Regenerating TCP Dynamics From Traces Path Characteristics

Cesar Marcondes[1], Claudio Palazzi
M. Y. Sanadidi and Mario Gerla
Computer Science Department
University of California (UCLA)
Los Angeles, California, USA 90095
Email: (cesar,cpalazzi,medy,gerla)@cs.ucla.edu

Magnos Martinello[2]
and Marcos Tadeu Torres[3]
Computer Science Department
Universidade Federal do Espírito Santo (UFES)
Vitoria, ES, Brazil 29060
Email: (magnos,tadeu)@inf.ufes.br

Abstract— **A deep knowledge of TCP performance on the Internet at large is intimately related to understanding the role of the underlying path characteristics (bottleneck rate, buffer sizes, end-to-end delay, loss rate) on the performance of dynamic TCP flows. The practical way to study the TCP performance is by simulating on a dumbbell topology, Internet statistically equivalent cross-traffic along with simulated TCP, then extracting interest variables such as throughput as the indicator of performance. However, we advocate that much more can be learned about TCP performance by regenerating path characteristic conditions derived from traces, and let TCP flows run loose on top of it. In this study, we investigate several Internet public TCP flows, using LAN-specific traces (Berkeley LBNL traces), and we show how path characteristics can be extracted. We devise a new technique to obtain path capacity by searching longitudinal patterns of packet pairs on each TCP flow present in a trace. In addition, we look on buffer usage patterns based on effective throughput/estimated capacity ratio, as a way to relate RTT variation, packet loss and disclosure buffer sizes. Finally, the natural application is to test advanced TCP proposals on a per-trace scenario basis by regenerating dynamic traffic patterns directly from stored traces, such method is also called source-based traffic pattern. In this trace regeneration testbed, we have as input the TCP trace, then, a post-processing mechanism inferring path characteristics per flow. Finally, emulated virtual machines running in parallel enable the TCP traffic dynamics to run loose.**[123]

I. Introduction

In this paper, our goal is to present ideas for low cost, scalable path characteristics estimations and TCP replayability based purely on observed TCP behavior derived from traces. As a proof-of-concept, we study and estimate the network path capacity and buffer size of networks traces from a well-know dataset of the LBNL Lawrence Berkeley National Lab enterprise network. This dataset promises to be an interesting material for our study, since previous papers [14] [13] have exposed in detail the application heterogeneity of the experimental samples and the process of anonymization. In addition to the derivation of path conditions, we describe in great level of detail the design of a research testbed to regenerate TCP dynamics from traces.

[1]Cesar Marcondes is supported by CAPES/Brazil #1283-02-2
[2]M. Martinello is supported by CNPq/Brazil Pos-Doc Junior (PDJ)
[3]M. T. Torres is supported by CNPq/Brazil #50.6284/04-2

Extract path characteristics information from Internet traces, such as capacity, delay variation, etc. is an accurate way to represent the exact conditions on which TCP protocol were running prior to be stored on a trace file. If one could extract precisely such conditions and emulate them using a controlled testbed, the emulation of Internet dynamics using regular TCPs and source-level behavior information could be achieved. The regeneration helps on the development of new transport protocols, switching equipment and tuning end-servers.

We begin by verifying that the current literature in the area of replayable traces is in its early stage, the important papers so far include [1] and [18]. In [1], the authors extract basic information about the delays of the TCP connections to feedback on a Google Search Server. They model the service rate using a customized algorithm that execute similar tasks as the Google Search Engine does. However, the material fall short on exploiting path capacity estimations, buffer size variability and thinking user times. The paper [18] shows another possible way to use the interarrival of packets from traces as thinking times and response times. In this study, the authors basically extract the exact gap period between packets, and tag them as transactions time (socket reads/writes), therefore there is no separation on queuing time and service time. In addition, their testbed does not support different end-to-end capacities and buffer variability.

Therefore, there is a lack on using precise path characteristics to lead TCP performance research. Such issue can be further explained by the lack of tools to extract path characteristics from stored traces based on IP/TCP headers. Thus, we propose a simple technique derived from packet pairs serialization delay and its relationship to capacity to obtain precise path capacity from traces, called TraceProbe. From this information we can infer capacities from TCP traces by searching specific packet pair patterns. Moreover, this tool is developed in python script language and the results are applied to the mentioned enterprise trace dataset. For validation purposes, a proper comparison with another passive estimation tool, PPRate [16] demonstrates the accuracy of our method and a pathological discrepancy based on flow controlled application such as ssh and NFS on Section 2.

Another major effort in this paper is on the study of buffer estimation on traces, on Section 3. We start our approach by collecting a straightforward metric from the traces, namely, the relationship between the RTTs from the TCP feedback loop and the RTT just before the buffer overflows. The latter is identified through a packet loss or 3 duplicate ACK. In order to derive reliable buffer sizes, our approach also requires the knowledge of narrow link capacity beforehand. The buffer size scheme is validated through simulation of realistic Internet traffic patterns. It is then applied to the LBNL traces, using the narrow capacity estimation from the previous set of experiments.

The final piece is the detailed description of applications including the design of a research testbed for replayable traces. Our goal is to develop a system to automatically extract path characteristics on a per-trace basis. Then prepare an end-to-end structure where the flows maximum capacity is shaped according to the derived characteristics. In addition, the delay and buffer variability behavior is to be emulated using middleware software, in order to reproduce with high fidelity the environment where the trace was collected. We conclude the paper, on Sections 4 and 5, by introducing another possible application that reveals the path capacities behind the router ports where the traces were obtained as a way to fingerprint specific machine connectivity in the enterprise LAN.

II. Gathering Capacity Estimations From Traces

Capacity estimation has been a very attractive doctrine on Internet measurement research. The work on [3] and [9] are among the earliest and most complete on the area, which subsequently led to the creation of Pathrate, a very popular active tool to assess capacities. Recently, researchers have looked into the possibility of removing the active restriction posed by the access to both end points in order to obtain path capacity.

This has led to the extention of the capacity estimation work to passive measurement (in particular, to the study of traces). PPRate [16] is one example of those efforts. It uses a method similar to Pathrate: it relies on a relatively large number of interarrival samples (at least 300) to reveal capacity modes. In addition, it uses a statistical process to identify modes, and to select the mode corresponding to the narrow link capacity.

In spite of this early solution, passive capacity estimation is an active area of research. For example, PPRate has shown several drawbacks. Firstly, it tends to require considerable computation effort. In addition, as mentioned in [16] there is an intrinsic difficulty on dealing with application flow control on top of TCP. If one has a large number of idle periods, it potentially generates a bias on the capacity mode detection histogram, when packet pairs are not correctly identified. Other areas of potential improvement are convergence time (for real time applications) and number of samples necessary for sufficient reliability on the estimate.

Among the active tools, one in particular, Capprobe [10] deserves special attention as the least expensive computationally, without post-processing statistical method. Given its low cost, CapProbe will be the starting point from which we are proposing to extend our work on scalable dataset trace studies. The basic active principle of Capprobe consists of sending pairs of packets back to back from source to destination, in order to estimate the capacity. Once passing through the narrow link capacity, this packet pair ought to be dispersed according to the maximum sending rate. Thus, the dispersion represents an estimation of the end-to-end narrow link. The key aspect of Capprobe is that packet pairs could be impacted by cross-traffic, hence by applying a proper filter, identifying the packet pair that suffers the minimal queueing delay, the dispersion measurement can be made reliable.

We devise a technique based on these same insights in our tool TraceProbe. The key advantages compared to PPrate are: low computational cost since it doesnt need an extensive statistical processing, just one pass through the data and the correct identification is sufficient. Another advantage is that it needs only 40 samples [10] to obtain a good estimation what makes it a valuable tool for wider analysis of data traces.

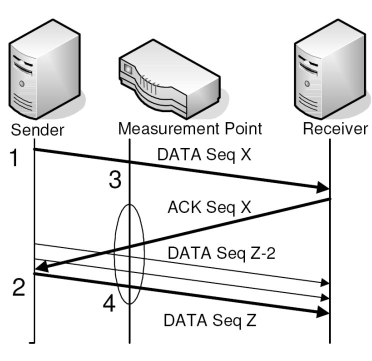
A. Estimating RTT

In addition to the task of finding the right capacity, our method relies on RTTs been estimated from network flows stamped in the traces. This RTT estimation is also known to be non-trivial. In fact, it has received recently a fair amount of attention (i.e. [17]). The main point of difficulty is that we are interested in inferring the RTT from the TCP sender perspective while observing the communication at the measurement point. In addition the measurement point can be located in any part of the path.

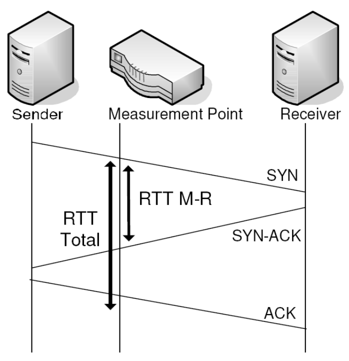
Indeed, this RTT estimation can be obtained by a simple observation that the distance 1 to 2 and 3 to 4 is similar (Figure 1(a)). Therefore, if one can obtain the association of DATA packet X and Z, the problem of determining RTT estimation from the sender would be solved. However, since TCP expand its congestion window and packets could be in-flight the association is non-trivial in the trace.

We describe briefly 3 known methods to infer this RTT.

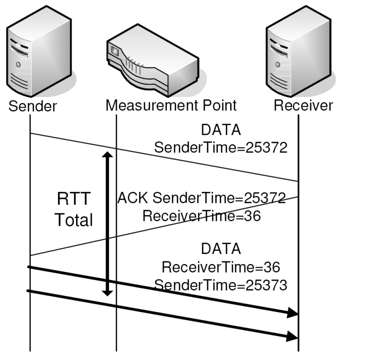
- The first, and hardest one, is based on emulating the state of the congestion window that an ideal TCP flow would have, from the packet communication seen in trace file [9]. The difficulty of this method is based on the fact that there is a vast heterogeneity of TCP implementation on Internet hosts, leading to difficulty on emulating different



(a) RTT 3-4 is a projection of RTT 1-2



(b) Measurement Point Distance



(c) RTT Estimation using TCP Timestamp

Fig. 1. RTT Estimation on Traces

reactions to loss, non-conformity with standards and other issues.

- The second method is the simplest one, and relies on the identification of the position of the measurement point compared to sender and receiver. If the position is close to the sender, RTT estimation is trivial (Figure 1(b)).
- Finally, the third method is based on the usage of the TCP timestamp option ([8]), where the sender echoes the receiver internal clock in the optional TCP header, while the receiver echoes the sender clock as well (Figure 1(c)).

This last RTT estimation method is the basis for our passive capacity estimation technique. Instead of just estimating RTT, we search specifically for matching packets (ACKs and DATA packets) and estimate delays on the packet pairs. A method like this permits precise identification of the DATA packets following an ACK. Furthermore, its advertised that the option is well deployed in the Internet, according to previous research 76.5% of Forbes 500 web servers [4] have support for this option, so we expect our method to work on a large sample of the TCP flow population.

B. The Search of Packet Pairs

The quest for packet pairs, the set of packets that define precisely the physical line rate, is the key for realistic capacity estimation on traces. These microbursts would happen mainly during slowstart and at congestion avoidance only once per 1-2 RTT, when the window increases ([3]). So the question becomes how to identify these micro-bursts in a trace easily, without relying on congestion state-machine schemes? We claim that the guaranteed method of search consists on do association of pairs of packets on the traces using TCP timestamp and applying the minimum delay sum in a longitudinal way, specially on the duration of slow start, where we will be able to obtain sufficient samples. We implemented the described methods, in script language by relating (ACK)-(DATA)-(NEXT DATA) packets (figure 1(b)) on a longitudinal search until the end of the slow start phase is reached. Our results are based on the dataset previously described, and we validate comparing results with another popular tool, PPRate. The following is a precise description of the Traceprobe algorithm (Algorithm 1).

Algorithm 1 TraceProbe Algorithm - Longitudinal Search

```
1: Phase 1 - Longitudinal Search Loop
2: Search for Delayed ACKs
3: if $PreviousACK - CurrentACKNumber > 1MSS$ then
4:     Store this ACK
5:     Current_Packet.type = (ACK)
6:     Current_Packet.num = (ACK TCP Timestamp Number)
7:     if Current_Packet.type == (DATA) && Current_Packet.num == (ACK TCP Timestamp Number) then
8:         if DATA TCP Timestamp Number + 1 then
9:             Found a ACK, DATA, DATA + 1 Triple
10:            Store Association Triple
11:        else
12:            Next Packet
13:        end if
14:    end if
15: end if
16: End Loop
17:
18: Phase 2 - Apply Capprobe Filter
19: $pair \leftarrow minSumRTT(AssociationTripleArray)$
20: $firstPacket \leftarrow minFirstPacketRTT(AssociationTripleArray)$
21: $secondPacket \leftarrow minSecondPacketRTT(AssociationTripleArray)$
22: $sumRTT \leftarrow pair[0].rtt + pair[1].rtt$
23: if $sumRTT <= (firstPacket.rtt + secondPacket.rtt)$ then
24:     $dispersion \leftarrow pair[1].timeStamp - pair[0].timeStamp$
25:     $secondPacketSize \leftarrow pair[1].packetSize$
26:     $narrowLinkCapacity \leftarrow secondPacketSize \div dispersion$
27: end if
```

C. Traces Handling

The dataset of LBNL/ICSI Enterprise Project consists of roughly 11GB of data, spread over more than 150 trace files. We start our analysis by extracting each TCP flow contained in the dataset, since our methods work on a per TCP flow basis. From the traces, half-million flows were generated by applying a collection of in-house tools and tcptrace, ethereal, tcpsplit, etc. to obtain the raw data on a per-flow basis. Based on previous results on this dataset [13], we know beforehand that the majority of the byte transport is carried out by TCP (particularly network file systems, scp, backup and bulk).

For statistical purposes, we started cutting off short lived connections that are intrisically difficult to assess, by creating a threshold of flows of more than 100 pkts, the number of connections dropped immediatelly to about 42,000 unique TCP flows. For the sake of comparison, and due to the restriction of PPrate at least 300 interarrivals, this number dropped again to 13,125 spread over all the router monitored ports. While we notice that 15,082 supported the TCP timestamp option. In spite of the reduction on the number of flows, the extracted sub-population is still representative of the amount of traffic transferred through LBNL, since the remaining flows are large in size, and most of the demand on this site is known to be TCP-related.

III. Capacity Estimation on LBNL

We evaluate our capacity estimation scheme by comparing it with PPRate. We use the PPRate matlab code provided in [5] to extract interarrival times to feed the statistical process of estimation capacity modes. On the same traces, we also run our tool in order to compare and have insights about both. Figure 2 presents the capacity estimation on the population from a common TCP set for both tools. In other words, the intersection of restrictions of both tools (timestamps + 600 samples) restricted the study to 5000 flows. As we have already pointed out earlier, this small population is still representative since we got samples from the majority of the ports and address space monitored.

The graph shows us some insight about the organization and the internal connectivity, while most of the flows have capacity about 100Mbps, there is a small fraction of 10Mbps connections and a number of sub-Mbps, that is related to the nature of some applications, like ssh.

From the direct handling of the discrepancy cases, we have verified that some applications do not actually follow the TCP phases (slow-start and congestion avoidance). The sending rate of the packets seems to be determined by a higher layer state-machine transport architecture. These lower sending rate samples happen to disrupt the PPRate statistical analysis, while Traceprobe relies only on the microburst identification. This way, regarding the mode distribution there is more smoothness due to statistical process on PPRate compared to Traceprobe when taking into account flow-controlled applications.

A. Algorithm Performance Evaluation

Our advantage compared to PPRate lays in the low cost solution. From the chart (Figure 2) both have obtained comparative results. In fact, the relative error (the difference of the same estimation by the two tools) shows that nearly 70% of the estimations were equal. In addition to the need of a commecial Matlab license, several drawbacks turn PPRate tool expensive computationally. In particular, PPRate needs the estimation of multi-modes and a smoothness fitting phase, several order of algorithmic complexity larger (up to $O(n^3)$) than the single straighforward lookup on Traceprobe ($O(nlogn)$, assuming the triples are stored in a sorted fashion). "n" in this asymptotic analysis represents the interarrival input size.

IV. Deriving Buffer Space from Traces

The next step on the trace assessment is the buffer estimation. These are the memory/physical storage of forwarding devices, consisted of constrained NIC, switches, and routers. Prior work in buffer size estimation has included [11] and [2]. In [11], the authors utilize the modal round trip time from passive packet pair to infer queue size, while [2] addresses queue estimation via an ICMP mechanism. While all prior techniques have small footprints, they sample infrequently compared to TCP methods. In particular, in case of [2], the estimations rely on network services, which are sometimes disabled on Internet routers. Our in-band method infers buffer size based upon the many RTT samples gathered from TCP traces. The buffer estimation technique we utilize is based upon a simple observation. The amount of time to traverse a queue completely full multiplied by the capacity yields the buffer size of a network entity. Thus, the capacity technique mentioned in the prior section is used to calculate the buffer size.

Let's assume that the traversal of packets go through the same path, and TCP is the underlying protocol generating traffic. In addition, another assumption is that a TCP flow is short enough such that the bottleneck do not shift due to change in network dynamics. Thus, these assumptions lead to the common scenario of TCP naturally trying to overflow the bottleneck. In fact, after"n" RTT cycles in congestion avoidance, TCP will insert "n" extra packets in the pipe eventually filling the bottleneck buffer.

As the bottleneck router queue is filled, every packet that traverses the bottleneck, in addition to the propagation time and service time (similar to all packets), will have an additional queueing time embedded in the RTT metric. At the end, TCP will force a packet to be dropped, since the bottleneck buffer will not hold the extra amount. At this exact moment, the packet p_n will be dropped (identified by the first duplicate ACK of a 3 DUPACK). It is p_n that serve as an indicator that the prior packet p_{n-1} or the last good packet is the most likely packet to have experienced the queue completely full. Hence, we can use p_{n-1} round trip time as a measure of the traversal time of the queue.

We refer in this section to the p_{n-1}'s RTT as the last good RTT. Trivially, the round trip time (current RTT) of p_{n-1} is a function of two factors, the queuing delay (queue delay) and the path delay (propagation delay), assuming transmission and processing negligible. Therefore, We will measure buffer using the queuing delay: $current\_RTT^{p_{n-1}} = queue\_delay + min\_RTT$. The path delay (min_RTT) is obtained by extracting the smallest RTT estimated out of the flow trace.

A. Buffer Estimation Validation

Even though some prior research sources [15], [7] have shown evidence (by simulation or controlled measurements) that relating RTT samples, as a predictor of buffer overflow is not sufficient in the context of the Internet, we claim that

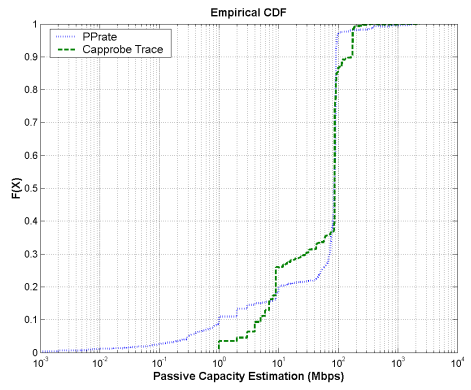


Fig. 2. Side by Side Capacity Estimation - TraceProbe versus PPRate

there is still relevant information available on these RTT samples that can help on uncovering the amount of buffer space been used on a path, and ultimately discover typical buffer spaces allocated on enterprise networks and Internet.

We start our argumentation, considering that there is no known easy way to assess buffer. The bursty aggregate traffic of the Internet (long-range dependent LRD) potentially perturbs measurements of this nature. Thus, we argue that what is needed is extra information that can make buffer size predictions more reliable. We advocate that such information can be derived from the flow trace and it's the amount of utilized capacity, obtained by comparing the ratio of narrow link capacity and aggregate throughput passing through the path.

The validation of these ideas was done via simulation using the ns-2 [12]. In a typical research scenario consisted basically of a bottleneck is shared with long-lived on-off pareto sources (Figure 3). The basic configuration of the simulation is the following: access links with bandwidth of 50Mbs, local propagation delays of 1ms. Access network links with bandwidth of 7.5Mbs and propagation delay of 40ms. All buffers on each router along the path were set to 50 packets and are pure drop tail policy. We launch connections performing bulk transfers using FTP between the nodes "Start" and "End", where we monitor (in the access router) both the minimal RTT and the RTT just prior to a packet loss (3 DUPACK) throughout the connection, as our buffer space interest measure. These bulk transfers are 600 seconds long in order to simulate download of large files (long-lived flows), proven to be good to gather reliable loss quantities.

We studied mainly the impact of demanding realistic traffic scenarios on the metric, such as the cross-traffic pattern generated an similar LRD traffic by using agreggated on-off sources (on the links 3-4), the parameters choice was: idle time=0.5 sec, burst time=0.1 sec, pareto shape 1.5 and rate equal 2.25Mbps.

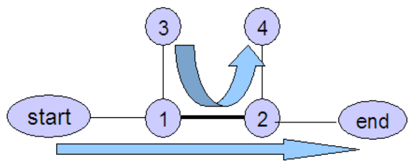
As expected, the burstiness of the traffic impacted the end-to-end TCP congestion control performance. The following presents the frequency on which the RTT samples are obtained (early drops due to bursts) and moreover leading to underestimation of the buffer space by our metric. While exploring this issue in detail, we performed simulations using parallel flows as a way to gain easily in performance, keeping the ratio capacity/achieved throughput close to 1. And we observed that indeed the burstiness impact on the estimation could be alleviated if enough achieved throughput is passing through the path over-loading it properly. Such results are in conformance to findings described in [15].

In Figure 3, there is only one bottleneck (and the aggregate Pareto sources do not fill up the capacity). We calculate the buffer based on converting the delay signal into a physical utilization of the buffer. The first curve shows the distribution of "last good RTT" or buffer size estimate distribution inferred from a single flow crossing the start-end segment. We can verify how it is shifted with the median pointing to buffer size of 28.49. In subsequent tests, we include parallel connections (Figure 3) to present the insight of buffer sampling on the presence of a large aggregate throughput passing. The aggregate in fact, help to narrow down the variability and skew the buffer estimation to a better estimation position (median size = 39.47 and 44.89 in the respective cases of starting 4 and 8 parallel connection). The inference is done on a single flow basis but the aggregate throughput can be obtained using a trace.

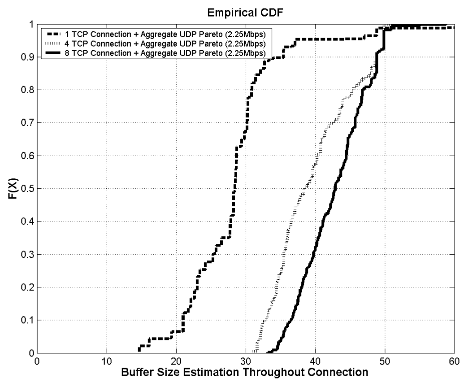
In the next sub-section, we apply our buffer space techniques on the enterprise traces, separating the best performers as our sample population and looking at the aspect of buffer size conformity to regular values applied by vendors.

B. Buffer Inference on LBNL Traces

Our analysis of LBNL trace files was a multi-step process. First, we isolated the highest throughput flows to exploit the lessons learned in simulation. Secondly, we chose the



(a) Simulation Topology



(b) Bursty Cross-Traffic

Fig. 3. Simulation Evaluation - Buffer Estimation Under Realistic Bursty Traffic

flows that were closest to the sender. Third, we log both the RTT for every packet sent and the RTT value before a loss occurred (last good RTT).

We derived meaningful RTT measurements from enterprise traces by collecting samples closer to the sender (the easiest way to estimate RTT). To select these flows, we used the RTT Estimation method of inferring the distance of the measurement point. The idea is to calculate the time interval between SYN and ACK as an estimate of RTT and the total distance. RTTm-r characterizes the distance of the measurement device from the receiver while RTTtotal represents the total distance from the sender to the receiver. We chose the flows that maximized RTTm-r / RTTtotal. In other words, if the starting point of the flow measured by the trace was really close the sender, LBNL flows.

This method is subject to delayed ACKs. We argue that the delay imposed by the receiver will be accounted for by the sender and, hence, is an accurate representation. We found this to be true of the flows selected. Unlike capacity estimation, RTT loss measurement techniques require distributions formed by much more samples and therefore, necessitate larger flows. Our analysis of these flows revealed a long tail distribution for the RTTs. This result motivated the use of median as a central value which best describes the last good RTTs gravity center.

Typically, we would choose the minimum RTT amongst all RTTs to characterize path delay. Our computation of the median RTT and the capacity resulted in buffer size estimations from 64 KB to 600KB buffers for some of the flow traces. This result is consistent with some of the switch manufacturers. Another key observation was the trend of multimodality on buffer sizes. Figure 4 presents a 64KB buffer estimation from a flow trace and a smoothed fitting curve showing the multimodality pattern of certain buffer sizes.

V. Applications

A. *Replayable TCP Flows*

The task of building a research testbed that can reproduce TCP flows from traces needs several software pieces to be glued together in order to create the right testbed environment. We have developed so far TCP trace analysis, and we are designing and implementing the delay box and validation of original and emulated traces on a Linux testbed.

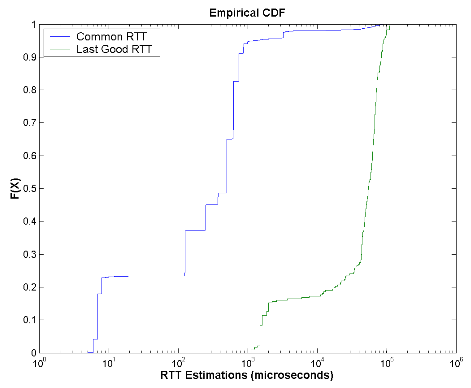
In terms of trace analysis, in addition to the difficulties described in the previous sections, we need a per IP-address path characteristics linear-time majority algorithm to be included. The idea is to shield the network estimations by making several flows from the same address to agree on the order of magnitude and values of the estimators. So, the first step is to gather and analyze many TCP traces in order to extract path characteristics. This process involves the extraction of every flow in order to obtain the round trip times (RTT) and the buffer sizes. The analysis tools were developed on the context of Traceprobe and Buffer Estimation techniques and latter aggregate the different flows from same IP addresses' pair to have similar path conditions.

The implementation of the delay box using C++ is the second step. The delay box will be similar to the one which has been implemented for ns-2, and allows the delay variability due to buffers to be included easily. It involves the pre-configuration of socket pipes used for communication along with the replay tool.

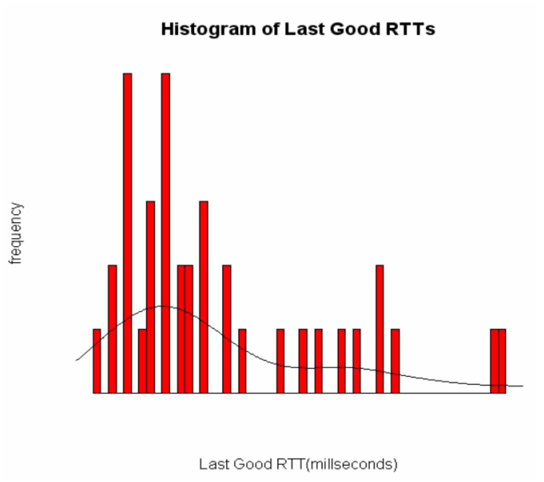
Upon the completion of the delay box implementation, there are a few additional steps to be taken. For validation, the original and emulated traces will be tested with the delay box on the network. Each link on the network will have 100Mb of traffic. When the validation is complete, we will compare the results of the original traces and the ones generated using the delay box.

B. *System Architecture*

The following diagram (Figure 5) presents the basic components of this testbed.



(a) Difference of Last Good RTT and Common RTTs



(b) MultiModality of Buffer Sizes

Fig. 4. Buffer Size Estimation on Traces

1) we investigate some Internet trace files, extracting its basic characteristics using tools developed on this paper (like traceprobe and tracebuffer).
2) we spawn several virtual network stacks, keeping separate the IP queueing and CPU/mem such that different flows don't interfere with each other.
3) we configure the testbed based on the constrains derived. The configuration will be done using a user-level library Trickle [6] that cap the transmission rate of applications on a per-flow basis.
4) the middle machine (based on a network emulator) needs the queueing delay distribution to be generated according to the TCP overflow behavior (buffer size estimation).
5) Finally, our system will require the separation of application delays and TCP overflow delays. Our preliminar idea is to transform this problem, into a signal theory problem, and therefore by performing the deconvolution of the overall delay signal (the interarrival/interdeparture times) extracted per flow from the estimated buffer size delay. Such that, we could emulate a meaningfull thinking time and service time on our environment.

At this time, we have completed the implementation of the infra-structure and we expect to start validating the testbed in the future.

1) Similarity Metrics: In terms of validation, it's important to note that additionally to normal interest variables such as single flow throughput and aggregate throughput. We could compare emulated and original traces by re-analysing the path characteristics from both. Another, stronger statistical test includes a non-parametric goodness-of-fit test, based on the Kullback-Leibler distance. Such statistical method is usually used to evaluate the similarity of the interarrival and interdeparture distributions that result from the original trace and the replayable trace. The idea is to show that for the majority of enterprise traces, there is no statistically significant difference between the original distributions and the emulated distribution.

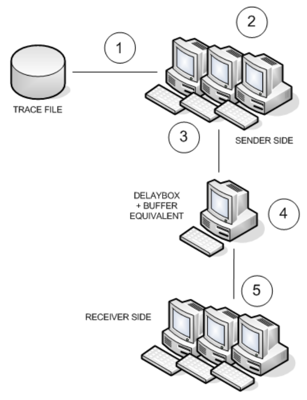
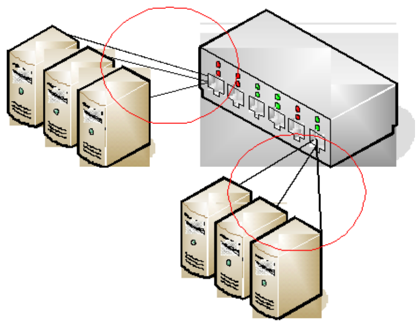


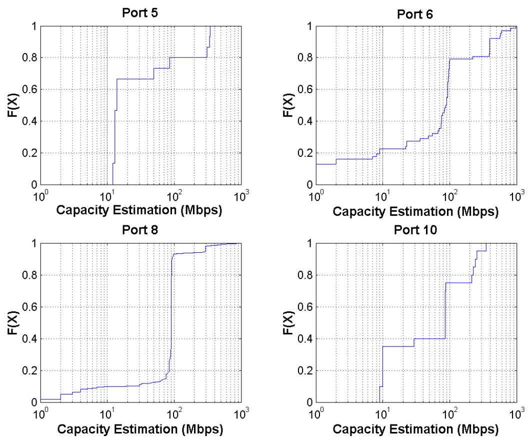
Fig. 5. Reproducible Trace System

C. Learning Population Structure by Narrow Down Per Router Port

Another interesting application of the passive capacity estimation of these flows was that we could infer some aspects of the LBNL network organization, since passive capacity estimation is a pure pre-trace method, and since the monitoring and trace was done on a router at LBNL. Here, we grouped the population in terms of server capacity behind every port on the monitor switch. And we discover that the machines' capacity is not evenly distributed over the ports. Therefore, behind every port, there is either a population of updated machines or outdated ones. For example, Router Port 8 have faster servers with 100Mbps NICs, in comparison to Router Port 5 that has 70% of the machines as 10 Mbps(Figure 6). This application has network infra-structure planning and security implications, since it can be used to learn the network topology and capillarity (end-to-end capacities between IP-addresses) as a whole.



(a) Narrow Down Per Port



(b) Cumulative Distribution Per Port

Fig. 6. Per Port Estimation

VI. Conclusions

We present several trace analysis tools and validation of the tools that can extract accurately path characteristics conditions from stored TCP flows on traces. The path characteristics include a non-exhaustive list of end-to-end path capacities from both directions, the buffer size, minimum delay associated with propagation delay, etc. The tools, called Traceprobe and Tracebuffer, can be derived reliably the estimation that recover the exact conditions on which the underlying TCP flows experience on the time of the trace. We present an evaluation of the tools using both side-by-side comparison with other state of the art tools and simulations. We also describe in great detail a new testbed infra-structure that can replay the dynamics of the stored TCP flows. We intend in a short time to pursue the evaluation and validation of original traces and emulated traces using such deployed testbed.

Acknowledgment

The authors would like to thank Anil Kapur (UCLA) for providing scripts to extract buffer sizes from traces. This research was supported by grants NEC System Platforms Research Laboratories #4-522511-SY-57454 and FAPES (Fundacao de Apoio a Ciencia e Tecnologia do Esprito Santo) with grants of Telecardio Project #31024866/2005.

References

[1] Yu-Chung Cheng, Urs Hlzle, Neal Cardwell, Stefan Savage, and Geoffrey M. Voelker. Monkey see, monkey do: A tool for tcp tracing and replaying. *USENIX 2004 Annual Technical Conference*, 2004.
[2] Mark Claypool, Robert Kinicki, Mingzhe Li, James Nichols, and Huahui Wu. Inferring queue sizes in access networks by active measurement. *Lecture Notes in Computer Science, Volume 3015, Jan 2004, Pages 227 236.*
[3] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages no.pp.905–914 vol.2, 2001.
[4] Michael Dyrna. Network tomography tools. *Institut Eurecom Sophia-Antipolis Master's Thesis 2005.*
[5] T. En-Najjary. Pprate - matlab code. *http://www.eurecom.fr/ ennajjar/PPrate.html*.
[6] Marius A. Eriksen. Trickle: A userland bandwidth shaper for unix-like systems. *In Proceedings of USENIX 2005 Annual Technical Conference, FREENIX Track, pp. 6170.*
[7] Masaki Hirabaru. Impact of bottleneck queue size on tcp protocols and its measurement. *IEICE TRANS COMMUN. OL. E89-B, No. 1 JANUARY 2006. Special Section/Issue on New Technologies and their Applications of the Internet III.*
[8] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance. *IETF RFC 1323*, 1992.
[9] S. Jaiswa, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring tcp connection characteristics through passive measurements. *IEEE INFOCOM 2004 Conference on Computer Communications.*
[10] Rohit Kapoor, Ling-Jyh Chen, Li Lao, Mario Gerla, and M. Y. Sanadidi. Capprobe: A simple and accurate capacity estimation technique. *ACM SIGCOMM '04*, 2004.
[11] J. Liu and M. Crovella. Using loss pairs to discover network properties. *In Proceedings of IEEE/ACM SIGCOMM Internet Measurement Workshop, San Francisco, CA, Nov. 2001.*
[12] S. McCanne and S. Floyd. The ns manual (formerly ns notes and documentation). *ns-2 simulator http://www.isi.edu/nsnam/ns/.*
[13] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and Brian Tierney. A first look at modern enterprise traffic. *Proceedings of Internet Measurement Conference (IMC05), October, 2005.*
[14] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review, 2006.*
[15] Ravi S. Prasad, Manish Jain, and Constantinos Dovrolis. On the effectiveness of delay-based congestion avoidance. *Second International Workshop on Protocols for Fast Long-Distance Networks PFLDNET 2004.*
[16] G. Urvoy-Keller T. En-Najjary. Pprate: A passive capacity estimation tool. *Proceeding of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON06), Vancouver, Canada, April 2006.*
[17] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of tcp round-trip times. *Proceedings of Passive and Active Measurements (PAM), 2005.*
[18] M.C. Weigle, P. Adurthi, F. Hernndez-Campos, K. Jeffay, , and F.D. Smith. Tmix: A tool for generating realistic application workloads in ns-2. *In ACM SIGCOMM Computer Communication Review (CCR), July 2006, Vol 36, No 3, pp. 67-76.*