POLITECNICO DI TORINO
Repository ISTITUZIONALE

Strengthening measurements from the edges: application-level packet loss rate estimation

Original
Strengthening measurements from the edges: application-level packet loss rate estimation / Basso, S., Meo, M., DE MARTIN, J.C.. - In: COMPUTER COMMUNICATION REVIEW. - ISSN 0146-4833. - 43:3(2013), pp. 45-51. [10.1145/2500098.2500104]

Availability:
This version is available at: 11583/2516320 since:

Publisher:
ACM New York, NY, USA

Published
DOI:10.1145/2500098.2500104

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

07 November 2026

Strengthening Measurements from the Edges: Application-Level Packet Loss Rate Estimation

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

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

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

ABSTRACT

Network users know much less than ISPs, Internet exchanges and content providers about what happens inside the network. Consequently users cannot either easily detect network neutrality violations or readily exercise their market power by knowledgeably switching ISPs.

This paper contributes to the ongoing efforts to empower users by proposing two models to estimate – via application-level measurements – a key network indicator, i.e., the packet loss rate (PLR) experienced by FTP-like TCP downloads.

Controlled, testbed, and large-scale experiments show that the Inverse Mathis model is simpler and more consistent across the whole PLR range, but less accurate than the more advanced Likely Rexmit model for landline connections and moderate PLR.

Categories and Subject Descriptors

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

General Terms

Measurement, Performance

Keywords

Network neutrality, TCP, Application level measurements

1. INTRODUCTION

One of the focal points of the network neutrality debate is the deep information asymmetry between the network and its edges. Internet service providers (ISPs), Internet exchanges and content providers, in fact, have access to a wealth of information on data traffic, user behavior and peering. On the contrary, users have access to very coarse-grained information, mediated by a number of intermediaries (e.g., upstream providers, content delivery networks) and interfaces (e.g., the TCP/IP stack).

The first step to re-balance such information asymmetry is to develop tools and services that make access networks more transparent. There are several ways to increase transparency and several possible perspectives: in the following we list some of the most relevant projects in this area.

The Web100 project exposes the internals of the Linux TCP/IP stack [7] to userspace applications like NDT (the Network Diagnostic Tool) [3]. Measurement Lab is a distributed server-side platform that provides network-transparency applications with hosting and other services, including Web100 support [6]. Glasnost compares a certain protocol flow (e.g., BitTorrent) with a reference flow to detect traffic shaping and its cause (e.g., the port number, the payload) [5]. Neubot (the tool used in this paper) uses centrally-coordinated agents to run periodic multi-protocol network performance tests [2]. Dasu is a plugin for the Vuze BitTorrent client that combines passive and non-intrusive active measurements to monitor the level of service provided by its users' ISPs [10].

All these tools and services have the objective of collecting data from a great number of users and networks. If the tools operate at the application level (either as stand-alone applications or as plugins), they are more likely to involve a great number of users, although this design choice implies that the collected data is more coarse grained than the data collected with kernel level measurements.

To help gathering more data from application-level measurements, we recently proposed the Inverse Mathis model [1]. Inverse Mathis is a simple model (based on the Mathis model [8]) that provides a coarse-grained estimate of the packet loss rate experienced by TCP downloads from application-level measurements made at the receiver. The model is well suited to estimate the packet loss rate of any FTP-like large download; as such, it is meant to be implemented, e.g., into a web browser as a plugin, to analyze data collected during large downloads. However, the model is not applicable to cases when small messages are exchanged (e.g., Web browsing, BitTorrent).

In this paper we go beyond the Inverse Mathis model by introducing the Likely Rexmit model, a new model based on a more sophisticated analysis of the receiver's application-level data, specifically designed to be more accurate than the Inverse Mathis model for the important scenarios of ADSL and Fast Ethernet connections. To produce data useful to compare the new model to the previous one, we implemented a new test for our network measurement application, Neubot[1]. This test (called 'raw test') performs a TCP download from a random Measurement Lab server, and produces the data needed for both the Likely Rexmit and the Inverse Mathis models.

We analyze the results of running the 'raw test' from ADSL, Fast Ethernet and 3G connections (henceforth, controlled experiments), from a testbed where we varied the

[1]http://neubot.org.

packet drop rate (testbed experiments), and from the access networks of the 1,480 users that have so far installed Neubot (large-scale experiments).

The packet loss rate estimated by our models can be one of the many 'signals' that could feed a meta measurement system, such as the one envisioned by Palfrey and Zittrain [9], that collects insights from diverse sources (measurement tools, applications, content providers, ISPs) to measure the Internet from different perspectives and with different levels of precision, using coarse-grained results to trigger more specific measurements - a powerful approach to re-balance the information asymmetry currently disfavoring users.

The remainder of this paper is organized as follows. In Section 2 we describe the two models. In Section 3 we describe the 'raw test' implementation. In Section 4, 5 and 6 we describe the results of controlled, testbed and large-scale experiments, respectively. In Section 7 we draw the conclusions.

2. MODELS DESCRIPTION

In this Section we recall the description of the *Inverse Mathis* model and we introduce the *Likely Rexmit* model.

Both models use information collected at the application level only; both address a single TCP connection downloading data at full speed; both assume that the flow lasts long-enough to reach the equilibrium and that the impact of the initial transient is limited.

Consequently, both models can address HTTP and FTP downloads, but they can not properly model web browsing, since the browser typically fetches small objects, or BitTorrent-like peer-to-peer downloads, when small pieces are fetched from peers using request-response protocols.

Additionally, they can not properly model connections over very lossy links, when TCP never exits Slow Start, thus working in stop-and-wait mode.

2.1 The Inverse-Mathis Model

The Inverse-Mathis model [1] derives from the Mathis model, a well-known model that describes the macroscopic behavior of the TCP Congestion Avoidance algorithm and provides a simple formula to predict the goodput of a sustained steady-state TCP connection [8].

The Mathis formula predicts a TCP connection goodput given estimates of the packet loss rate (PLR) and of the round trip time (RTT). By inversion of the Mathis formula, we compute the Inverse-Mathis (Inv-M) estimated PLR (EPLR):

$$EPLR_{Inv-M} = \left(\frac{MSS \cdot C}{goodput \cdot RTT} \right)^2 \quad (1)$$

where:

1. *goodput* is computed by dividing the number of received bytes by the download time;
2. MSS is the maximum segment size;
3. RTT is the round trip time;
4. C is an empirical constant (defined by the original Mathis model) 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.

The Mathis model assumes that the connection bottleneck is always the network, i.e., that the PLR is always greater than zero. As a consequence, the PLR estimated by Inv-M is never zero. This means that it cannot detect cases when there are no losses (e.g., when the performance is limited by an under-dimensioned receiver's buffer), in which case its PLR estimate cannot be trusted.

Moreover, the Mathis model makes the assumption that the RTT is constant and equal to the RTT measured by the TCP sender during the download. This assumption implies that the Inv-M model accuracy depends on the accuracy of the RTT estimate.

In our implementation, the RTT estimate is the *connect delay* (i.e., the time required for the connect() system call to complete). We use the connect delay because it is readily available at application level. However, this choice poses a problem. It is well known in literature, in fact, that (especially in the context of home networks) the average RTT during a download is higher than the one at connect time due to the gateway's queuing delay [4].

To quantify the error caused by the gateway's queuing delay, in the following Sections we will compare the results of the Inv-M model using connect delay to the results of the Inv-M model using the average Web100 RTT (a case later indicated as 'Benchmark Inverse Mathis', or B-IM).

2.2 The Likely-Rexmit Model

We designed the Likely Rexmit (L-Rex) model when we noticed the negative impact of the gateway's queuing delay on the average RTT and, in turn, on the Inverse Mathis model accuracy. We specifically designed L-Rex to be more accurate than Inverse Mathis for ADSL and Fast Ethernet connections, which are, with cable, the most commonly used landline connections.

The model is called 'Likely Rexmit' because it analyzes the application-level dynamics of the recv() system call to identify likely 'rexmit' (i.e., retransmission) events.

To better understand L-Rex, it is useful to describe the steps we made to design it. We started from the empirical observation of what happens at the application receiver side of ADSL and Fast Ethernet connections:

1. since recv() returns as soon as data is available, it is usually triggered at the reception of segments. Typically, in slow networks (e.g., ADSL) one segment is received at a time, while faster networks (e.g., Fast Ethernet) receive one or two segments with similar probability. In short, any level 2 connection has its typical number of bytes returned by recv();
2. during a burst of packets, the time elapsed between the occurrence of two consecutive recv()s is very small, because many back-to-back packets are received.
3. between two consecutive bursts there is typically a 'silence period', which is usually smaller than one RTT. If the pipeline is full, however, the receiver sees the continuous arrival of more-or-less equally-spaced segments (i.e., there is no silence period).

Successively, we observed what happens after a loss:

4. the silence period is longer than usual (typically more than 1 RTT). In fact, recv() cannot return because it

is waiting for the missing segment. In the best case (i.e., Fast Retransmit) an additional RTT is needed to retransmit the packet and to trigger recv();

5. when the window is large, recv() returns to userspace a large, non-typical number of bytes (the lost segment plus all the segments that arrived while recv() was blocked).

We empirically noticed that many losses followed the pattern indicated by 4. and 5.: a longer-than-usual silence period, and a large, non-typical number of bytes. We used this information to define two empirical rules that count the number of estimated losses, which, in turn, is needed to compute the estimated packet loss rate.

The two empirical rules operate on a list that contains the number of bytes received by each recv() and the time elapsed since previous recv(). As a preliminary step, before we can apply the rules, we need to scan the list once, to compute the number of received bytes frequency. Then, we scan again the list and we count the number of cases for which:

(i) the time elapsed since the previous recv() is greater than $H \cdot RTT$;

(ii) the number of received bytes is (a) greater than 1 MSS and (b) less frequent than K.

Once the number of estimated losses is known, the L-Rex estimated PLR (EPLR) is computed as follows:

$$EPLR_{L-Rex} = \frac{losses}{received_bytes \ / \ MSS} \qquad (2)$$

where *losses* is the number of estimated losses, *received_bytes* is the number of received bytes, and MSS is the maximum segment size.

From repeated experiments, we optimized the value of the parameters H and K to maximize the number of identified losses for ADSL and Fast Ethernet access networks with moderate losses[2]. The optimal values are, respectively, 0.7 and 1%.

Unlike Inverse Mathis EPLR estimates, the L-Rex EPLR can be zero. However, there is no certainty that a zero value corresponds to no losses. For example, when there are losses that do not match the empirical rules (i) and (ii) above, the EPLR is zero.

3. IMPLEMENTATION

To produce data useful to compare the Likely Rexmit (L-Rex) model to the previously-developed Inverse Mathis (Inv-m) model, we wrote and deployed a new ad-hoc Neubot test, called 'raw test'. This test is an active transmission test that connects to a random Measurement Lab server (on port 12345) and performs a 10-second random-data TCP download.

During the 'raw test', the sender saves TCP state variables, exported by Web100. Notably, it saves: (a) *CongestionSignals*, the number of congestion signals experienced by TCP; (b) *SegsOut*, the total number of segments sent by TCP, including retransmitted ones.

[2]ADSL and Fast Ethernet were chosen because they represent important Internet access technologies to which we had easy access.

Meanwhile, the receiver saves application-level data: (c) the *connect delay*; (d) the *goodput*; (e) the *maximum segment size*, obtained via getsockopt(); (f) a list that contains the number of bytes returned by each recv() and the time elapsed since the previous recv().

The (a)-(f) data above allows us to compute the Inv-M and the L-Rex PLR, using the procedures described in Section 2.1 and 2.2. It also allows us to compute the real PLR (RPLR) by dividing *CongestionSignals* by *SegsOut*. This is the PLR experienced by the transmitter, and we will use it as the ground truth to evaluate both models.

4. CONTROLLED EXPERIMENTS

In this Section, we discuss the results of 'raw test' experiments performed on access networks for which we could capture packets (called 'controlled experiments' because we had full control over the computer where Neubot was installed). The objective is to investigate how both models approximate the Real PLR (RPLR).

4.1 Qualitative Overview

Nine controlled experiments on a combination of Internet connections (ADSL, Fast Ethernet and 3G) and platforms (Linux, Windows, and MacOS)[3] were run. Neubot testing policy was modified to start a new 'raw test' every 15 seconds. Tests were performed towards random Measurement Lab servers. Each experiment is relative to one connection and one operating system, and contains at least 100 tests.

Fig. 1 shows the results of 3 experiments for both the Inverse Mathis (Inv-M) and the Likely Rexmit (L-Rex) models. The dashed bisector line represents the exact prediction by a model that covers the whole RPLR range. Therefore, the closer to the bisector the points are, the more accurate the model is.

Inv-M points are reasonably close to the bisector, but do not evenly distribute around it. L-Rex points, instead, are considerably closer to the bisector and are more evenly distributed. The points distribution suggests that we can reduce the error by averaging multiple experiments on the same network path.

Windows XP ADSL points differ significantly between the two models, with L-Rex points much closer to the real PLR. Tcptrace[4] reveals that Windows XP performance is in most cases (79%) limited by the receiver's buffer; therefore, Mathis assumptions are not met, and the corresponding EPLR cannot be trusted.

4.2 Median Behavior

In the previous Section, we qualitatively showed that the L-Rex model is more accurate, with points evenly distributed around the bisector. Now, the question is whether we can reduce the noise and obtain a robust estimate of the typical PLR of a network path. As an indication of that, we use the median PLR.

We decided to use the median, and not the average, because the former is more robust to outliers. The PLR, in fact, can potentially range over a few orders of magnitude,

[3]We never changed the system's default TCP/IP configuration; therefore, we always run a flavor of NewReno. Windows 7 and Linux automatically scaled their TCP buffers, while MacOS 10.6 and Windows XP had fixed buffers.

[4]http://www.tcptrace.org/.

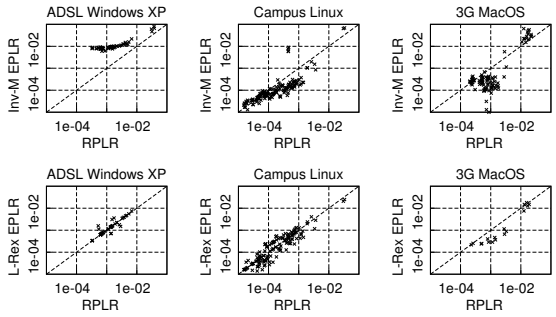

Figure 1: Estimated PLR (EPLR) versus the real PLR (RPLR).

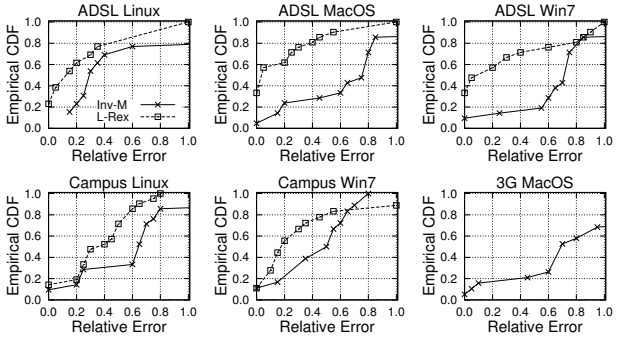

Figure 2: Empirical cumulative distribution function (CDF) of the relative error.

and we want to avoid that occasional large values skew the estimate. For example, if the typical PLR is $5 \cdot 10^{-4}$, one single sample equal to 10^{-3} can considerably shift the average.

To study the models median behavior, we have performed repeated controlled 'raw test' experiments with seven Measurement Lab sites (corresponding to 21 servers). They were selected to represent near (i.e., same continent) and far (i.e., transoceanic) destinations. From the results we computed the RPLR, the Inv-M EPLR and the L-Rex EPLR.

Successively, we grouped experiments performed by the same client with the same server together, and we computed the medians (ignoring cases where we had less than 4 samples).

4.3 Comparison of the models

Fig. 2 shows the empirical cumulative distribution function (CDF) of the relative error, computed on the medians. There are at least 13 points for each CDF.

The distribution indicates that L-Rex is more accurate than Inv-M. There is good probability that the L-Rex error is limited; e.g., in the ADSL Linux case, for more than 75% of the samples the relative error is lower than 0.4.

In Fig. 2, no CDF is plotted for 3G MacOS L-Rex; we investigated and we noticed that the relative error is 100% (i.e., the median EPLR is zero) in 18 cases out of 19. This happens because the variance of the number of bytes returned by recv() was higher than it is for the ADSL and Fast Ethernet connections for which we optimized L-Rex; therefore, the 1% threshold was too strict to detect the vast majority of the losses. In our future work, we look forward to better adapt the 0.7 and 1% thresholds to the characteristics of the access network.

We also investigated whether L-Rex is more accurate than Inv-M because of the RTT estimation error induced by the gateway's queuing delay. To do that, as anticipated in Section 2.1, in the Inv-M formula we replaced the connect delay with the average Web100 RTT. This change reduced the error, but the model was still not as accurate as L-Rex, suggesting that the gateway's queuing delay is not the only reason why Inv-M is less accurate than L-Rex (more on this point in Section 6).

5. TESTBED EXPERIMENTS

In the previous Section, we have tested both models on access networks that we controlled. However, we did not

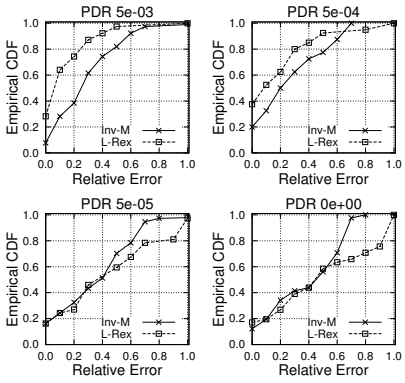

Figure 3: Empirical cumulative distribution function (CDF) of the relative error for varying packet drop rates (PDR).

control the typical loss ratio of the end-to-end path, which was a mixture of bursty losses (caused by the gateway queue dynamics) and much-less-bursty background losses, typical of the traversed (long distance) networks.

In this Section, we use a testbed to add random losses to the typical background losses of a network path. This allows us to study how the model behaves when the background loss rates of a network path increases (simulating what happens when there is congestion or when a Provider is actively managing router queues).

To run the experiment, Neubot was installed on a Linux box which was attached to a Linux gateway in our campus Fast Ethernet network. The gateway was configured to discard packets with a random loss model and no correlation. To do that, we used Netem, the Linux kernel's network emulation framework[5].

Successively, we run 'raw test' experiments with varying packet drop rates (PDR) at the gateway. We varied the PDR from 0 (i.e., the PLR was the typical PLR of each network path) to $5 \cdot 10^{-3}$ (where the PDR dominated over the typical PLR of every path we tested).

Fig. 3 shows the empirical cumulative distribution function (CDF) of the relative error. Each experiment consists of at least 37 samples, with only three experiments with

[5]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

RPLR equal to zero (two for PDR equal to $5 \cdot 10^{-5}$, and one for PDR equal to zero).

The trend is that Likely Rexmit (L-Rex) is more accurate than Inverse Mathis (Inv-M) when the PDR is higher than $5 \cdot 10^{-5}$. For lower PDR, L-Rex is less accurate than Inv-M. This trend reflects the general behavior of the models for different RPLR ranges, as we will see in the next Section.

6. LARGE-SCALE EXPERIMENTS

In this Section, we analyze the results of 'raw test' experiments run from the access networks of the 1,480 users that have installed Neubot. The objective is to study the models accuracy at different RPLR ranges (including when the RPLR is zero).

6.1 Data Overview and Preprocessing

Our dataset contains the results of the 'raw test' performed by Neubot instances installed worldwide. The dataset is in the public domain and can be downloaded freely from the Neubot web site[6].

From 18th November 2012 to 17th February 2013, 565,559 tests were performed by 22,681 unique IP addresses and by 1,480 unique Neubot IDs, using four diverse platforms: Win32 (259,910; 46% of the tests), MacOS (228,712; 40%), Linux (75,497; 14%), and FreeBSD (1,440; 0.25%). We will concentrate on the first three cases. Tests were performed towards 33 Measurement Lab sites (corresponding to 99 distinct servers), with a nearly uniform distribution of the load among them. Countries with more tests are: the U.S. (195,059; 34%), Italy (92,210; 17%), Germany (67,496 11%), France (32,098; 5%), and the U.K. (27,271; 4%).

Neubot does not gather information on the type of access network used by the device on which it is installed (i.e., whether it is fixed or mobile). However, a quick analysis of the involved Autonomous Systems shows very few mobile providers.

From the above data, we computed the RPLR, the Inverse Mathis (Inv-M) and the Likely Rexmit (L-Rex) EPLR. As anticipated in Section 2.1, we also estimated the PLR with Inv-M using the average Web100 RTT in place of the connect delay. We will indicate this case as the 'Benchmark Inverse Mathis' (B-IM).

As in Section 4.2-4.3, we compute the medians to reduce the noise caused by outliers. We group together experiments performed by the same client with a Measurement Lab site, and we compute the median EPLR and the median RPLR (ignoring the cases with less than 8 samples).

6.2 Relative Error

Fig. 4 shows the empirical cumulative distribution function (CDF) of the relative error computed on the medians, for different RPLR ranges and operating system platforms. Unlike Section 4, here we can not distinguish between Windows 7 and XP because Neubot identifies only the platform. Tab. 1 shows the number of points used for the CDF.

In Fig. 4, for high RPLR (i.e., greater than 10^{-3}) and moderate RPLR (i.e., between 10^{-4} and 10^{-3}), the L-Rex model error is reasonably limited, and the model is more accurate than Inv-M: in the worst case (Win32 for moderate RPLR), in fact, L-Rex has lower relative error than Inv-M for 65% of the samples. However, for low RPLR (i.e., lower than 10^{-4}), it is less accurate than the Inv-M model.

[6]http://neubot.org/data

RPLR	MacOS	Linux	Win32
$[10^{-3}, \infty)$	1,456 (24%)	378 (20%)	1,529 (22%)
$(10^{-4}, 10^{-3})$	2,179 (35%)	839 (44%)	2,115 (30%)
$(0, 10^{-4}]$	389 (6%)	373 (19%)	357 (5%)
0	2,148 (35%)	333 (17%)	3,039 (43%)

Table 1: Number of points in Fig. 4 (first three rows) and Fig. 5 (last row).

As we noted in Section 2.2, in fact, the L-Rex model was designed and optimized for landline networks with moderate losses. This explains why it is more effective than Inv-M for RPLR greater than 10^{-4}.

We performed one extra experiment on a Gigabit Ethernet with default MSS (1460 bytes) to understand L-Rex reduced accuracy with low RPLR. We noticed that the variance of the number of bytes returned by each recv() was higher than it is for the ADSL and Fast Ethernet connections for which we optimized L-Rex; therefore, the 1% threshold was too strict to detect all the losses.

The large-scale experiments confirm the gateway's queuing delays. Indeed, when we use the Web100 average RTT in place of the connect delay, we typically obtain a more accurate estimate (indicated as B-IM in Fig. 4). In particular, for Win32 and MacOS, B-IM is the most accurate model for both high and low RPLR conditions.

To complete our comparison of the two models, we study now the case where the RPLR is zero. We did not study it before because in controlled and testbed experiments the case was not frequent. Instead, as Tab. 1 shows, this case is more significant in large scale experiments.

6.3 Absolute Error When RPLR is zero

Fig. 5 shows the empirical cumulative distribution function (CDF) of the absolute error for different operating system platforms when the RPLR is equal to zero. The error is computed on the medians and is equal to the EPLR (since the RPLR is zero). Tab. 1 shows the number of points used for the CDF.

When the RPLR is zero, L-Rex is the best model, because in the worst case (Win32) it yields zero 65% of the times. On the contrary, Inv-M is much less accurate, but this should not come as a surprise: as we anticipated in Section 2.1, in fact, the model EPLR cannot be zero.

The distribution also shows that the L-Rex model is less likely to yield high estimates when the RPLR is zero. In general, L-Rex is the most accurate model (equally with B-IM for Linux). In the worst case (Win32), for 25% of the samples the model yields a value greater than 10^{-4} when the RPLR is zero.

6.4 Comparison of the models

Large scale experiments results confirm that L-Rex is more accurate than Inv-M to estimate moderate to high PLR. However, L-Rex is less accurate than Inv-M for low PLR. L-Rex is also good at detecting when there are no losses (even though this is not the case for which it was designed).

Inv-M, instead, is consistent along the whole PLR range. However, Inv-M is less accurate than L-Rex for high and moderate PLR, and it cannot identify (by construction) the case when the PLR is zero.

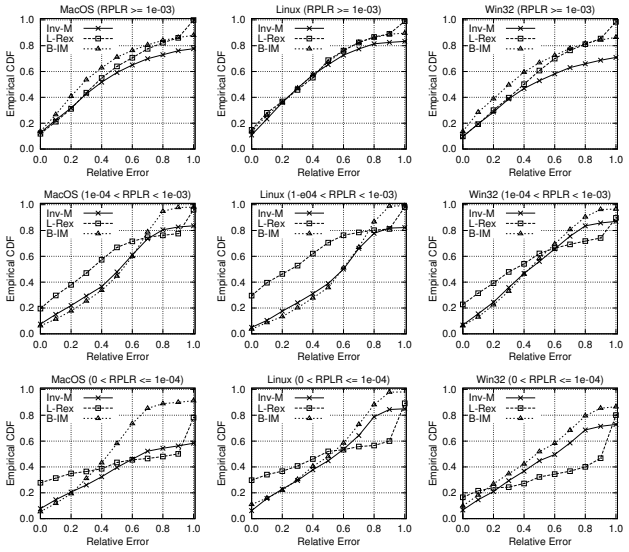

Figure 4: Empirical cumulative distribution function (CDF) of the relative error for different RPLR ranges and operating system platforms.

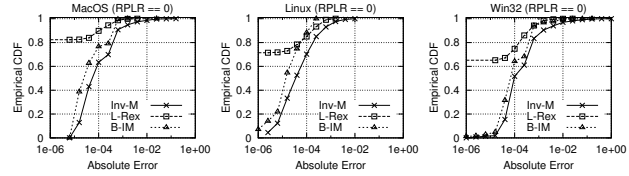

Figure 5: Large-scale experiments: Empirical cumulative distribution function of the absolute error for different operating system platforms when RPLR is equal to zero.

We also observed that Inv-M precision is reduced by the gateway's queuing delay problem. This suggests that the model can be improved by using a better average RTT estimation. However, since B-IM (which uses Web100 average RTT) is still less accurate than L-Rex for moderate RPLR (see Fig. 4), the gateway's queuing delay seems not to be the only reason why Inv-M is less accurate than L-Rex.

7. CONCLUSION

In this paper, we proposed the Likely Rexmit model for estimating the packet loss rate (PLR) experienced by a sustained TCP connection (e.g., an FTP download) from application level measurements.

We compared the proposed model to our previous model (Inverse Mathis) in large scale experiments, as well as in experiments performed in access networks we controlled (e.g., an ADSL that we could observe), and in a testbed, where we varied the packet drop rate.

Results show that the Likely-Rexmit model is more accurate than the Inverse-Mathis model for PLR greater than 10^{-4}. However, Likely Rexmit is more complex because it gathers data each time recv() returns, and, at the end of the transfer, it needs to process such data to identify likely retransmissions. Also, Likely Rexmit works poorly with mobile networks. The Inverse Mathis model, instead, is simpler and less accurate, but its accuracy is more consistent across the whole PLR range, and it is also much more reliable with mobile networks.

To summarize, the Likely Rexmit model should be clearly preferred over the Inverse Mathis model for landline access networks when moderate PLR are expected.

Our experiments also show that Likely Rexmit accuracy is slighly lower (e.g., Gigabit Ethernet) or significantly lower (e.g., mobile) in the access networks where the variance of the number of bytes returned by each recv() is higher than what Likely Rexmit assumes. As part of our future work, we aim to make Likely Rexmit generally more accurate by replacing the static threshold on the number of bytes returned by each recv() with a more dynamic rule that depends on the distribution of the number of received bytes.

We also plan to perform controlled experiments on cable access networks, as well as experiments to study the effect of competing traffic over the network link (e.g., a background download from Windows update).

8. REFERENCES

[1] S. Basso, M. Meo, A. Servetti, and J. C. De Martin. Estimating Packet Loss Rate in the Access Through Application-Level Measurements. In *ACM SIGCOMM W-MUST 2012*, pages 7–12, 2012.

[2] S. Basso, A. Servetti, and J. C. De Martin. The Network Neutrality Bot Architecture: a Preliminary Approach for Self-Monitoring of Internet Access QoS. In *IEEE ISCC 2011*, pages 1131–1136, 2011.

[3] R. Carlson. Developing the Web100 Based Network Diagnostic Tool (NDT). In *PAM 2003*, 2003.

[4] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *ACM SIGCOMM IMC 2007*, pages 43–56, 2007.

[5] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *USENIX NSDI 2010*, pages 405–418, 2010.

[6] C. Dovrolis, K. Gummadi, A. Kuzmanovic, and S. Meinrath. Measurement Lab: Overview and an Invitation to the Research Community. *ACM SIGCOMM Computer Communication Review*, 40(3):53–56, 2010.

[7] M. Mathis, J. Heffner, and R. Reddy. Web100: Extended TCP Instrumentation for Research, Education and Diagnosis. *ACM SIGCOMM Computer Communication Review*, 33(3):69–79, 2003.

[8] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

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

[10] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing Experiments to the Internet's Edge. In *USENIX NSDI 2013*, pages 487–499, 2013.