BUFFERBLOAT_DISSECTOR HOWTO

ANDREA ARALDO

1. Introduction

`bufferbloat_dissector` is a patched version of Tstat. Therefore, to install and run `bufferbloat_dissector`, you have to perform the same steps as when installing and running Tstat. Also the syntax remains the same.

We highly recommend to read the following sections of the "Tstat howto"[1] before reading this document. The sections are:

- "Installation"
- "Usage"
- "Output/Logs"

If you have any problems, if anything is not so clear, if I forgot to say something or I said something wrong don't hesitate to contact me at andrea.araldo@gmail.com . I will thank you for having helped me in improving `bufferbloat_dissector` and in making the life of any other guys who will use it a little easier. I would also be very glad also if you will contact me simply to let me know that you are using `bufferbloat_dissector` and you are having fun (going mad) with it.

andrea.araldo@gmail.com

2. Tstat overview

Tstat [FMM+11] is an open source passive monitoring tool developed by the networking research group at Politecnico di Torino[2] since 2000. It started as an evolution of tcptrace[3] and offers live monitoring (specifying the interface to monitor) and offline analysis (specifying the input file to analyze, e.g. a pcap trace). It is written in ANSI C for efficiency and allows sophisticated multi-gigabit-per-second traffic analysis to be run live using common hardware and Libpcap [JLM94], the de facto standard application programming interface (API) to capture packets.

Tstat sniffs IP packets and aggregate them in *flows*. Each flow is typically defined as the sequence of packets characterized by the same flowID that have been observed in a given time interval, where flowID = (ipaddress1, port1, ipaddress2, port2, direction) so that TCP and UDP flows are considered. The direction indicates if the data go from ipaddress1 to ipaddres2 or in the opposite direction. For this work, only TCP flows are studied. The start of a new flow is commonly identified when the TCP three-way handshake is observed; similarly, its end is triggered when either a proper TCP connection teardown is seen, or no packets have been observed for some time.

Date: 05 August 2013.

[1] http://tstat.tlc.polito.it/HOWTO.shtml

[2] Tstat Homepage: http://tstat.tlc.polito.it

[3] TCPTrace Homepage: http://www.tcptrace.org

By configuration files, it is possible to specify the hosts that are considered as internal or external and the traffic is organized in

- Incoming traffic: The source is external and the destination is internal.
- Outgoing traffic: The source is internal and the destination is external.
- Local traffic: Both source and destination are internal.
- External traffic: Both source and destination are external.

We monitor only outgoing traffic and divide internal and external hosts.

Tstat collects several network-layer as well as transport-layer measurements, which are described in full details in http://tstat.polito.it/measure.shtml. Part of the Tstat output is represented by plain text logs. In the offline analysis (the one that we are performing), for each pcap trace Tstat produces a folder with different log files. Examples of log files are: `log_tcp_complete`, `log_tcp_nocomplete` , `log_udp_complete` , `log_skype_complete`, `log_chat_complete`, `log_streaming_complete`. Their form is quite similar: they are plain text files where each row corresponds to a different flow and each column is associated to a specific measure (for `log_tcp_complete` we have number of data packets, number of SYN messages, maximum receiver window announced, ...).

Tstat is also able to identify the application that generated the traffic. Primarily, Tstat implements a deep packet inspection (DPI) technology.

3. Queueing delay measurement implementation

Bufferbloat-related code. When implementing our methodology in Tstat, our goal was to isolate the code related to the queueing delay calculation from the rest of Tstat. Tstat is now a complex tool able to accomplish different measurement tasks and it's important to maintain it modular: the experimenter should be able to activate and deactivate what he needs for his measurement. We know that not all the experimenters will be interested in queueing delay measures and we want that they should be able to use Tstat for their goals simply ignoring the existence of portions of code related to queueing delay.

All the C procedures related to queueing delay measurement are in `tstat/bufferbloat.h` and `tstat/bufferbloat.c`. In particular, the functions declared here are:

- `bufferbloat_analysis(..)`:
  - called in `tstat/tcp.c`, `tstat/ledbat.c`, `tstat/rexmit.c`
- `chance_is_not_valid(..)`:
  - called in `tstat/tcp.c`
- `check_direction_consistency(..)`:
  - called in `tstat/tcp.c`, `tstat/ledbat.c`, `tstat/rexmit.c`

These function calls are activated only if the precompilation option `BUFFERBLOAT_ANALYSIS` is enabled (see "Configuration parameters").

The code in `tstat/bufferbloat.c` is an evolution of the code developed by Chiara Chirichella [Chi12] to study the bufferbloat by monitoring the uTP flows. We generalize and extend the code to make it work both with uTP monitoring (as in Chirichella's work) and TCP monitoring (as in our work). The calculation method used for the uTP case is very similar to ours. They both estimate the queueing delay collecting the samples of *gross delays*. Gross delay is the general

term with which we indicate the one-way delay in the uTP case and the data-to-acknowledgement time in our TCP case. Therefore, if `bufferbloat_analysis(..)` is called by `tstat/rexmit.c`, the following parameter is passed:

`last_gross_delay = etime_rtt/1000`

where `etime_rtt` is the data-to-acknowledgement time in microseconds.

On the contrary, if `bufferbloat_analysis(..)` is called by `tstat/ledbat.c`, the `last_gross_delay` will be assigned a value depending by `time_diff`, that is the one-way delay.

We inserted comments in the code with the aim to make it self-explanatory and thus we do not dwell on the code description.

Configuration parameters. Our implementation on Tstat is highly configurable by precompilation options[4].

The general options are

- `BUFFERBLOAT_ANALYSIS`: if enabled, the queueing delay estimation will be performed. Otherwise, the "classical" version of Tstat will run
- `DATA_TRIGGERED_BUFFERBLOAT_ANALYSIS`: if enabled, in addition to the queueiif enabledng delay inference method that we are presenting in this work, a different method will run. It is based on the ack-to-the-following-data-segment time (rather then on data-to-ack time).We implement this method but do not analyze it. This method will be ignored in this work.
- `FILTERING`: if enabled, all the LEDBAT-like filtering operations to calculate the baseline will be used.
- `FORCE_CALL_INLINING`: if enabled, gcc compiler will be forced to compile using the optimization technique called "call inlining"[5]: in the compilation phase, the calls to a function are replaced with the body of the function. This generally permits shorter running times, because, at run time, the overhead of the function call (branch, parameter passing, allocation of space in the stack, return parameter handling, ...) is avoided. The disadvantage is the increase of the object code size.

The debug options are:

- `SEVERE_DEBUG`: If it is enabled, a lot of redundant and overabundant checks will be performed to check for inconsistent states or data. If an inconsistent state is detected, Tstat terminates and issues an error message. This can be useful for further editing of the code to be sure that the modifications do not introduce inconsistency.
- `SAMPLE_VALIDITY`: If it is enabled, Tstat will perform the calculations to check how many acknowledgements are considered valid or invalid (see the validation documentation).
- `SAMPLE_BY_SAMPLE_LOG`: If it is enabled, the log files with all the queueing delay samples will be produced (in addition to the log files of the aggregated queueing delays).
- `ONE_FLOW_ONLY`: it can be used when, for debugging purposes, it is necessary to be sure that Tstat is monitoring only one flow.

The default configuration is with only `BUFFERBLOAT_ANALYSIS` and `FORCE_CALL_INLINING` are enabled.

[4]See `tstat/Makefile.conf`

[5]See http://www.greenend.org.uk/rjk/tech/inline.html for a good howto on the question

4. Output log file

In addition to the other Tstat log files, the following files will be produced[6]:

- `log_tcp_windowed_qd_acktrig`: if `BUFFERBLOAT_ANALYSIS` is enabled
- `log_ledbat_windowed_qd`: if `BUFFERBLOAT_ANALYSIS` is enabled
- `log_tcp_windowed_qd_datatrig`: if `BUFFERBLOAT_ANALYSIS` and `DATA_TRIGGERED` are enabled
- `log_ledbat_qd_sample`: if `BUFFERBLOAT_ANALYSIS` and `SAMPLE_BY_SAMPLE_LOG` are enabled
- `log_tcp_qd_sample_acktrig`: if `BUFFERBLOAT_ANALYSIS` and `SAMPLE_BY_SAMPLE_LOG` are enabled
- `log_tcp_qd_sample_datatrig`: if `BUFFERBLOAT_ANALYSIS`, `SAMPLE_BY_SAMPLE_LOG` and `DATA_TRIGGERED` are enabled

For the measures that we present here, only `log_tcp_windowed_qd_acktrig` is used.

Each row of this file describes a 1-second-window of a certain flow pair characterized by the same (ipaddress1, port1, ipaddress2, port2)-tuple. Considering the role of the host that sends the packet, for each tuple like above Tstat distinguishes between client and server, i.e., host that opens a connection and and host that replies to connection request. The role of client or server does not correspond to the classification in internal or external host.

The columns of the log file are as follows:

(1) `timestamp` (in seconds)
(2) `ip_addr_1`: IP address of the client
(3) `port_1`: port of the client
(4) `ip_addr_2`: port of the server
(5) `port_2`: port of the server
(6) `internal_src`: 1 if the host identified as client is internal; 0 otherwise[7]
(7) `internal_dst`: 1 if the host identified as server is internal; 0 otherwise
(8) `connection_type` (client-to-server direction[8]): a string in the form `<con_type>:<p2p_type>`. `<con_type>` can be P2P, HTTP, SMTP, ... (see `tstat/protocol.h` for the complete list). If the connection is of type P2P, `<p2p_type>` indicates more detail about the type of connection. In our work, we use only the information in `<con_type>`.
(9) `aggregated_queueing_delay` (client-to-server direction): in milliseconds
(10) `window_error` (client-to-server direction): in milliseconds
(11) `qd_max_w1` (client-to-server direction): in milliseconds, it indicates the maximum of the queueing delays collected in the last 1-second-window

[6]Notice that the term "windowed" stands for "aggregated"

[7]For more details, search `tstat/struct.h` for "internal_src"

[8]Actually, this is a redundant information beacause the two directions are always of the same type.

(12) `chances_in_win` (client-to-server direction): how many acknowledgements (either valid or invalid) are collected in the 1-second-window. Each acknowledgment is considered as a "chance"[9] to calculate the queueing delay. When `SAMPLE_VALIDITY` is disabled, this column is always equal to "-".
(13) `aggregated_grossdelay` (client-to-server direction): this is the average data-to-acknowledgement time (milliseconds) of all the samples collected in the 1-second-window.
(14) `connection_id` (client-to-server direction): this is meaningful only in the LEDBAT contest.
(15) `samples_in_win` (client-to-server direction): number of queueing delay samples collected in the 1-second window. The aggregated queueing delay is calculated on the base of these samples.
(16) `not_void_windows` (client-to-server direction): meaningless, used only for debugging purposes
(17) `qd_measured_sum` (client-to-server direction): in milliseconds, it is the sum of the queueing delay samples seen from the beginning of the flow
(18) `aggregated_qd_sum` (client-to-server direction): in milliseconds, it is the sum of the aggregated queueing delay of the 1-second-windows seen from the beginning of the flow.
(19) `sample_qd_sum_until_last_window` (client-to-server direction): in milliseconds, it is the sum of all the queueing delay samples from the beginning of the flow to the last closed 1-second window.
(20) `baseline` (client-to-server direction): in milliseconds: it is the last baseline calculated in the 1-second window. Notice that this value is not used for the calculation but only written here for debugging purposes. Indeed, the baseline used for the queueing delay calculation is recomputed at every sample.
(21) `connection_type` (server-to-client direction)
(22) `aggregated_queueing_delay` (server-to-client direction)
(23) `window_error` (server-to-client direction)
(24) `qd_max_w1` (server-to-client direction)
(25) `chances_in_win` (server-to-client direction)
(26) `aggregated_grossdelay` (server-to-client direction)
(27) `connection_id` (server-to-client direction)
(28) `samples_in_win` (server-to-client direction)
(29) `not_void_windows` (server-to-client direction)
(30) `qd_measured_sum` (server-to-client direction)
(31) `aggregated_qd_sum` (server-to-client direction)
(32) `sample_qd_sum_until_last_window` (server-to-client direction)
(33) `baseline` (server-to-client direction)

The procedures responsible for writing the log file are `print_last_window_general(..)` (it writes the first five columns of each row) and `print_last_window_directional(..)` (it writes all the other columns, starting from the ones related to the client-to-server direction and then writing the columns related to the server-to-client direction).

[9]The log file `log_tcp_windowed_qd_datatrig` (that we do not use) has the same form presented here. In that contest, every data segment is considered as a chance to calculate the queueing delay. This is why we do not call this column "number_of_acks" and prefer the more general name "chance"

Both the procedures are in `tstat/bufferbloat.h` and `tstat/bufferbloat.c`. They are used to produce not only `log_tcp_windowed_qd_acktrig`, but also `log_ledbat_windowed_qd` and `log_tcp_windowed_qd_datatrig`. This is why, for the sake of generality, some columns are meaningless in our contest and some column names may seem too abstract.

5. Post-processing

All the results are obtained by means of calculations that have the log file described above as input. The script files that implement these calculations are in the folders `offline_analysis`, `performance_evaluation` and `ping_validation`.

Most of them are Linux bash scripts and use tools like awk[10]. We use Gnuplot ([Jan09]) for some of the plots.

For more advanced data analysis, we use R ([IG96]), a programming language for statistical computing and graphics. We choose to use this language because

- it permits to associate to raw data semantic information that are essential for further complex processing
- it provides ready-to-use statistical functions and plotting capabilities
- it permits to combine data in a very natural way, similar to SQL
- it is suitable for mathematical computation, like Matlab

The major limitation that we find in R is that it reads data into memory by default. The data needed for some of our computations are huge and cannot be contained in the RAM only. The result is that, by default, R run into "cannot allocate memory" problems. Fortunately, R has few packages for big data support[11]. We use *ff* package, that provides file-based access to datasets that cannot fit in memory: the programmer invokes the same functions[12] as if the data set were entirely in the memory and the ff package performs the underlining operations to map from disk to memory only necessary/active parts of the data.

All the functions we use are in `offline_analysis/R_scripts/get_affected_flows.r` . The best way to use these functions is editing the file above in its last part and inserting the function calls that are needed; then, launching

```
sh offline_analysis/Rlauncher.sh
```

from the shell. This script will execute the R code: all the function definitions will be loaded and the function calls executed. A log file will be written in the file specified inside `get_affected_flows.r` to check for possible errors.

6. Statistical characterization of the queueing delay and per-application breakdown

Refer to the figures of [Ara13][13]. Almost all plots of chapter 4 are obtained with R. To obtain them from scratch, edit `offline_analysis/Rscripts/get_affected_flows.r`, insert the following lines as the main instructions and run that script.

`eps` files will be produced as output. To locate them, you can check the code.

---

[10] In a Linux shell, run `man awk` for more details

[11] See
http://www.bytemining.com/2010/08/taking-r-to-the-limit-part-ii-large-datasets-in-r/
for a good introduction to this topic

[12] Actually, we found that this is only partially true

[13] http://perso.telecom-paristech.fr/~araldo/pmwiki/uploads/Main/thesis.pdf

To obtain fig 4.1 and fig 4.3, insert:

```
build_outgoing_windows_df()
calculate_percentiles()
build_frequency_plots()
```

To obtain fig. 4.2, insert:

```
build_outgoing_windows_df()
calculate_percentiles()
plot_percentiles()
```

To obtain fig. 4.4, insert:

```
build_outgoing_windows_df()
build_quantile_time_evolution()
plot_quantile_time_evolution()
```

To obtain fig. 4.6, insert:

```
build_outgoing_windows_df()
plot_percentage_bar()
```

To obtain fig. 4.7 and fig 4.8, insert:

```
build_outgoing_windows_df()
plot_class_distinguished_frequency_plots()
```

References

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

[Chi12] C. Chirichella. A methodology to gauge the extent of Bufferbloat in the Internet: LEDBAT vs TCP . Tesi di laurea specialistica, Universita' degli Studi di Napoli Federico II, 2011/12.

[FMM+11] A. Finamore, M. Mellia, M. Meo, M.M. Munafo, and D. Rossi. Experiences of internet traffic monitoring with tstat. *Network, IEEE*, 25(3):8–14, 2011.

[IG96] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314, 1996.

[Jan09] Philipp K Janert. *Gnuplot in action: understanding data with graphs*. Manning Publications Co., 2009.

[JLM94] V Jacobson, C Leres, and S McCanne. libpcap, lawrence berkeley laboratory, berkeley, ca. *Initial public release June*, 1994.