Dynamic Performance Limits of the Benko-Veres Passive TCP Packet Loss Estimation Algorithm

Claudio Favi[1], Grenville Armitage
Centre for Advanced Internet Architectures.
Swinburne University of Technology
Melbourne, Australia
claudio.favi@epfl.ch, garmitage@swin.edu.au

***Abstract* - In this paper we evaluate and discuss the performance of a passive TCP packet loss estimation algorithm described by Benko and Veres. Our analysis is motivated by recent work modeling web traffic patterns and the network conditions experienced by TCP sessions carrying web traffic. We experimentally determine the dynamic characteristics of the Benko-Veres estimation algorithm, focusing on how quickly the algorithm converges to a reasonable estimate of packet loss rate for a given TCP session and path. The Benko-Veres algorithm is stimulated in a small IP network testbed, using repeated TCP sessions over a path with variable and controlled packet loss rates. Our results show that the Benko-Veres algorithm requires many packets and loss events before it converges to reasonably accurate loss rate estimates. Finally we identify and discuss the limited real-world scenarios where this method of passive, external estimation of packet loss rates can be applied.**

Keywords- Traffic Characterization, TCP, Packet Loss

I. Introduction

This paper describes our evaluation of a previously published technique for passively estimating packet loss on individual TCP sessions. Our interest in passive loss estimation derives from a broader project aimed at modeling web traffic dynamics with realistic network conditions [4]. We want to build on previous work estimating the cachability of web objects [2] and the dynamic rates of change of web objects [3] by experimentally determining the packet loss rates typically seen by TCP sessions originating and terminating on web servers. Our ultimate goal is to produce a network simulator that embodies experimentally verified, real-world network characteristics.

Benko and Veres described a method for estimating packet loss by passively monitoring TCP sessions at an independent measurement point on the network [1]. Their algorithm estimates packet loss rates by observing inconsistencies in the sequence number progression seen in TCP packets belonging to a particular flow. However, they do not evaluate the convergence speed of their algorithm.

For our purposes the convergence speed is the number of packets that must pass the measurement point before the loss rate estimate is within a reasonable percentage of the actual loss rate. Real TCP sessions have finite durations and packet counts, and the loss characteristics of an IP path can change over time. The convergence speed of a loss estimation algorithm places a practical bound on how much we can accurately know about loss rates through passive monitoring of real-world TCP sessions at intermediate points along an IP packet path.

For these reasons we decided to implement a small testbed (based on Figure 1) capable of introducing controlled, constant packet loss rates and experimentally evaluate the Benko-Veres algorithm's dynamic characteristics. Given our ultimate real-world scenario involves tapping close to operational web servers we can make the simplifying assumption that measurement occurs at a place where IP routing is symmetric and packets in both directions will be seen and captured.

The rest of this paper explains the Benko-Veres loss rate estimation algorithm, outlines our test methodology, and discusses the implications of our results. We also make some suggestions for improving their algorithm.

Several passive TCP session analysis tools are available to the research community. Tcptrace [5], Tstat [6] and Nprobe [7] are examples of those. Neither tcptrace or tstat has a built in packet loss estimator and although Nprobe claims to be able to compute packet loss its code had not been released when we performed our experiments. We used Tstat as a starting point to code the packet loss estimator algorithm.

II. Testing The Benko-Veres Algorithm

A. Algorithm description

First we summarise the Benko-Veres algorithm [1] with reference to the generalised network in Figure 1.

The client opens a TCP connection to the server, sends some data and then closes the connection. The monitoring point M sees all the packets exchanged by the client and the server and, by examining those packets' sequence numbers, estimates the packet loss rate in network A and network B.

The basic idea of the algorithm is to analyse the sequence number progression from the perspective of the measurement point M, and infer where and which packets are lost. An out of order sequence number that has previously been seen implies that the corresponding

1 This work was performed while working for Swinburne University of Technology. Claudio can be contacted at claudio.favi@epfl.ch

packet was lost in the network *after* the measurement point. On the contrary, an out of order sequence number that had never been seen before implies that the corresponding packet (first transmission) was lost *before* it got to M hence lost in the network before M. The algorithm doesn't try to use acknowledgment packets or any sort of timing estimation to infer losses.

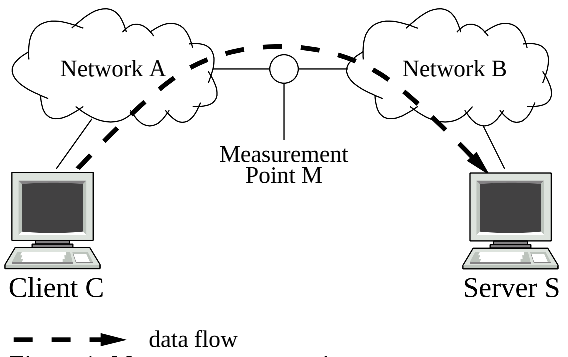

Figure 1: Measurement scenario

To further understand the algorithm it is necessary to realize that we're not interested in the exact number of packets lost, but only in the loss ratio. We select a subset of packets for which it can be determined, with relatively high probability, whether and where the packets were lost. Those packets are called "significant packets".

When we see a given TCP segment for the first time, that packet is marked as a "significant packet". Based on the TCP segment's sequence number, and using the algorithm described above, we then decide whether this packet indicates a loss event has occurred (either before or after the measurement point). The running loss estimate is the number of loss events divided by the total number of "significant packets" at any given point in time.

To improve the accuracy of the loss estimation, Benko and Veres ignore the first and last segments of a TCP session. They also ignore any sequence of three or more "significant packets" that have been flagged as representing consecutive loss events.

It is important to note that the monitoring point, M, is passive – it does not represent a congestion point, and does not introduce packet loss artifacts of its own. It might typically be implemented using a broadcast hub (on low speed links), 'port-mirroring' (off high speed routers) or some form of optical tap.

B. *The Testbed*

Our actual testbed consisted of three x86-based hosts running FreeBSD 4.8 [8] - a client (C), a server (S) and a third machine (B) acting as an Ethernet bridge (Figure 2). The bridge host utilized FreeBSD's kernel-resident packet filtering (ipfw [9]) and traffic shaping (dummynet [10]) functionality to introduce configurable and controlled packet loss on the IP path between the client and server. (Dummynet has previously been shown to generate uniformly distributed packet loss at rates within a few percent of configured values [12].) We used a freely available tool (ypfw [11]) to ease the configuration of ipfw, and used Python [13] scripts to create the customized client and server programs.

Each test run involved the client connecting to the server (via a TCP socket) and generating 10000 send calls to the socket with random data. The server would simply read and discard the transmitted data. To ensure that for each data unit we want to send a TCP packet is generated, and that on a loss event an exact copy of the original packet is retransmitted, we created random data with a length equal to the difference between the link layer MTU and a TCP packet header length. (In our case this was 1500-52, or 1448 bytes.) We run several simulations for different packet loss rates.

We run tcpdump on the client thus limiting the results to the after-monitoring-point precision of the estimator. Although our preliminary tests showed not much difference between the before and after monitoring point cases.

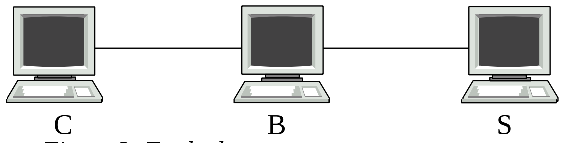

Figure 2: Testbed

It is important to note that node B did not introduce an uncontrolled congestion point – all packet losses introduced by B were configured through dummynet.

C. *Measuring the algorithm's dynamic behaviour*

We began by running one trial at each of a number of actual packet loss rates, and plotted the algorithm's running estimate (as a function of packets transmitted) in Figure 3. Not surprisingly the algorithm converges faster at higher loss rates because there are more loss events from which to refine its running estimate.

Each trial run has a certain amount of randomness inherent in the interaction between TCP and the distribution of packet drops introduced by dummynet. Figure 4 shows how the broad shape of the algorithm's convergence is nevertheless relatively consistent (in this case over 30 trials with the bridge configured for a 3% packet loss rate).

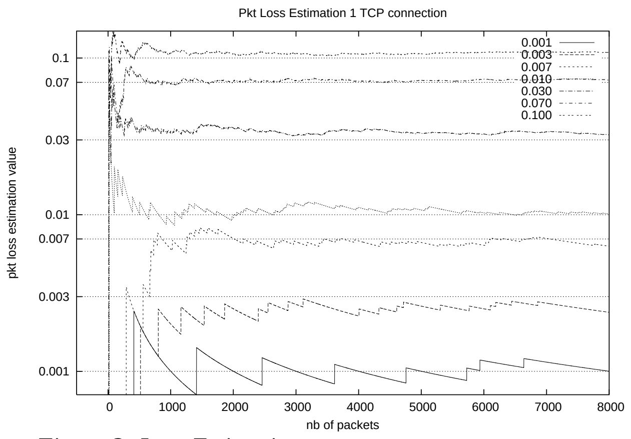

Figure 3: Loss Estimation

Figures 3 and 4 clearly suggest that the algorithm estimates the real packet loss rate quite well in the long run. But we are interested in knowing the minimum number of packets needed to be sure that the estimate is within a certain percentage of the actual loss rate.

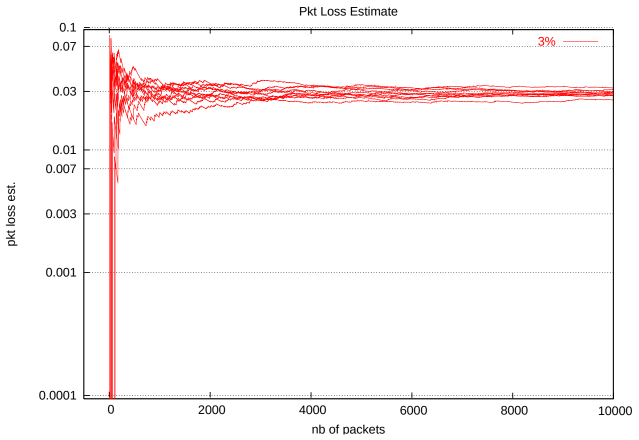

Figure 4: Estimation @ 3% packet loss

The next set of figures focus on the *estimation error*, expressed as the difference between the estimated and actual loss rate as a fraction of the actual loss rate.

Figure 5 shows the normalized maximum estimation error at each loss rate, plotted as a function of packets transmitted. We ran 30 trials at each simulated packet loss rate, and then calculated the *maximum envelope*[2] of the range of estimates produced over the 30 trials (i.e.. the upper and lower bounds of the curves in Figure 4). We then computed the difference between the real packet loss rate and the maximum envelope's upper and lower bounds, which is then normalized (expressed as a fraction of the loss real rate).

Using the maximum envelope gives us an insight into the likely worst case accuracy of the loss estimator. For example if we wish to obtain an estimate that almost certainly lies within 30% of the real value for packet loss rate above 0.7% Figure 5 shows we need to see around 4000 packets on the TCP connection being monitored.

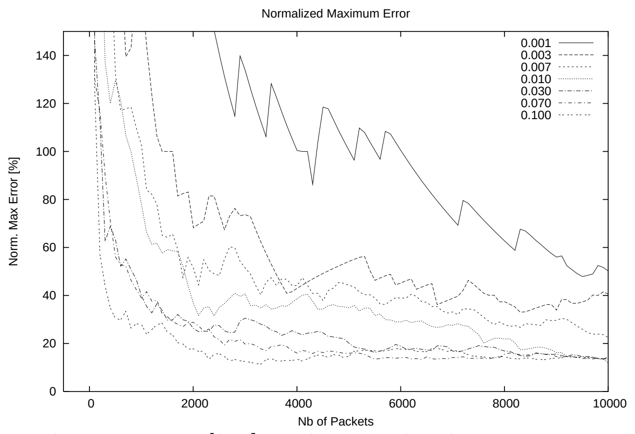

Figure 5: Normalized Maximum Estimation Error

The maximum envelope provides a worst-case estimate on the number of packets we would need to see for a given estimation accuracy. For comparison we also calculated a 'one standard deviation estimation error' plot – the number of packets required for the estimation error to be below a certain value with 68% likelihood (assuming the estimates of different runs, e.g. in Figure 4, are normally distributed in the Y-axis).

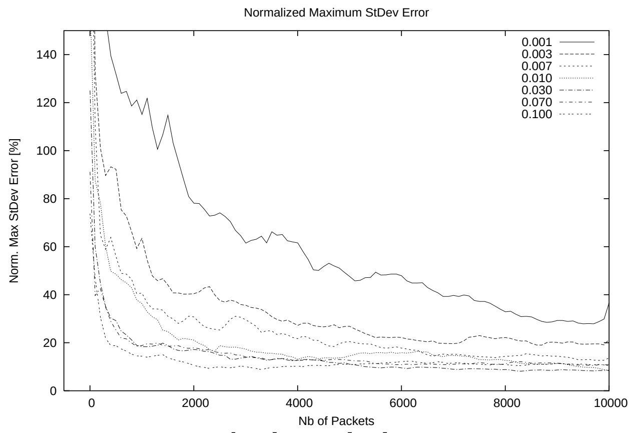

Figure 6: Normalized Standard Deviation Estimation Error

We computed the average estimation error and standard deviation (as opposed to the maximum and minimum for the envelope), then calculated the maximum of the difference between the real rate and the computed mean plus or minus the standard deviation. The results are shown in Figure 6. For example, when the real error rate is 0.7% (0.007) we would need to see around 1500 TCP packets to get an error estimate that is 68% likely to be better than 30%.

Better accuracy may be obtained by calculating the mean of the repeated loss rate estimates if the same TCP session can be repeated over time periods where the network path is essentially unchanged. Figure 7 shows the plot of the mean of the relative error (the computed mean used in Figure 6). In this case an estimation error of less than 30% can be obtained by repeating TCP connections that each last only a couple of hundred packets, regardless of the real loss rate.

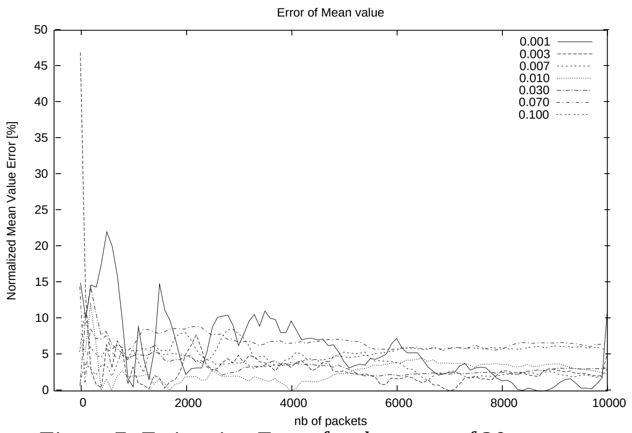

Figure 7: Estimation Error for the mean of 30 runs

2 To calculate the envelope we windowed the data in blocks of 50 packets interval and computed the max and min of all the samples in the window.

Copyright © 2004, ATNAC 2004, Australia
ISBN 0-646-44190-6

III. Discussion

Clearly the Benko-Veres algorithm will generally converge, but it takes quite some time (many packets) to converge on a significantly accurate loss estimate. Naturally it converges quicker the higher the actual loss rate (given that the algorithm refines its running estimate each time a loss event occurs). Our first question is to identify the real-world traffic scenarios in which the Benko-Veres algorithm would perform acceptably. A secondary question is how we might improve on the algorithm.

The following disscussion raises points for future work stimulated by this paper.

A. *Usage Scenarios*

Our overall motivation is monitoring traffic in and out of popular web servers to characterize the packet loss statistics along the paths out to each server's many clients. In conjunction with IP header information from the passive network tap we hoped to build maps relating estimated loss rates and client locations (derived from their IP addresses and TTLs). However, it is clear that we will not be able to estimate losses to clients who are simply downloading typical sets of web pages (non-streaming text and graphical content).

The first reality is that web pages designed for consumer access are going to trigger downloads towards the client in the 20-60KBytes range [2] [16] The associated TCP session would generate in the order of tens to hundreds of IP packets – not quite the 1000 or more packets we see as the low end for reasonably bounded estimation error.

A second reality is that we cannot take unbounded periods of time (in the real time domain) to make our loss estimates because the network conditions causing loss are themselves time varying. For example, we might think to average over many TCP sessions to the same client over a period of visits to the web server (appealing to the benefits shown in Figure 7). But if the client's actual download speed (or web site traversal speed) is low the real network conditions (and hence loss rates) are likely to have fluctuated over the sampling period.

We would have more success mapping the loss rates to clients who are accessing a streaming-content server, or downloading large multi-megabyte files, at a reasonably high packet rate. For example, if we assumed an MSS of 1480 bytes along a TCP session's path, then downloading a typical 620MByte CDROM image would generate in the order of 440K packets – significantly in excess of our requirements. Coming from the other direction, if we set a target of 2000 packets then this would be achieved by clients downloading content in the 3MByte range – probably not unusual, for example, of the server is an archive of mp3 files.

Multi-site enterprise VPNs are another scenario where this loss rate estimator might find applicability. For example, imagine a field office doing nightly file system backups (using a TCP-based file transfer application, e.g. rsync [15]) over a VPN link back to head office. The VPN link has been sold with a particular service level agreement (SLA) mandating worst-case packet loss rates. Over a period of time a passive monitoring point would see enough TCP sessions (and enough packets per session, if the nightly updates involved more than a few megabytes of data) to reach a relatively accurate estimate of actual loss rates over the VPN link. This has potential applications for the VPN customer to ensure their ISP is meeting the SLA.

B. *Improving the Algorithm*

As published, the Benko-Veres algorithm simply averages its loss rate estimate over all the significant segments since the beginning of the TCP session. This means that the estimated loss rate over a very long lived session might not reflect a slow change in the loss rate over the time of the session. (For example, if the loss rate was 1% for the first ten thousands packets, and then 5% for the next ten thousand packets, the algorithm would estimate a loss rate around 3%.)

Now that we know how quickly the algorithm converges, we could propose to introduce long term memory loss into the algorithm. In other words, the estimator would calculate a running average based solely on the last X thousand packets, where X depends on your desired estimation accuracy and the current estimated loss rate. Based on the figures shown earlier in this paper, X is likely to be between two and six.

We would also propose that a better loss estimator would use information in ACKs flowing in the opposite direction to improve the packet loss event detection. Matching ACKs to data packets and inferring losses would be a start although such an improvement requires lot of complexity added to the algorithm.

Also by adding time awareness to the algorithm one could improve it greatly. It requires estimating the end-to-end round trip time between hosts and being able to tell losses from delayed answers due to computation load. Previously published work uses a similar technique to study the effect of “Early Packet Loss” on web pages download times [14].

C. *Changing the testbed*

One key difference between our testbed and likely real-world scenarios involves our use of uniform packet loss distribution at node B in Figure 2. Future research on packet loss estimation (whether incremental improvement to Benko-Veres or entirely new approaches) should 'drive' the testbed with non-uniform packet loss distributions (e.g. Randomly spaced short bursts of packet loss, perhaps more representative of congestion-based loss in the network.)

IV. Conclusion

We have presented a detailed analysis of the Benko-Veres packet loss rate estimator [1], showing its limitations and discussing in which cases and to what extent its result are usable. We find that it typically needs to see at least one to two thousand packets before its estimated loss rate is within 30-50% of the actual loss rates on the TCP session's path. Because of the algorithm's convergence rate, it is suitable for passively monitoring loss rates on TCP sessions carrying objects upwards of 3-4MBytes long, but is not suitable for estimating packet loss rates on TCP sessions carrying short-lived (20-100Kbyte) HTTP responses to typical web page GET requests. The estimator may also find use passively monitoring SLA-conformance of multi-site VPN links where the VPN customer does regular, repeated TCP-based file transfers (e.g. nightly file system dumps between remote and head offices). We conclude by observing that our convergence time analysis allows us to propose a refinement to the algorithm such that it "forgets" any packet traffic more than two to six thousand packets in the past. This would allow the algorithm to more accurately track loss rate estimates on extremely long lived sessions.

References

[1] Peter Benko and Andras Veres, "A Passive Method for Estimating End-to-End TCP Packet Loss", Proc. IEEE Globecom 2002, Taipei, Taiwan

[2] S. Zander, G.J. Armitage, C. Malcolm, "Dynamics and Cachability of WebSites: Implications for Inverted Capacity Networks," 11th IEEE International Conference on Networks (ICON 2003), Sydney, Australia, September 2003

[3] C. Malcolm, G.J. Armitage, "Bandwidth Efficient Web Object Change Interval Estimation" Australian Telecommunications Networks & Applications Conference 2003 (ATNAC 2003), Melbourne, Australia, December 2003

[4] "Inverted Capacity Extended Engineering Experiment (ICE3)", Centre for Advanced Internet Architectures, http://caia.swin.edu.au/ice/ (as of February 2004)

[5] S. Osterman, "TCPTrace, TCP traces analyser" (http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html) (as of February 2004)

[6] Tstat, "TCP Statistics and analysis Tool" (http://tstat.tlc.polito.it/) (as of February 2004)

[7] Nprobe, "Network protocol analysis" (http://www.cl.cam.ac.uk/Research/SRG/netos/nprobe/publications/pam2003hall.html) (as of February 2004)

[8] FreeBSD, "Operating System", http://www.freebsd.org (as of February 2004)

[9] IPFW, "IP Firewall for FreeBSD", http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls.html (as of February 2004)

[10] Dummynet, "Bandwidth management tool", http://info.iet.unipi.it/~luigi/ip_dummynet/ (as of February 2004)

[11] C.Favi, "YpFw, ipfw frontend," http://caia.swin.edu.au/ice/tools/ypfw (as of February 2004)

[12] W.A Vanhonacker, "Evaluation of the FreeBSD dummynet network performance simulation tool on a Pentium 4-based Ethernet Bridge," CAIA Technical Report 031202A, Centre for Advanced Internet Architectures, Swinburne University of Technology, December 2003 (http://caia.swin.edu.au/reports/031202A/CAIA-TR-031202A.pdf) (as of February 2004)

[13] Python, "Programming Language" (http://www.python.org) (as of February 2004)

[14] J. Hall, I. Pratt, I. Leslie and A. Moore, "The Effects of Early Packet Loss on Web Page Download Times", Proceedings of the Fourth Passive and Active Measurement Workshop (PAM 2003), April 2003 (http://www.cl.cam.ac.uk/Research/SRG/netos/nprobe/publications/pam2003hall.html) (as of February 2004)

[15] Rsync, "Fast incremental file transfer" (http://samba.anu.edu.au/rsync/) (as of February 2004)

[16] Cyvelliance, "Sizing the Internet", July 2000 (http://www.cyveillance.com/web/corporate/white_papers.htm) (as of February 2004)