VALIDATION OF BUFFERBLOAT_DISSECTOR

ANDREA ARALDO

This work is based on [Ara13] that you can download from http://perso.telecom-paristech.fr/~araldo/pmwiki/uploads/Main/thesis.pdf.

In a local testbed, we compare our measures to the ones obtained by considering the ping RTT, as ground truth. We also change the data rate at which the network interface works in order to understand in what scenario our measurement is more accurate. We repeat the experiment with different values of a parameter (the quadrant size) to evaluate its impact on the accuracy.

1. Testbed description

We send bidirectional TCP traffic between two hosts connected via an Ethernet LAN, as in 1.1. We perform the measurement of the queueing delay of the queue on host A. We start three flows:

- the *monitored TCP flow*: host B sends data segments to host A that replies with the acknowledgements.
- the *cross traffic flow*: host A sends burst of data to B
- the *ICMP flow*: host B sends an ICMP echo requests to host A at 1 Hz that replies with an ICMP echo replies.

Our monitor runs on host B and estimate the queueing delay on A observing the data-to-ack time on the monitored flow, applying the methodology of chapter 2 of [Ara13]. In particular, the monitor infers how much time the acknowledgments (that host A sends on the monitored flow) pass in the local queue. The cross traffic flow is used to periodically form the queue inside A, filling it with bursts of data segments with which the host A acknowledgements in the monitored flow have to compete. Compare fig. 2.1 of [Ara13] with 1.1 to have a better idea of how the methodology is applied in the testbed.

1.1. Crafting traffic.

The monitored and cross traffic flow are artificially created with *Iperf* ([TQD+05]). Iperf is a tool that consists of two softwares: iperf client and iperf server.

To create the monitored flow, we first launch iperf server on host A listening on port 5011. Then we launch the iperf client on host B specifying to open a TCP connection to the port 5011 on host A. In this way, iperf client starts to send data segment to iperf server that replies with acknowledgements.

To create the cross traffic flow we launch iperf server on host B listening on port 5012, then we launch iperf client on host A a series of times, separated by 15 seconds, for 2 seconds each time. iperf client acts to inject traffic at a data rate such to use the entire available bandwidth. Therefore, during the intervals of 2 seconds in which iperf client is active, we recreate the scenario of full link utilization

Date: 05 August 2013.

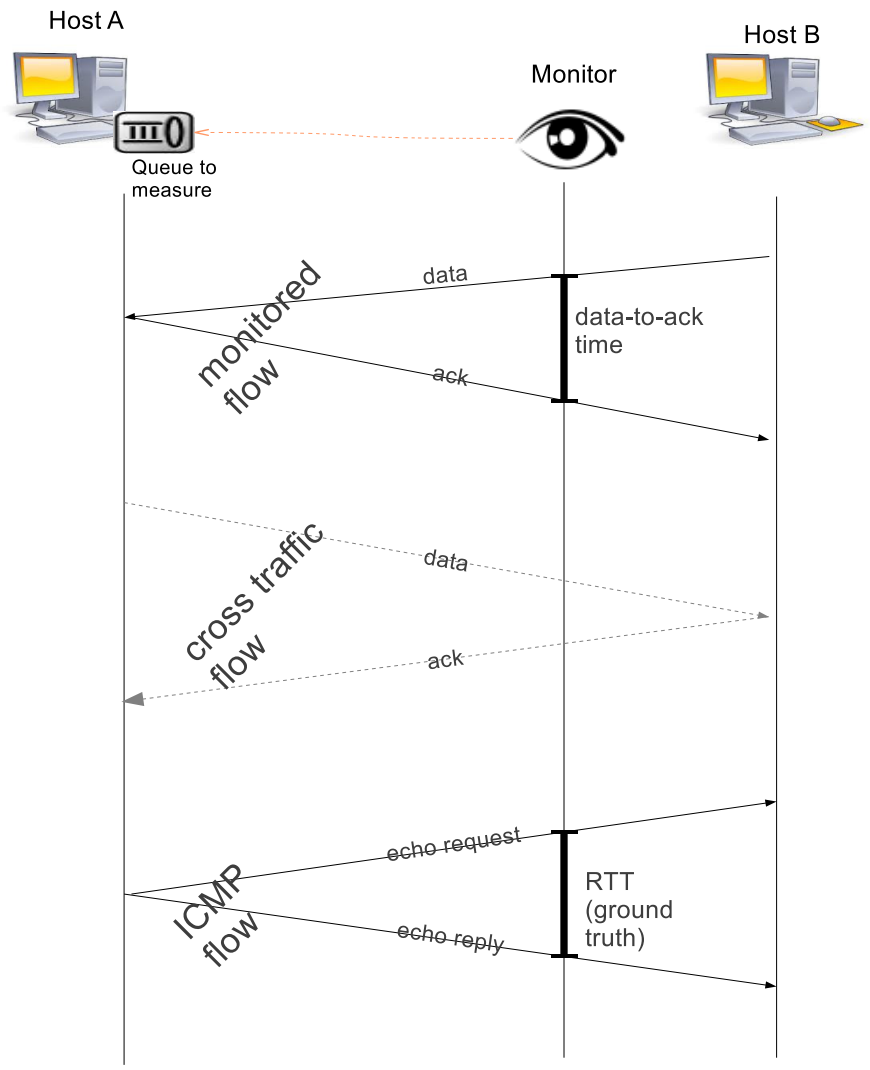

FIGURE 1.1. Local testbed

and we impose a large queue in the exit link of host A, hampering in this way the acknowledgments of host A on the monitored flow.

The ICMP flow is created launching the Linux utility *ping* on host B, setting host A as destination.

1.2. **Creating the bottlenecks.** To create some artificial bottlenecks, we use Ethtool[1] and Netem[2].

[1]See http://en.wikipedia.org/wiki/Ethtool
[2]See http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

Ethtool permits to physically slow down the Ethernet interface[3]. Our hosts are equipped with a 100 Mbps Ethernet interface and we slow down the interface of host A to 10 Mbps.

On the other hand, Netem creates software bottlenecks. We set Netem to use the Hierarchical Token Bucket (HTB) method to implement the bottleneck. HTB is now part of the official Linux Kernel. We use Netem to limit the rate at which host B sends packets to host A in the in the monitored flow. The limit we impose is 7Mbps . In the same way, we impose a limit of 1 Mbps in the data rate at which the host A send packets to host B. Doing this way, we want to simulate the bottleneck represented by the ADSL uplink.

It should be noticed that, to slow down the link exiting from host A, we use both Netem and Ethtool. We do this beacuse during our experiments we observed that slowing down the 100Mbps ethernet interface only via Netem may lead to an instable evolution of the RTT we use as ground thruth (as can be seen in 1.2 - for now observe only RTT (green line) and ignore the rest). In particular, the RTT goes up and down very rapidly. We suppose that this is a consequence of the token bucket mechanism: when there are free tokens, they are rapidly consumed by iperf client that can send a lot of packets and fill the queue, thus imposing high round trip times[2]. In the time intervals when there are no more tokens, the packets of iperf client cannot go out and have to wait until they are recreated. In these intervals the queue has the chance to partially empty and the round trip times decrease. 1.2 shows also that when Netem is used in conjunction with Ethtool, the RTT evolution is more natural, because the Ethernet interface physically works at 10 Mbps and Netem has to slow down the link only by a factor of 10 (instead of slowing down by a factor of 100 like when using only Netem).

Because we want to avoid these irregular and unnatural behaviors, we choose to use Netem and Ethtool together.

1.3 completes 1.1 with some more details about the tools that we use.

1.3. **Replicating the experiments.** The experiments that we propose for the validation are easily reproducible. They are automated by means of distributed bash scripts. To have an idea of what the scripts are and what process are launched, see 1.4.

Two identical version of our Tstat implementation are present on both host A and host B. Some preliminary configuration variable may be set up[4]

Then, launching `ping_validation/exp2-brain-iperf.sh`, the entire experiment is performed. This script is the "orchestrator" and is responsible to launch all the softwares we need. It runs in host A but is able also to launch processes in host

[3]On some network interfaces it may not work

[4]In `ping_validation/exp2-brain-iperf.sh` on the host A side and in `ping_validation/exp2-ping-netbook.plot.sh` and `ping_validation/exp2-netbook-variables_conf.sh` on the host B side. The word "netbook" that appears in the names of the host B side scripts is not meaningful. It appears only because in our experiment we originally used a netbook as host B.

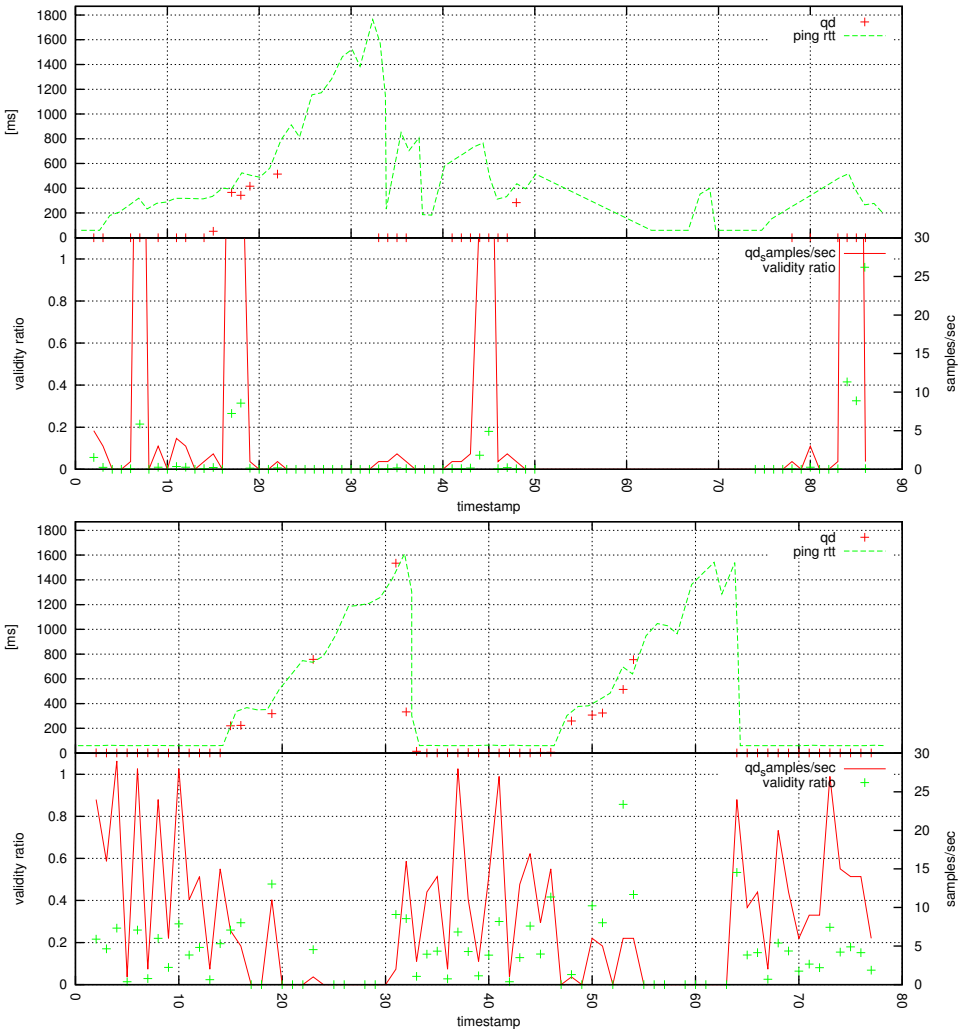

FIGURE 1.2. (top) RTT (green line) when only Netem is the host A bottleneck. (bottom) RTT (green line) when Netem is used in conjunction to Ethtool.

B[5]. This script also calls a script on host B, `ping_validation/exp2-ping-netbook.plot.sh`, that is delegated to launch other processes on host B.

[5]Some network settings are needed on both host A and host B:

- `/etc/hosts` file must be edited in host A, to associate the IP address of host B to the name "netbook"
- `/etc/hosts` file must be edited in host B, to associate the IP address of host A to the name "desktop"
- from host A it must be possible to access via SSH the host B as root user without password. A good howto about this is: http://www.linuxproblem.org/art_9.html
- `ping_validation/exp2-brain-iperf.sh` must be launched with root privileges

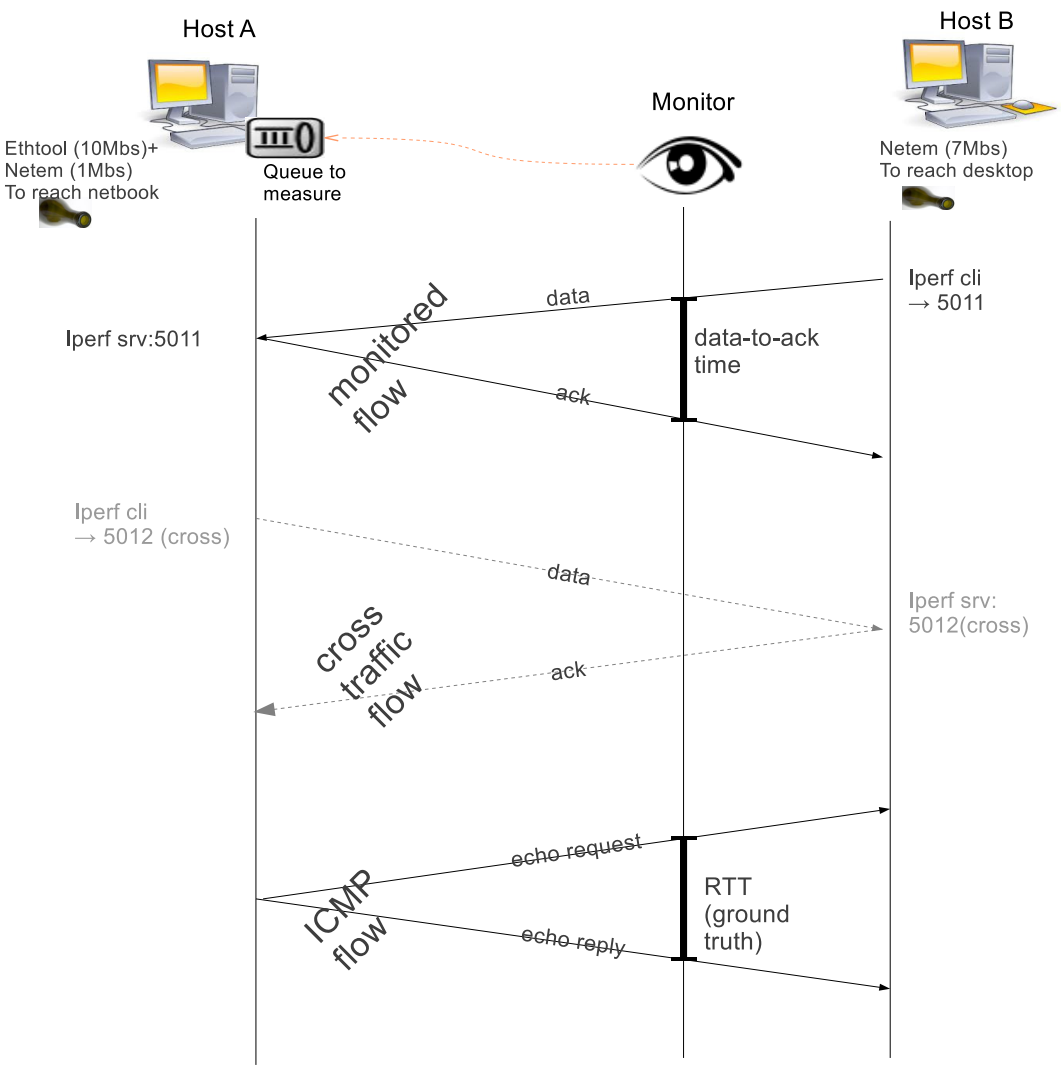

FIGURE 1.3. Local testbed: detailed schema

First, the bottlenecks are created on host A and host B. Then, the iperf servers start (both the one for the monitored flow and the one for the cross traffic flow) to listen on their respective ports. On host B, tcpdump starts to sniff the monitored flow. At the end of the experiment, it will produce a pcap trace[6], useful for further analysis of the experiment. Then host B starts to ping host A. The round trip times are recorded in the ping output file, together with the relative timestamps [7]. Finally, the monitored and the cross traffic flows start.

[6]The file name is specified by the variable `PCAP_TRACE` of `ping_validation/exp2-netbook-variables_conf.sh`

[7]The file name is specified by the variable `PING_OUT_FILE` of `ping_validation/exp2-netbook-variables_conf.sh` .

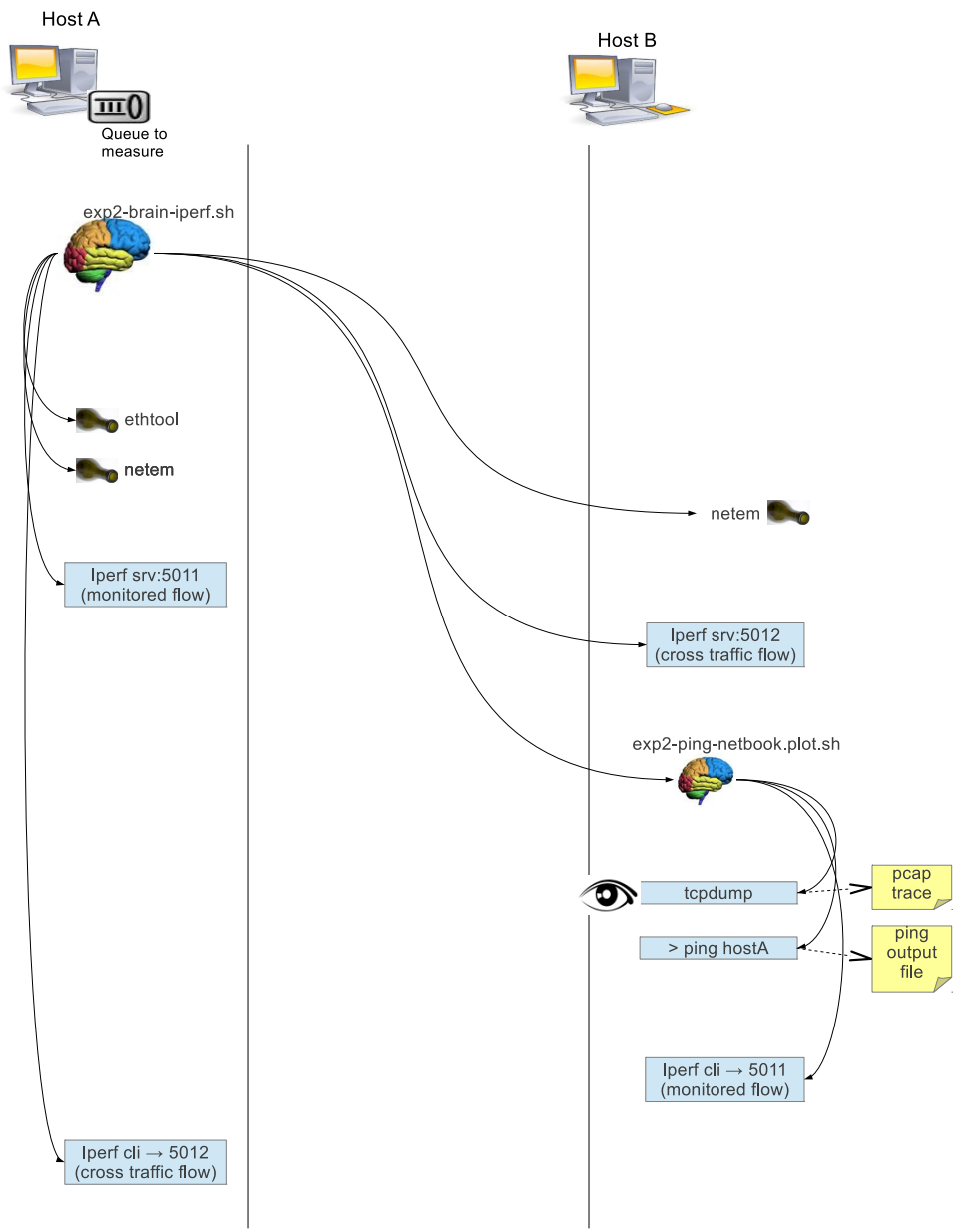

FIGURE 1.4. Testbed automating scripts

1.4. **Analyzing the results.** After running the experiment, it's possible to obtain plots like 1.2 using the the pcacp trace and the ping output file produced with the previous procedure. Launching `ping_validation/exp2-netbook-offline-plotting.bash` Tstat analyzes the pcap trace producing its log files[8]. At the end of the process, a

[8]All the log files written by Tstat will be in the folder specified by the variable `TSTAT_OUT_FOLDER` of `ping_validation/exp2-netbook-variables_conf.sh` .

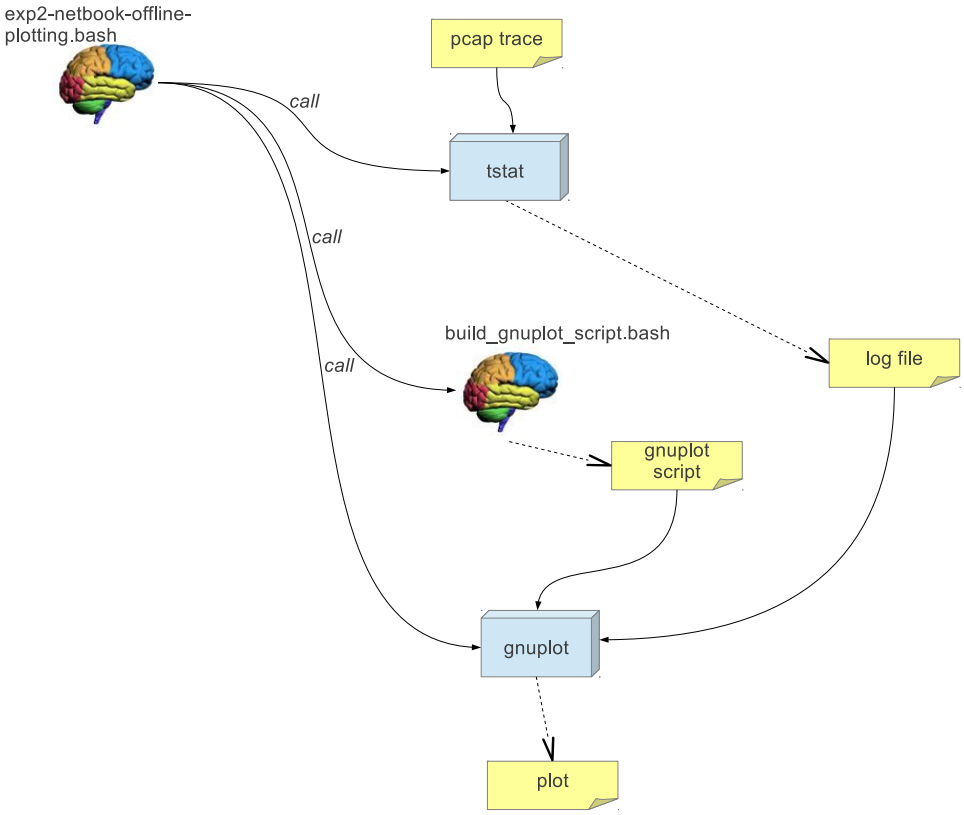

FIGURE 1.5. Offline analysis process

gnuplot script examines the Tstat log file and the ping output script and generate the plots [9]. Separating the experiment runs from the analysis of results permits to compare the results produced by different configurations of Tstat on the same pcap trace.

To have an idea of the data taken as input, the data produced during the analysis and the scripts and sofwares involved, see 1.5.

2. IMPACT OF PARAMETERS AND SURROUNDING CONDITIONS

2.1. **The impact of the quadrant size.** At low level, tcptrace (and thus Tstat too) keeps sequence numbers in a circular data structure named quad (i.e., after the "quadrants" the structure is divided into, to speed-up lookup). In case the quad has a fixed size, it may happen that, if the number of outstanding segments grows larger than the quad size, then sequence numbers are overwritten – so that acknowledgements cannot be paired with data and RTT samples are lost. We show an occurrence of this problem in 2.1, where we configured two values of the quad size[10] – one is fixed (to a purposely small value) and the other is variable and can

[9] An eps file is generated. Its path is specified in `ping_validation/build_gnuplot_script.bash` .

[10] The quadrant size is represented by `MAX_SEG_PER_QUAD` in `tstat/param.h`

grow arbitrarily large to avoid overwriting outstanding sequence numbers (notice that variable size is handled with linked lists, so that in this validation phase we take precisely the opposite direction to [GCCK13], as we do not want to compromise accuracy).

2.1 and 2.2 represent three measurements on exactly the same traffic trace[11]. Each measurement correspond to a configuration of Tstat. For each of the three configurations, we provide three plots: in the top plot we compare the ground truth (the RTT of the ICMP flow) with our estimation. In the bottom plot we represent the number of acknowledgments that host A sends to host B every second in the monitored flow and the validity ratio, i.e. the ratio between the number of valid acknowledgements observed in a second (see section 2.2 of [Ara13]) and the total number of acknoledgements (either valid or invalid). In the middle plot, we represent how many queueing delay samples Tstat is able to calculate. These samples are the ones that Tstat uses to calculate the aggregated queueing delay, every second (see section 2.5 of [Ara13]).

First, we easily observe that when the cross traffic is inserted, it saturates the link from host A to host B. Therefore, the queueing delay increases, as confirmed by the ground truth. As expected, the number of acknowledgements that succeed in arriving to host B decreases and thus Tstat has less information to infer the queueing delay, as confirmed by the decreasing of the number of queueing delay samples per second. Nevertheless, the qd_samples/sec value does not depend only on the number of acknowledgements but also on the quadrant size. From 2.1, it is clear that with a small quadrant size the number of qd_samples/sec that can be calculated is very small for sequence numbers are possibly overwritten. Observing the validity ratio plot, we can grasp why it happens: with a small quadrant size, most of the acknowledgements are marked as invalid, and cannot be used for our estimation. For the configuration `MAX_SEG_PER_QUAD`=2, the validity ratio never exceeds 0.25, while for the configuration `MAX_SEG_PER_QUAD`=INFTY it reaches 1.0 .

As a consequence, for small values of the quadrant size, Tstat estimation is very coarse, while, increasing the quadrant size, Tstat better approaches the ground truth (in 2.1, compare the top plot of the two configurations, in particular after second 55). We point out that, when quadrant size is small, the error is not "random" but systematically may lead to an underestimation of the queueing delay: in particular, during our experiments we observed that when the ICMP RTT is not so high Tstat gives a good estimation but when the RTT is high Tstat cannot produce its estimates. Providing estimates of the queueing delay only when it is low clearly is a measurement bias.

The 2.2 represents an intermediate case: the validity ratio is slightly lower than with `MAX_SEG_PER_QUAD`=INFTY, and so the number of queueing delay samples per seconds; nonetheless Tstat succeeds in estimating the queueing delay.

The results that are presented in the following chapter are obtained with `MAX_SEG_PER_QUAD`=INFTY. The considerations that we've given in this section highlight that to infer the queueing delay (like when performing any other measurement), attention should be paid in tweaking the measuring tool, otherwise the results may be distorted. We point out that other authors do not report much on tweaking the tool they use.

[11]We sniff the traffic with tcpdump and then apply the measurement on the offline trace

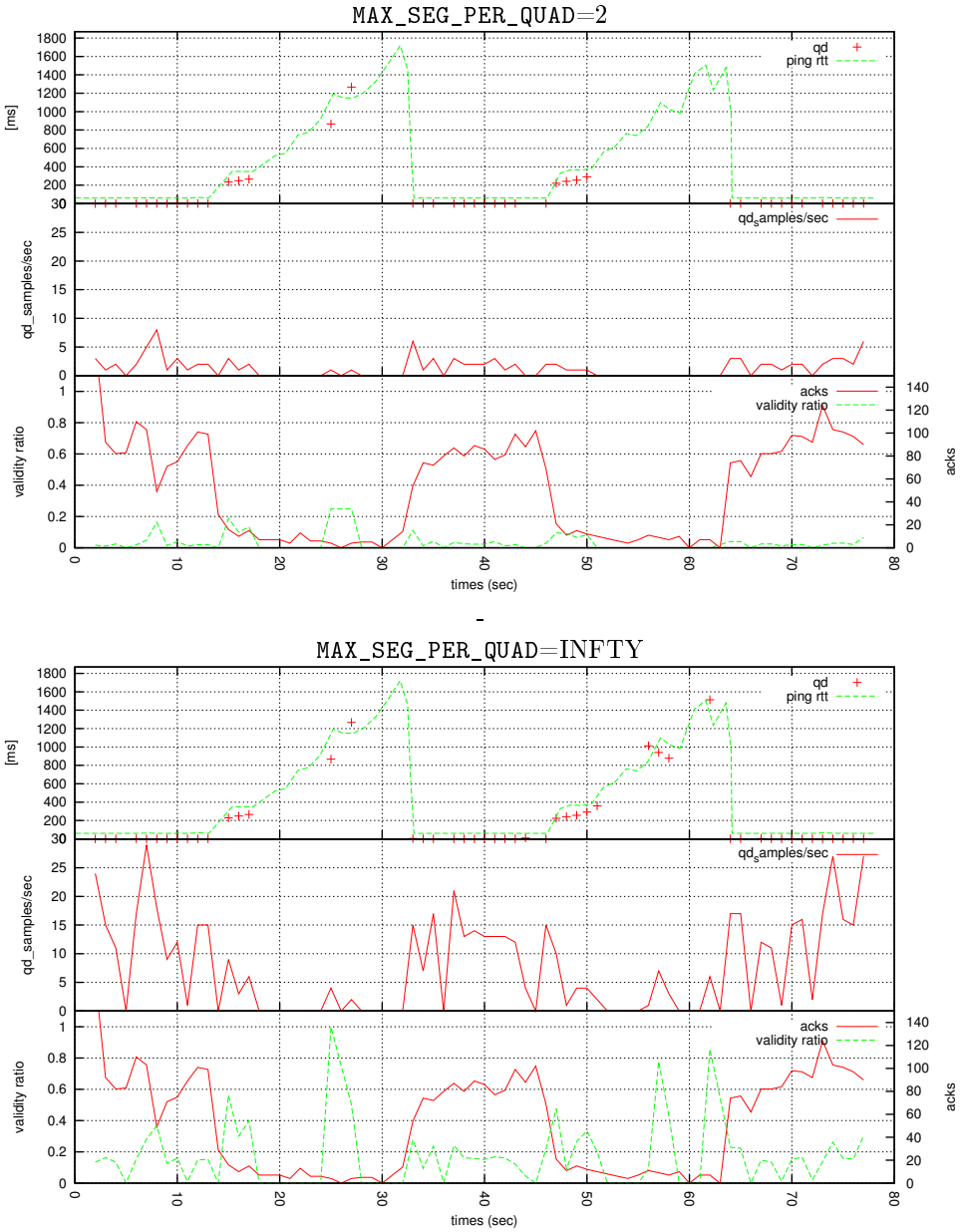

FIGURE 2.1. Queueing delay estimation using different values of quadrant size. `MAX_SEG_PER_QUAD`=2 in the top figure and `MAX_SEG_PER_QUAD` is INFTY in the bottom figure (INFTY means that Tstat keeps track of all the segments: the only limit is the system meory).

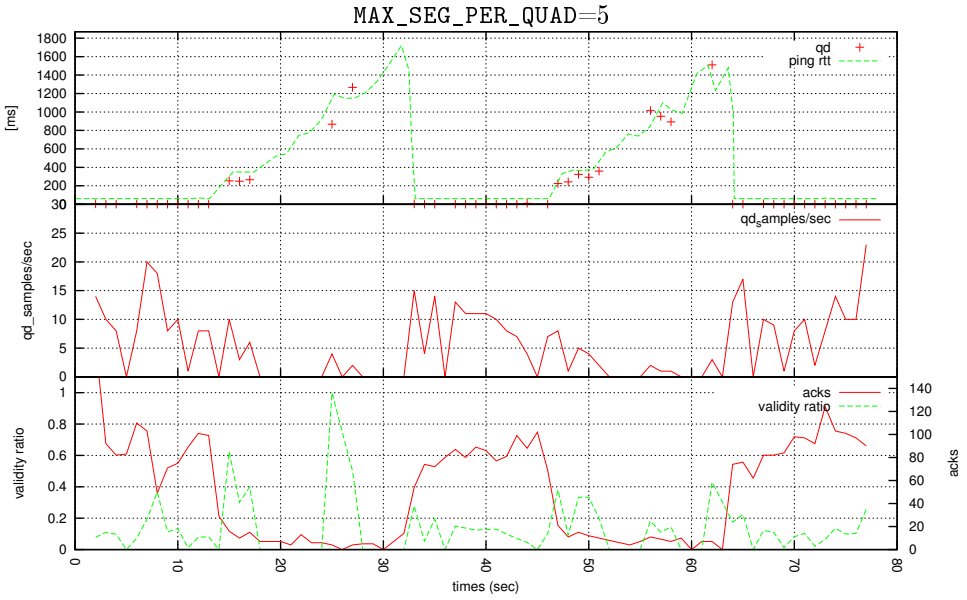

FIGURE 2.2. Queueing delay estimation with `MAX_SEG_PER_QUAD`=5

2.2. **Dependency on the data rate.** In this subsection, we show that the expected data rate should be taken into account when tweaking Tstat. For the data rates that we have when using the bottlenecks (Netem and Ethtool), a `MAX_SEG_PER_QUAD`=5 is enough to have a good estimation of the queueing delay. Now, we provide the results obtained replicating the experiments of 1 without any bottleneck (neither in host A nor in host B, neither Netem nor Ethtool). Both the monitored flow and the cross traffic flow are free to run at the maximum speed.

2.3, 2.4 and 2.5 show that the number of acknowledgements in the monitored flow is more than a magnitude larger than the previous experiments, but the validity ratio is almost 0 even with MAX_SEG_PER_QUAD=15 and thus the number of queueing delay samples per second is not sufficient. In the previous experiments, with
MAX_SEG_PER_QUAD=2 the estimation was coarse but at least possible. Now, with that configuration, Tstat is not able to provide any estimation. While in the previous example MAX_SEG_PER_QUAD=5 guaranteed a good estimation, now the estimation is very coarse. To have a good estimation, we have to set MAX_SEG_PER_QUAD=15.

3. PERFORMANCE ANALYSIS AND ANALYSIS OF THE OVERHEAD

In this section, we coarsely analyze the overhead imposed on Tstat by the queueing delay estimation both in terms of required disk space for the output files and of execution time.

We run Tstat on a set of 59 compressed pcap traces. The total size of the traces is 43 GB. With bufferbloat analysis disabled, Tstat running time is 23 minutes and the output log files are 1,462GB. Enabling bufferbloat analysis, the log files increase

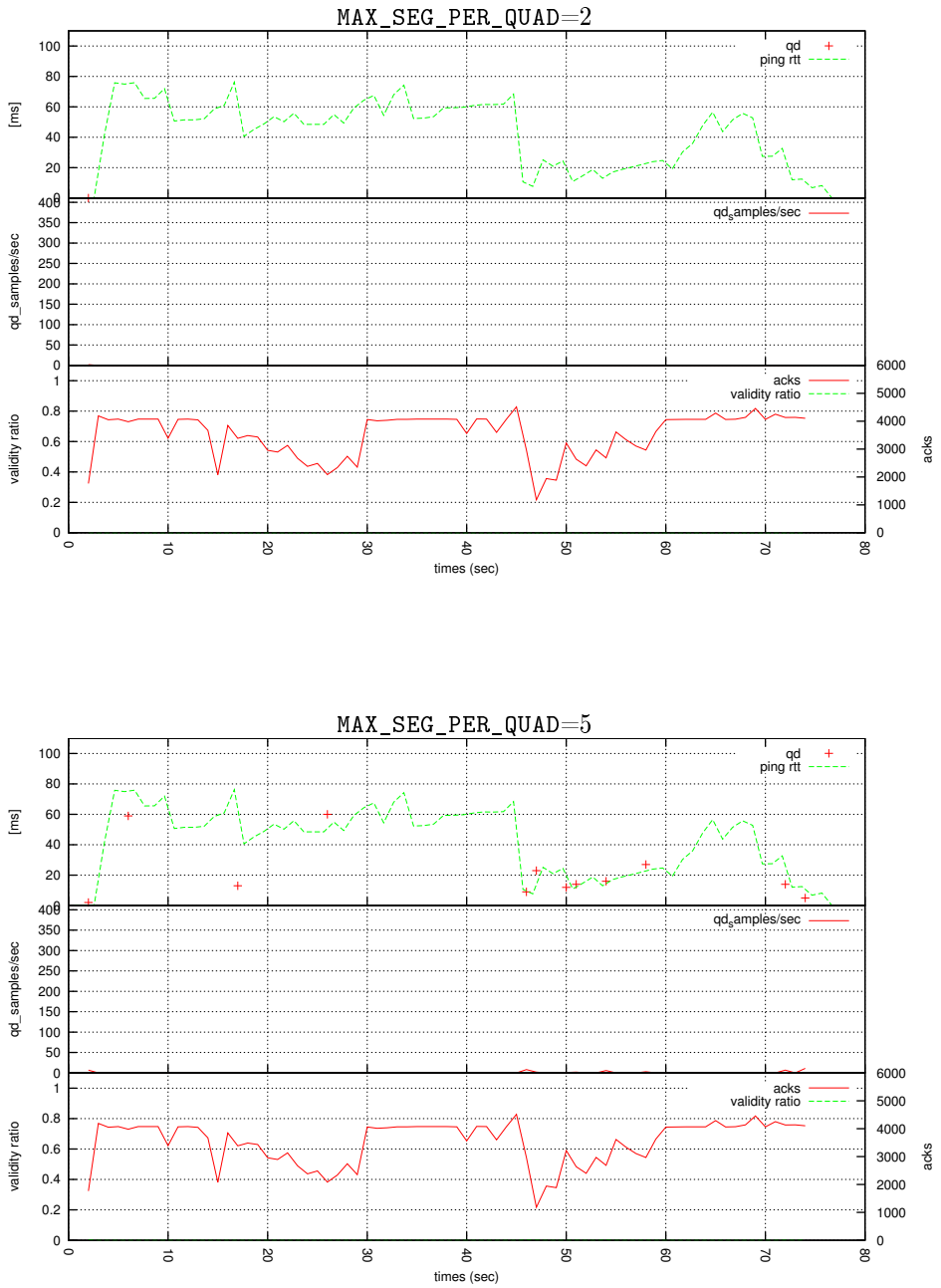

FIGURE 2.3. Experiment without bottlenecks

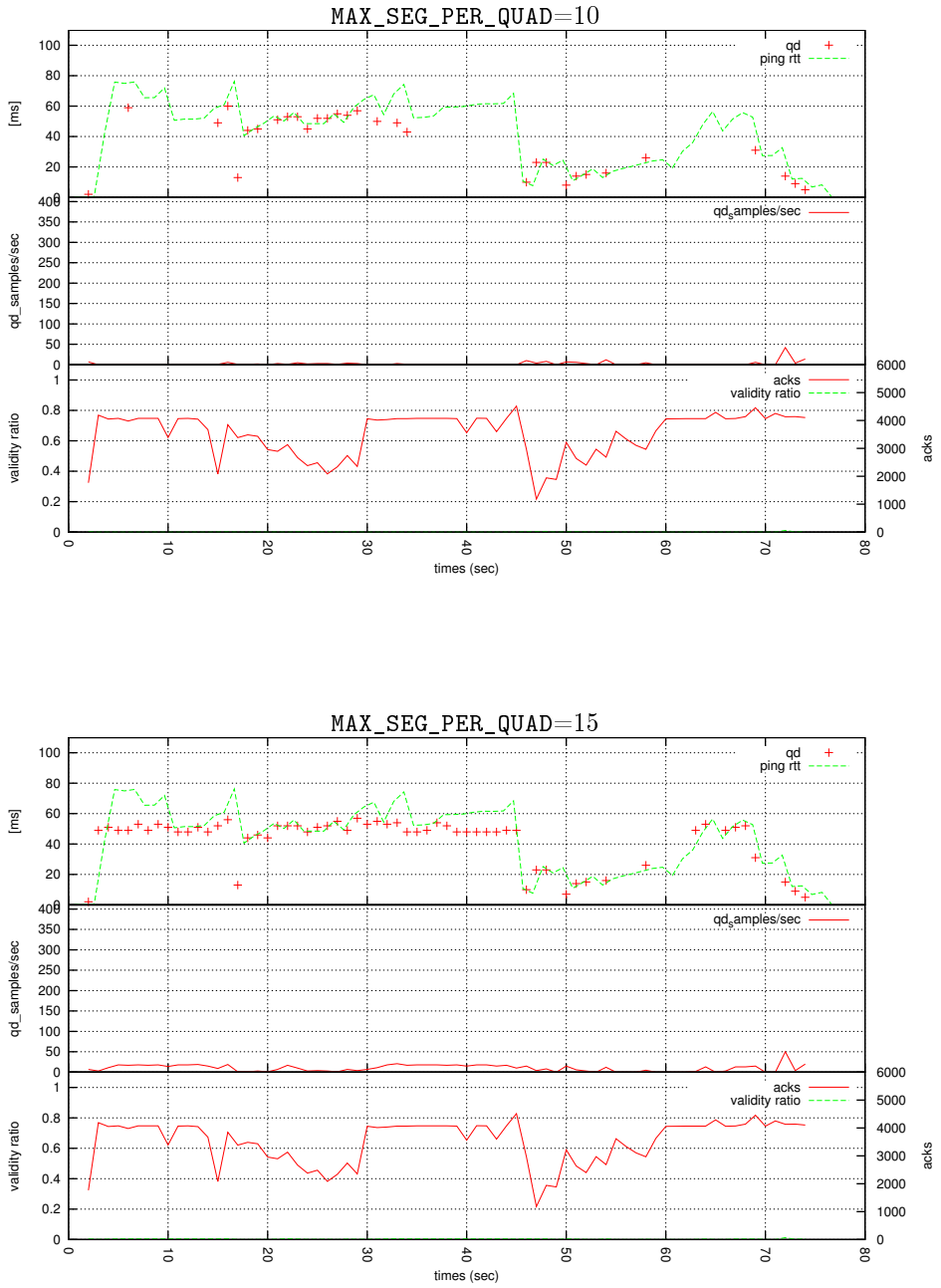

FIGURE 2.4. Experiment without bottlenecks

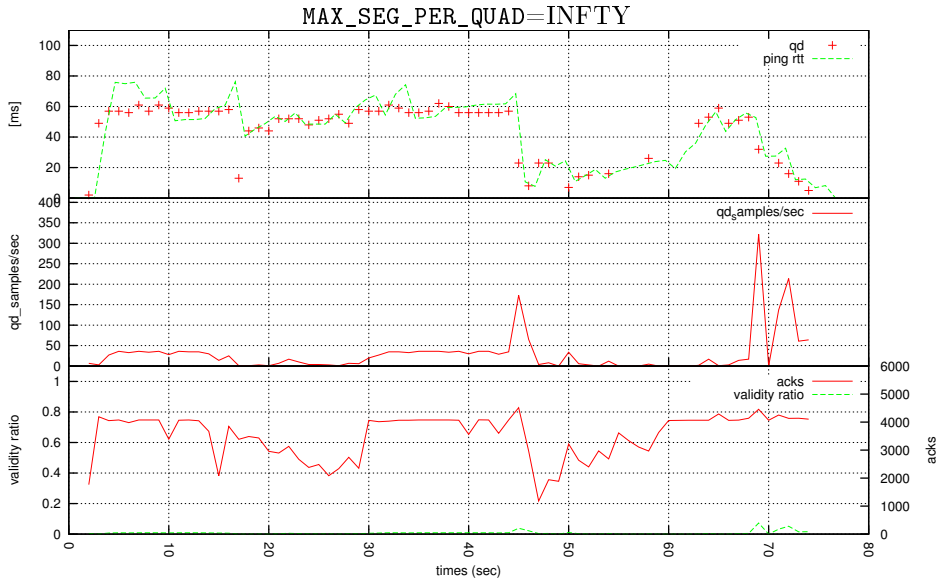

FIGURE 2.5. Experiment without bottlenecks

to 2,328 GB and running time increases to 25 minutes. We can conclude that the bufferbloat analysis imposes an overhead of 60% in terms of required space on the disk and of 8% in terms of running time.

REFERENCES

[Ara13] A. Araldo. Passive analysis of queueing delay in the Internet. Tesi di laurea magistrale, Universita' di Catania, 2012/13.

[GCCK13] Sriharsha Gangam, Jaideep Chandrashekar, Ítalo Cunha, and Jim Kurose. Estimating tcp latency approximately with passive measurements. In *Passive and Active Measurement*, pages 83–93. Springer, 2013.

[TQD+05] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005.