationale, Design, and Implementation of the Network Neutrality Bot. In *Congresso AICA 2010 (L'Aquila)*, 2010.

[13] Basso, S. and Servetti, A. and De Martin, J.C. The hitchhiker's guide to the Network Neutrality Bot test methodology. In *Congresso AICA 2011 (Torino)*, 2011.

[14] Basso, S. and Servetti, A. and De Martin, J.C. The network neutrality bot architecture: a preliminary approach for self-monitoring of Internet access QoS. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 1131–1136. IEEE, 2011.

[15] Bauer, S. and Clark, D. and Lehr, W. Understanding broadband speed measurements. 2012.

[16] Bischof, Z.S. and Otto, J.S. and Sánchez, M.A. and Rula, J.P. and Choffnes, D.R. and Bustamante, F.E. Crowdsourcing ISP characterization to the network edge. In *ACM SIGCOMM Workshop on Measurements Up the STack (W-MUST)*. ACM, 2011.

[17] Cardwell, N. and Savage, S. and Anderson, T. Modeling TCP latency. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1742–1751. IEEE, 2000.

[18] De Martin, J.C. and Glorioso, A. The Neubot project: A collaborative approach to measuring internet neutrality. In *Technology and Society, 2008. ISTAS 2008. IEEE International Symposium on*, pages 1–4. IEEE, 2008.

[19] Guillemin, F. and Robert, P. and Zwart, B. Performance of TCP in the presence of correlated packet loss. In *15th ITC Specialist Seminar on Internet Traffic Engineering and Traffic Management (Wurzburg)*, 2002.

[20] Mathis, M. and Semke, J. and Mahdavi, J. and Ott, T. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[21] Mellia, M. and Zhang, H. TCP model for short lived flows. *Communications Letters, IEEE*, 6(2):85–87, 2002.

[22] Padhye, J. and Firoiu, V. and Towsley, D. and Kurose, J. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 303–314. ACM, 1998.

[23] Palfrey, J. and Zittrain, J. Better Data for a Better Internet. *Science*, 334(6060):1210–1211, 2011.

[9]These are similar conditions since the two connections experience the same RTT. And since, for both connections, the imposed PLR is much higher than the default one, and dominates over it.