Remotely gauging upstream bufferbloat delays

C. Chirichella[1], D. Rossi[1], C. Testa[1], T. Friedman[2], A. Pescape[3]

[1]Telecom ParisTech - first.last@enst.fr, [2]UPMC Sorbonne Universites - timur.friedman@upmc.fr, [3]Univ. Federico II - pescape@unina.it

Abstract. "Bufferbloat" is the growth in buffer size that has led Internet delays to occasionally exceed the light propagation delay from the Earth to the Moon. Manufacturers have built in large buffers to prevent losses on Wi-Fi, cable and ADSL links. But the combination of some links' limited bandwidth with TCP's tendency to saturate that bandwidth results in excessive queuing delays. In response, new congestion control protocols such as BitTorrent's uTP/LEDBAT aim at explicitly limiting the delay that they add over the bottleneck link. This work proposes and validate a methodology to *monitor the upstream queuing delay experienced by remote hosts*, both those using LEDBAT, through LEDBAT's native one-way delay measurements, and those using TCP (via the Timestamp Option).

1 Problem statement

As a recent *CACM* article points out, "Internet delays now are as common as they are maddening" [3]. Currently, the combination of excessive buffer sizes (aka *bufferbloat*), with TCP's congestion control mechanism (which forces a bottleneck buffer to fill and generate a loss before the sender reduces its rate), queuing delays can potentially reach a few seconds [8]. This is confirmed by recent studies such as [5], showing that most home gateways have a fixed buffer size, irrespective of the uplink capacity. With cable and ADSL modem buffers ranging from, on average, 120 KB to a maximum of 365 KB [5], and common uplink rates of 1 Mbps, worst case queuing delays can range from 1 second on average to a maximum of 3 seconds.

To counter this problem, BitTorrent developers have proposed IETF LEDBAT [9] as a TCP replacement for data transfer. Like TCP, LEDBAT maintains a congestion window – but whereas mainstream TCP variants use loss-based congestion control (growing with ACKs and shrinking with losses), LEDBAT estimates the queuing delay on the bottleneck link and tunes the window size in an effort to achieve a target level of queuing delay (100ms by default). By explicitly capping the queuing delay, LEDBAT aims at protecting VoIP [2] and other interactive traffic (e.g., Web, Gaming) by congestion self-induced by other traffic of the same user.

Although TCP's loss-based congestion control, coupled with large buffers, can clearly cause significant bufferbloat delays, it is unclear how often this happens in practice, and how badly it hurts user performance. Indeed, active approaches such as Netalyzer [8], likely overestimate bufferbloat delay – by purposely filling the pipe, Netalyzer learns the *maximum* bufferbloat delay, but not its *typical* range. To counter this limitation, we design and validate a passive methodology for inferring the queuing delays encountered by remote LEDBAT and TCP hosts.

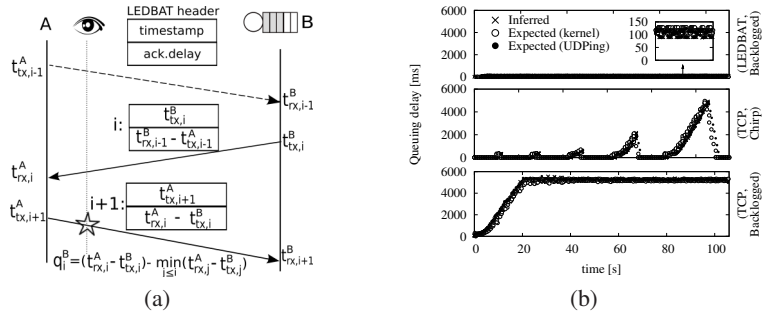

Fig. 1. Bufferbloat measurement methodology: illustration (left) and validation (right).

2 Methodology

We estimate queuing delay by collecting one-way delay (OWD) samples, establishing the minimum as a baseline delay, and then measuring the degree to which a sample differs from the baseline. This is a classic approach used in congestion control to drive congestion window dynamics since the late 1980s [7]. Our innovation is to demonstrate how a passive observer of LEDBAT or TCP traffic can use this approach to estimate the uplink delays experienced by a remote host.

To infer B queues, in our methodology an observer close to A sniffs the packets and performs the same state updates as does the LEDBAT congestion control protocol running on A. Our methodology is to sniff and inspect LEDBAT and TCP packets, and mimick the way the LEDBAT sender computes queuing delay based on header fields.

Fig. 1(a) illustrates the methodology. On reception of a new packet, the receiver calculates the OWD as the difference between its own local clock and the sender "timestamp" (the latter extracted from packet header[1]), and sends this "ack.delay" value back to the sender (using another header field). At each packet reception, the observer updates the base delay β_{BA} as the minimum over all OWD $B \to A$ samples:

$$\beta_{BA} = \min(\beta_{BA}, t^A_{\mathrm{rx},i} - t^B_{\mathrm{tx},i}), \tag{1}$$

$$q^B_i = (t^A_{\mathrm{rx},i} - t^B_{\mathrm{tx},i}) - \beta_{BA} \tag{2}$$

Then, the queuing delay q^B_i incurred by packet i can be inferred by subtracting β_{BA} from the timestamp difference carried in packet $i+1$.

Omitted here for lack of space but reported in [4], the methodology also applies to TCP traffic provided that the flow has the Timestamps Option [6] enabled. This means the observer must either be one of the hosts, work in cooperation with one of the hosts, or opportunistically measure only those flows that have this option enabled.

[1] In the absence of a finalized LEDBAT standard, our protocol parser is based on BitTorrent's currently implemented BEP-29 definition [1]

3 Validation

We validate our methodology in Fig. 1(b), reporting testbed results with two ground truths: (i) kernel level queue logs (hacking the `sch_print` function of the `netem` emulator) and (ii) UDP ping-like measurements (as queuing occurs only at B, we have $q_i^B = RTT_i - min_{j \leqslant i} RTT_j$).

Host B has an ongoing backlogged LEDBAT flow to A (top plot), with possibly interfering on/off TCP (middle) or backlogged-TCP (bottom) traffic models. As expected, in the LEDBAT case queuing reaches the 100 ms target specified in the draft (top). In the on/off case, queuing can possibly grow very large depending on the amount of cross TCP traffic (middle). Finally, queuing delay attains the maximum value, that Netalyzer [8] would report, under backlogged TCP (bottom). In all cases, we see that our methodology is very reliable agains both ground truths (differences are on the order of 1 packet worth of queuing delay for LEDBAT).

In a typical scenario, however, the observer O will be able to observe *only part* of the traffic generated by the host of interest B (say, the traffic $B \rightarrow A$), but will miss another part (say, $B \rightarrow C$). Omitted here for lack of space but reported in [4], our validation shows the methodology to be accurate even in case the observer O has only a partial view of B traffic: the error in the inferred measure is negligible in cases where a sizable amount of traffic makes it to the observer, but is still robust and reliable even when the observer is able to sniff only very few samples.

Acknowledgement

This work has been carried out at LINCS `http://www.lincs.fr` and funded by the FP7 mPlane project (grant agreement no. 318627).

References

1. `http://bittorrent.org/beps/bep_0029.html`.
2. ITU Recommendation G.114, One Way Transmission Time.
3. V. Cerf, V. Jacobson, N. Weaver, and J. Gettys. Bufferbloat: what's wrong with the internet? *Communications of the ACM*, 55(2):40–47, 2012.
4. C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape. Passive bufferbloat measurement exploiting transport layer information. `http://www.enst.fr/~drossi/dataset/bufferbloat-methodology/techrep.pdf`, 2012.
5. L. DiCioccio, R. Teixeira, M. Mayl, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *PAM*, 2012.
6. V. Jacobson et al. TCP Extensions for High Performance. IETF RFC 1323, 1992.
7. R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM CCR*, 19(5):56–71, 1989.
8. C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (ACM IMC'10)*, 2010.
9. S. Shalunov et al. Low Extra Delay Background Transport (LEDBAT). IETF draft, 2010.