TEACUP v1.0 – Data Analysis Functions

Sebastian Zander, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 150529B
Swinburne University of Technology
Melbourne, Australia
szander@swin.edu.au, garmitage@swin.edu.au

***Abstract*—Over the last few decades several TCP congestion control algorithms were developed in order to optimise TCP's behaviour in certain situations. While TCP was traditionally used mainly for file transfers, more recently it is also becoming the protocol of choice for streaming applications, for example video streaming from YouTube and Netflix is TCP-based [1], [2] and there is an ISO standard for Dynamic Adaptive Streaming over HTTP (DASH) [3]. However, the impact of different TCP congestion control algorithms on TCP-based streaming flows (within a mix of other typical traffic) is not well understood. Experiments in a controlled testbed allow shedding more light on this issue. TEACUP (TCP Experiment Automation Controlled Using Python) version 1.0 is a software tool for running automated TCP experiments in a testbed [4]. This report describes the analysis functions TEACUP provides to extract statistics and plot graphs based on data collected during the experiments.**

***Index Terms*—TCP, experiments, automated control, data analysis**

CONTENTS

I. Introduction

Over the last few decades several TCP congestion control algorithms were developed in order to optimise TCP's behaviour in certain situations. While TCP was traditionally used mainly for file transfers, more recently it is also becoming the protocol of choice for streaming applications, for example video streaming from YouTube and Netflix is TCP-based [1], [2] and there is an ISO standard for Dynamic Adaptive Streaming over HTTP (DASH) [3]. However, the impact of different TCP congestion control algorithms on TCP-based streaming flows (within a mix of other typical traffic) is not well understood. Experiments in a controlled testbed allow shedding more light on this issue.

This report describes the data analysis functionality of TEACUP (TCP Experiment Automation Controlled Using Python) version 1.0 – a software tool for running automated TCP experiments in a testbed [4]. The TEACUP project originated at Swinburne University of Technology's Centre for Advanced Internet Architectures (http://caia.swin.edu.au/tools/teacup), and from version 1.0 the source code is freely available on SourceForge at http://sourceforge.net/projects/teacup

Based on a configuration file TEACUP can perform a series of experiments with different traffic mixes, different bottlenecks (such as bandwidths, queue mechanisms), different emulated network delays and/or loss rates, and different host settings (e.g. TCP congestion control algorithm) [4]. For each experiment TEACUP automatically collects relevant information that allows analysing TCP behaviour. Here we describe the various tools provided by TEACUP for extracting and analysing the data of an experiment or a series of experiments.

This report is organised as follows. Section II provides an overview of TEACUP's architecture for data extraction and analysis. Section III describes the data extraction functions of TEACUP. Section IV describes the data analysis and plotting functions of TEACUP. Sections V and VI describe customisation of plotting and additional analysis-related utility functions respectively. Section VII describes use cases in tutorial-style that demonstrate how to use various analysis tasks. Section VIII concludes and outlines future work.

II. Overview

This section provides an overview of TEACUP's architecture for data extraction and analysis including some

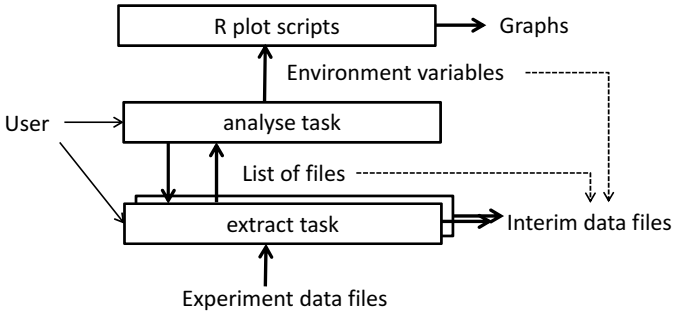


Figure 1: TEACUP's data extraction and analysis architecture

information on how to modify or extend it. It also provides some information on implementation details that are common to all extract and analysis functions.

A. Architecture

TEACUP's extract and analysis functions are implemented as Fabric tasks [4].

The architecture of TEACUP's analysis part is relatively straight-forward. At the lowest layer TEACUP uses extract tasks to extract statistics from the data collected during experiments and store these in interim data files. The extract tasks can be invoked by end-users directly to extract data, but they are also called by higher-layer analyse tasks, invoked by the user, that generate graphs based on the extracted data. TEACUP's analyse tasks pre-process the data for plotting, but the actual graph generation is carried out by R functions. Figure 1 depicts this architecture. As the figure indicates there is not a one-to-one relationship between analysis and extract tasks, as some analysis tasks may run multiple extract tasks.

Extract tasks generate files with interim data and pass the names of these files to an analyse task if called from that analysis task. The analysis task will do further processing and eventually pass (a subset) of the files together with other parameters to the plot function.

TEACUP cleanly separates the Python pre-processing and R plotting code with a well-defined interface between the the two parts. As Figure 1 indicates, information regarding the data to be plotted is passed using environment variables, so TEACUP's existing plotting functions can be replaced by alternative user-supplied plotting functions implemented in any convenient language or tool.

B. Experiment configuration information

For experiments carried out with TEACUP prior to version 0.9, the config.py file used when extract or analysis tasks are executed must be the file used to run the experiments (as far as the testbed configuration is concerned) or the config file used to run the experiments must be specified with –set teacup_config (see technical report [4] on how to load user-specified config files).

Since version 0.9 TEACUP stores all config variables for a series of experiments in a file <test_id_pfx>_varying_params.log.gz located in the experiment series sub-directory (see technical report [4]). Extract and analysis functions will access this file and the config file used to run the experiments is not required anymore when running analysis tasks.

C. Modifying or replacing analysis functionality

The data extraction and analysis aspect of TEACUP is quite distinct from the tasks that actually run experiments and gather testbed data. Instead of using TEACUP's analysis functions, end-users may want to develop their own data analysis and graphing tools. However, TEACUP also provides some options to customise the behaviour of its analysis functions, so users may find it easier to modify or extend the existing functions instead. For example, Section V describes how to customise the plotting by (1) manipulating the environment variables passed to the plot script and (2) replacing the default plot scrips with customised scripts.

D. Interim data file names

Interim data files are named as follows. The first part of the file name is the experiment ID, which allows to link the interim data file to the experiment it was generated from. Then for extract and analysis task that process per-flow data, the next part of the file name is of the form `<src_ip>_<src_port>_<dst_ip>_<dst_port>`, where <src_ip> is the source IP address, <src_port> is the source port, <dst_ip> is the destination IP address and <dst_port> is the destination port. The last part of the file name is an extension that indicates the type of data a file contains. The extension is specific to the extract function. Section III describes the extension names in more detail. Listing 1 shows an example interim data file name.

```
20150218-135735_experiment_aqm_pfifo_responders_4_172.16.10.60_50001_172.16.11.61_80.rtimes
```

Listing 1: Example interim data file name

E. Directory caching

Most of TEACUP's extract and analyse tasks make use of the `find` command line tool to locate experiment data files or intermediate data files used for plotting. The root directory for find is the directory in which the task is run (or in other words the fabfile.py directory). In case there are many experiment series' stored in sub directories under this directory the find process will be slow.

To speed up the extraction and analysis, since version 0.9 TEACUP uses a directory cache, which stores a map of test ID prefixes and their associated sub directories. If there is a cache entry for the test ID of a particular file that TEACUP is trying to locate, then the search will only be carried out under the cached sub directory. The cache file is named teacup_dir_cache.txt and is stored in the directory where fabfile.py is located.

Note that if experiment data files or intermediate data files are moved to different locations, the user must manually update existing entries in teacup_dir_cache.txt or simply remove the cache file. In the latter case TEACUP will rebuild the cache based on the user's future extract or analyse commands.

F. Flow caching

Many of TEACUP's extract and analysis functions work on a per-flow basis. They first extract the 5-tuples (source IP address, source port, destination IP address, destination port, protocol) of the different flows that occurred during experiments. Then, they extract the per-flow data.

To speed up the extraction and analysis, since version 0.9 TEACUP uses a flow cache, which stores a map of file names (including the path relative to the directory in which fabfile is located) and their associated flow tuples. If a cache entry is present for a file, TEACUP will use the flow tuples from the cache as basis for the analysis. If there is no cache entry, TEACUP will extract the flow tuples from the file (e.g. tcpdump file) and create a new cache entry. The cache file is teacup_flow_cache.txt and is stored in the directory where fabfile.py is located.

Note that if experiment data files are moved to different locations, the user must manually update existing entries in teacup_flow_cache.txt or simply remove the cache file. In the latter case TEACUP will rebuild the cache based on the user's future extract or analyse commands.

III. Data Extraction

TEACUP provides a number of basic extract tasks for extracting data to be plotted from the data collected during the experiments. Here we describe the extract tasks, their parameters and their output. The extracted data is used by the analysis tasks described in Section IV. Note that analysis tasks automatically call relevant extract tasks to extract the data needed for plotting. However, the extract tasks can be executed directly by an end-user who wishes to only extract data but not plot graph(s).

First, we provide an overview of all extract tasks and describe common parameters for all extract tasks (which can also be passed to analysis tasks). Then we describe the different extract tasks and their task-specific parameters (if any).

A. Extract tasks overview

Currently extract tasks exist for:

1) Extracting the packet size data for throughput plots from tcpdump data (extract_pktsizes);
2) Extracting Round Trip Time (RTT) using SPP [5], [6] based on tcpdump data (extract_rtt);
3) Extracting the TCP congestion window size (CWND) from SIFTR and Web10G data (extract_cwnd);
4) Extracting TCP's RTT estimate from SIFTR and Web10G data (extract_tcp_rtt). The task extracts both, the smoothed estimate and an unsmoothed estimate (for SIFTR the unsmoothed estimate is the improved ERTT [7] estimate);
5) Extract arbitrary TCP statistics from SIFTR and Web10G data (extract_tcp_stat);
6) Extracting cumulative bytes acknowledged or dupACKs over time from tcpdump data (extracting_ackseq);

7) Extracting the goodput for DASH-like streaming clients (extract_dash_goodput);
8) Extracting the response times for incast experiments from httperf log files (extract_incast);
9) Extracting the response times for incast experiments from tcpdump files (extract_incast_restimes);
10) Extrating the inter-query times for incast experiments from tcpdump files (extract_incast_iqtimes).

B. Common parameters

This section describes the common parameter. Most of these are supported by all extract tasks, but some of them are only supported by a subset of tasks.

1) Specifying experiments to process: The `test_id` parameter specifies the ID of the experiment from which to extract data. The `test_id` parameter accepts multiple test IDs separated by semicolons.[1] If a list is specified, data will be extracted for each test ID. Extracted data is stored in files that start with the experiment ID and have a task-specific extension.

2) Replot only: All extract tasks have a parameter `replot_only`. While the parameter can be used when executing an extract task, it is mainly used for analysis tasks and passed on to the underlying extract tasks. This parameter allows to replot the graphs without extracting the data again (e.g. from tcpdump, SIFTR or Web10G files). If `replot_only` is set to '1' data is still extracted for experiments where data has not been extracted before, but for experiments with already extracted data the graph(s) are created based on the existing extracted data.

3) Data output directory: By default all extracted data files are generated in the sub directory where the experiment data is located (the sub directory name being the test ID prefix) inside the directory where fab is executed. To put the output files into a specific directory, the `out_dir` parameter can be used. If `out_dir` is an absolute path, the output files will be put in the specified directory. However, if `out_dir` is a relative path, it is treated as relative *to the location of the log files* when determining the directory for the output files.

4) Clock offset analysis and correction: TEACUP provides an additional mechanism to evaluate the quality of the time synchronisation between hosts and (optionally) correct for clock offsets in the post-analysis.[2] This mechanism requires that broadcast/multicast pings were enabled for an experiment (with TPCONF_bc_ping_enable='1') as described in Section IV-I in technical report [4],

a) Computing clock offsets: The get_clock_offsets tasks allows computing the clock offsets between all hosts using the clock of one host as reference (by default the router's clock). The task will generate a file for each experiment in experiments_completed.txt by default, for a specific experiment if the `test_id` parameter is specified, or for a number of experiments listed in a file specified with the `exp_list` parameter.

The `baseline_host` parameter can be used to specify the reference clock and the `out_dir` parameter specifies the directory in which the clock offset file is generated. Assuming <test_id> is the experiment ID, the name of the clock offsets file is <test_id>_clock_offsets.txt.

The following is an example of using the get_clock_offsets task where we specify a specific output directory and instruct TEACUP to use the clock of host 'testrouter' as reference:

```
> fab get_clock_offsets:out_dir=../clock_data,
baseline_host=testrouter
```

b) Correcting timestamps during data extraction or analysis: The `ts_correct` parameter instructs extract tasks to use previously-computed clock offset data to correct timestamps for plotting (e.g. timestamps from tcpdump or web10g files). As explained earlier, the correction can only be done if an experiment was run with broadcast/multicast pings (see Section IV-I in technical report [4]).

No timestamp correction occurs if `ts_correct=0` or is undefined. If `ts_correct=1`, the extract functions will generate uncorrected data files as well as additional data files with all timestamps corrected according to the clock offsets calculated by the get_clock_offsets task (the additional files have an appended .tscorr extension), and analyse functions will plot the data with timestamps corrected (based on the .tscorr files).

Note that if ts_correct is set to 1 and no clock offsets file is present, TEACUP will automatically try to create the file[3]. Also note that enabling `ts_correct` will only

[1] Since TEACUP v0.5. No trailing semicolon for only one test ID.

[2] Since TEACUP version 0.7.

[3] <test_id>_clock_offsets.txt file

modify timestamps in the additional interim data files used for plotting. Original timestamps in the actual log files (e.g. timestamps in tcpdump or web10g files) are *not* modified.

Prior to TEACUP version 0.9 `ts_correct=0` was the default, but since version 0.9 the default is `ts_correct=1`. If an experiment was run without broadcast/multicast pings, `ts_correct=0` must be specified for extract or analysis tasks, or otherwise these tasks will fail.

5) Data series selection: All extract tasks, except extract_dash_goodput, provide the `source_filter` parameter that provides each extract and analysis function with a rudimentary mechanism to identify the flows to be plotted. Note, `source_filter` only limits what is plotted. Intermediate data files are always extracted for all flows in an experiment. However, the list of files passed by the extract task to the analysis task will only contain files for the selected flows.

Desired flows may be specified using combinations of patterns matching source and/or destination IP address and port numbers.

The filter string format is:

```
(S|D)_<ip>_<port>[;(S|D)_<ip>_(<port>|'*')]*
```

For example, the following command only plots data for flows *from* host 172.16.10.2 port 80:

```
> fab analyse_all:source_filter=
"S_172.16.10.2_80"
```

Note, that the notion of 'flow' here is unidirectional. The flow consisting of packets heading *to* host 172.16.10.2 port 80 would be selected by specifying:

```
> fab analyse_all:source_filter=
"D_172.16.10.2_80"
```

The specified filter string also determines the order of the flows in the graph(s). Flows are plotted in the order of the filters specified. For example, if there are two flows, one from host 172.16.10.2 port 80 and another from host 172.16.10.2 port 81 by default the port 80 flow would be the first data series and the port 81 flow would be the second data series. One can reverse the two flows in the graphs by specifying:

```
> fab analyse_all:source_filter=
"S_172.16.10.2_81;S_172.16.10.2_80"
```

Instead of an actual port number on can specify the wildcard character ('*'). This allows to filter on a specific source or destination with any port number. For example, we can plot data for all flows *from* host 172.16.10.2 regardless of their port numbers:

```
> fab analyse_all:source_filter=
"S_172.16.10.2_*"
```

Note that `source_filter` identifies the flows selected, not the flows filtered out.

6) Filtering SIFTR log lines: The extract_all, extract_cwnd, extract_tcp_rtt and extract_tcp_stat tasks have a parameter `io_filter` that allows specifying whether TCP statistics are extracted based on incoming (set value to 'i') outgoing (set value to 'o') or incoming and outgoing packets (set value to 'io') from SIFTR log files. The default value is 'o'. Currently, the parameter is only effective with SIFTR log files, it does nothing with web10g log files.

Note: this parameter only takes effect if `replot_only` is set to '0' (i.e. you must re-extract intermediate data if you want to change io_filter)

7) Burst-separation (incast experiments): The tasks extract_rtt, extract_ackseq and extract_incast can extract data on a per-burst basis for incast experiments, where we have bursts of TCP requests sent by a querier to multiple responders interspersed by idle times. The parameter `burst_sep` allows to specify the time between bursts. Any idle time longer than `burst_sep` signals the start of a new burst. If `burst_sep` is set to 0, data will not be separated according to bursts.

The parameters `sburst` and `eburst` can be used to select the bursts to be plotted (start index of 1). They are not useful when running an extract task directly. They are only useful when passed to an analyse task, which passes them further to an extract task.

C. extract_pktsizes – Extract packet sizes for throughput computation

This task extracts packet size information from tcpdump files. The packet size data is used by the analysis_throughput task to compute and plot the throughput. Extracted data files are created for each traffic flow and have a .psiz extension. They contain the following space-separated columns with one line per packet: timestamp, packet size in bytes (from tcpdump file).

Besides the common parameters the task has the parameters `link_len` and `total_per_experiment`.

1) IP layer vs link layer packet size: The parameter `link_len` allows to specify whether the packet sizes extracted are the IP sizes ('0') or the sizes including the link layer header ('1').

2) Total over all flows: By default data is extracted per flow, but if `total_per_experiment` is set to '1' in addition a file with all packet sizes will be created. Note that if `total_per_experiment` is set to '1', a source_filter should be specified to prevent aggregating both directions of traffic.

3) Examples: The following shows an example of how to use the task:

```
> fab extract_pktsizes:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
link_len=1
```

D. extract_rtt – RTT estimates (based on SPP)

This task computes RTTs for bidirectional flows using SPP [6]. Extracted data files are created for each bidirectional traffic flow with .rtts extension. They contain the following space-separated columns with one line per timestamp: timestamp of RTT and RTT estimated by SPP.

Besides the common parameters the task has the parameter `udp_map`. This parameter allows to map unidirectional UDP flows in opposite direction into bidirectional UDP flows, as SPP needs bidirectional flows to estimate the RTT. The format for the parameter is a semicolon-separated list of source IP source port pairs. Each pair must be specified as <src_ip1>,<src_port1>:<src_ip2>,<src_port2>, where <src_ip1>,<src_port1> is the source IP addresses and source port pair of one UDP flow and <src_ip2>,<src_port2> is the source IP addresses and source port pair of another UDP flow (in opposite direction). The task also has the `burst_sep`, `sburst` and `eburst` parameters which make it possible to extract the RTT estimates on a burst-by-burst bases for incast experiments.

1) Examples: The following shows an example of how to use the task:

```
> fab extract_rtt:test_id=
20131206-102931_tcp_newreno,
out_dir=./results/
```

The following command shows how to combine two unidirectional UDP flows, one originating from 192.168.0.10 port 5000 and the other originating from 192.168.0.11 port 5001:

```
> fab extract_rtt:test_id=
20131206-102931_tcp_newreno, udp_map=
"192.168.0.10,5000:192.168.0.11,5001"
```

E. extract_cwnd – TCP CWND

This task extracts the CWND data from TCP log files. Extracted data files are created for each traffic flow and have a .cwnd extension. They contain the following comma-separated columns with one line per timestamp: timestamp and CWND value (from SIFTR or web10g log file). Besides the common parameters the task has the parameter `io_filter` that can be used in case of SIFTR logs to use TCP statistics only for incoming, outgoing or incoming and outgoing packets (explained in Section III-E).

1) Examples: The following shows an example of how to use the task:

```
> fab extract_cwnd:test_id=
20131206-102931_tcp_newreno,
out_dir=./results/
```

F. extract_tcp_rtt – RTT estimates from TCP

This task extracts TCP's RTT estimates from the TCP log files (SIFTR or web10g). Extracted data files are created for each traffic flow and have a .tcp_rtt extension. They contain the following comma-separated columns with one line per timestamp: timestamp, smoothed RTT estimate and sample/unsmoothed RTT estimate. Besides the common parameters the task has the parameter `io_filter` (explained in Section III-E).

1) Examples: The following shows an example of how to use the task:

```
> fab extract_cwnd:test_id=
20131206-102931_tcp_newreno,
out_dir=./results/,io_filter=o
```

G. extract_tcp_stat – Extract any TCP statistic

This task extracts a user-defined statistic from the TCP log files. Extracted data files are created for each traffic flow and have a .tcpstat<num> extension where <num> is the index of the statistic (the column number in the

SIFTR or web10g log file). They contain the following comma-separated columns with one line per timestamp: timestamp, TCP statistic selected (from SIFTR or web10g log file).

Besides the common parameters the task has the parameter `io_filter` (explained in Section III-E), and the parameters `siftr_index` and `web10g_index`.

1) Statistic to extract: The parameter `siftr_index` is used to specify the column of the statistic in SIFTR log files and the parameter `web10g_index` is used to specify the column of the statistic in web10g log files. The start index for both parameters is 1, which selects the statistic in the first column. If the log files are a mix of SIFTR and web10g both parameters must be specified. Otherwise it is sufficient to specify just one of the parameters.

2) Examples: The following shows an example of how to use the task:

```
> fab extract_tcp_stat:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
siftr_index=22,web10g_index=120
```

H. extract_all – Combination of basic extract methods

This task executes the following tasks: extract_rtt, extract_cwnd, extract_tcp_rtt and extract_pktsizes. The task parameters are a superset of the parameters of the individual tasks.

I. extract_ackseq – Extract acknowledged bytes or dupACKs

This task extracts TCP acknowledgement data. It is useful for incast experiments but not limited to these. Extracted data files are created for each traffic flow and have an .acks.0 extension (without burst separation) or a file extension is .acks.<burst_number> (with burst separation). The data files contain the following space-separated columns with one line per timestamp: timestamp, cumulative acknowledged bytes (from tcpdump log) and cumulative dupACKs (from tcpdump log).

The task has the common parameters as well as the burst-separation parameters `burst_sep`, `sburst` and `eburst` described in Section III-B7. The task also has the parameter `total_per_experiment` that if set to '1' will generate an additional file with the total cumulative acknowledged bytes and total cumulative dupACKs over all flows.

Since this function extracts data from the ACK stream, source_filter is effectively reversed compared to other tasks, such as extract_pktsizes. For example, if one wanted to extract throughput for the source with IP 192.168.0.10 one would specify `source_filter="S_192.168.0.10_*"`. However, to extract the corresponding ACK data with extract_ackseq one has to specify `source_filter="D_192.168.0.10_*"`, as the ACKs travel in the opposite direction.

1) Total over all flows: By default data is extracted per flow, but if `total_per_experiment` is set to '1' in addition a file with total acknowledged bytes and total dupACKs will be created. Note that if `total_per_experiment` is set to '1', a source_filter should be specified to prevent aggregating both directions of traffic.

2) Examples: The following shows an example of how to use the task:

```
> fab extract_ackseq:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
burst_sep=1.0
```

J. extract_dash_goodput – Extract goodput for DASH experiments

This task extracts goodput for DASH-like clients. Extracted data files are created for each DASH-like client and have a .dashgp extension. They contain the following comma-separated data columns with one line per data block downloaded: timestamp of request, size of requested and downloaded block in bytes, bit rate in megabit per second, response time in seconds, nominal cycle length in seconds, nominal rate in kilobits per second, and block number.

Besides the common parameters the task has a `dash_log_list` parameter. This parameter is an alternative to specifying a list of test IDs, and allows a user to specify a file name that contains a list of httperf log files for DASH-like clients which should be processed. The format of the log list file is one file name per line. The file paths do not need to be specified, as TEACUP will automatically find the files assuming they are in a sub directory below the fabfile.py directory. If the `test_id` parameter is specified, the `dash_log_list` parameter is ignored.

1) Examples: The following shows an example of how to use the task:

```
> fab extract_dash_goodput:
dash_log_list=dash_logs.txt,out_dir=./results/
```

K. extract_incast – Extract response times for incast experiments (from httperf logs)

This task extracts response times for incast experiments. Extracted data files are created for each traffic flow and have an .rtimes extension (single responders) or an .rtimes.all extension (all responders merged). They contain the following space-separated columns with one line per timestamp: timestamp, burst number (starting from 1), response time in seconds (from httperf log file).

Besides the common parameters the task has the `sburst` and `eburst` parameters (`burst_sep` is not needed here as the httperf log data is already separated in bursts) to enable plotting of the response times only for selected bursts.

1) Filtering on flows: Note that for extract_incast flows are bidirectional, and the source of an incast flow is always the querier and the destination of an incast flow is always one of the responders. This needs to be taken into account when using `source_filter`. Also note that since there is no information about a querier's source port numbers in httperf log files, TEACUP will create fake source port numbers.

2) Slowest response only: The task has a parameter `slowest_only` (default is '0'). If `slowest_only` is set to '1', the task will also generate a file with the extension .rtimes.slowest that only contains the slowest response time over all flows for each burst. If `slowest_only` is set to '2', the task will also generate a file with the extension .rtimes.slowest that only contains the time between the first request sent and the time the last response was finished for each burst (default is '0').

3) Examples: The following shows an example of how to use the task:

```
> fab extract_incast:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
sburst=2,eburst=0
```

L. extract_incast_restimes – Extract response times for incast experiments (from tcpdump)

This task extracts response times for incast experiments. The difference between the extract_incast task (Section III-K) and this task is that the former extracts the response times from httperf logs while this task extracts the response times from tcpdump files. Extracted data files are created for each traffic flow and have an .restimes extension. They contain the following space-separated columns with one line per timestamp: timestamp the request was sent, burst number (starting from 1), <IP>.<port> of the querier, <IP>.<port> of the responder, and the response time in seconds (time between last packet of response and GET packet from tcpdump).

The task has the common parameters. Furthermore, it has the parameter `query_host`, which must be used to specify the querier (set to name as used in TPCONF_hosts)

1) Filtering on flows: Note that for extract_incast_restimes flows are bidirectional, and the source of an incast flow is always the querier and the destination of an incast flow is always one of the responders. This needs to be taken into account when using `source_filter`.

2) Slowest response only: The task has a parameter `slowest_only` (default is '0'). If `slowest_only` is set to '1', the task will also generate a file with the extension .restimes.slowest that only contains the slowest response time over all flows for each burst. If `slowest_only` is set to '2', the task will also generate a file with the extension .restimes.slowest that only contains the time between the first request sent and the time the last response was finished for each burst (default is '0').

M. extract_incast_iqtimes – Extract inter-query times (incast experiments)

This task extracts the times between requests sent by the querier for incast experiments. Extracted data files are created for each traffic flow and have an .iqtime.all extension (combined inter-query times) and .iqtime.<responder> where <responder> is the IP plus port number of a responder (per responder inter-query times). They contain the following space-separated columns with one line per timestamp: timestamp, IP of responder, port number of responder, time between request and first

request in burst sent by querier, and time between request and previous request in same burst sent by querier.

The task has the common parameters. It also has the `burst_sep` parameter used to detect the bursts; however, the semantic here is slightly different as `burst_sep` *must* be a number greater than 0 (default is 1.0) in order to separate the query bursts. The parameter `query_host` must be used to specify the querier (set to name as used in TPCONF_hosts).

1) Aggregate responders: The parameter `by_responder` is set to '1' by default meaning per responder data files are created. If set to '0' one file with the times for all responders is created.

2) Cumulative data: By default and if the parameter `cumulative` is set to '0' raw inter-query times are computed (as explained above). If the parameter `cumulative` is set to '1' subsequent bursts are plotted in a cumulative fashion, i.e. the time of the first request in a burst is the time of the last request in the previous burst. Cumulative numbers make it easier to see long term trends, such as the inter-query times being consistently larger in one experiment compared to another experiment.

3) Examples: The following shows an example of how to use the task:

```
> fab extract_iqtimes:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
query_host=testhost1,cumulative=1
```

N. extract_pktloss – Extract packet loss information

This task can only be used with emulated FPS game traffic generated with the start_fps_game function. Extracted data files are created for each traffic flow and have a .loss extension. They contain the following space-separated columns with one line per packet: timestamp, lost flag (0 if packet arrived, 1 if packet was lost). The task has the common parameters.

1) Examples: The following shows an example of how to use the task:

```
> fab extract_pktloss:test_id=
20141003-111821_game_traffic_aqm_pfifo,
out_dir=./results
```

IV. Data Analysis

Now we describe TEACUP's analyse tasks that allow to plot various types of graphs. First, we give an overview of the existing analysis tasks. Then we discuss parameters common to most tasks as well as common environment variables that are used by the plotting code. Next, we discuss the tasks in detail. Finally, we discuss ways to customise the plotting beyond using the existing tasks.

A. Analysis tasks overview

Currently basic analysis tasks exist for:

1) Plotting the throughput including all header bytes based on tcpdump data (analyse_throughput);
2) Plotting the Round Trip Time (RTT) using SPP [5], [6] based on tcpdump data (analyse_rtt);
3) Plotting the TCP congestion window size (CWND) based on SIFTR and Web10G data (analyse_cwnd);
4) Plotting the TCP RTT estimate based on SIFTR and Web10G data (analyse_tcp_rtt). The task can plot both, the smoothed estimate and an unsmoothed estimate (for SIFTR the unsmoothed estimate is the improved ERTT [7] estimate);
5) Plotting an arbitrary TCP statistic from SIFTR and Web10G data (analyse_tcp_stat);
6) Plotting cumulative bytes acknowledged or dupACKs over time from tcpdump data (analyse_ackseq);
7) Plotting the goodput for DASH-like streaming clients (analyse_dash_goodput);
8) Plotting the goodput for TCP flows based on acknowledged bytes (analyse_goodput);
9) Plotting the response times for incast experiments (analyse_incast);
10) Plotting the inter-query times for incast experiments (analyse_incast_iqtimes).

A convenience function exists that plots graphs 1–4 listed above (analyse_all).

The above tasks usually plot a single statistic over time. However, there are two tasks that allow to compare the above statistics across different experimental settings:

1) Plotting boxplots, mean or median of a statistic for different combinations of experiment parameters (analyse_cmpexp);

2) Plotting 2D-density plots or ellipse plots of two statistics for different combinations of experiment parameters (analyse_2d_density).

B. Common task parameters

This section describes common task parameters. Many of them are supported by all tasks, but some of them are only supported by a subset of tasks, as noted in the task-specific sections. For precise information on task parameters refer to the TEACUP command reference [8]. Task-specific parameters are described in the sections that describes the task.

1) Specifying experiment(s) to analyse: Most analyse task have a `test_id` parameter. The analysis can be run for a single experiment only by specifying a single test ID. For example, the following command generates the RTT graph for the experiment `20131206-102931_dash_2000_tcp_newreno`:

```
> fab analyse_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno
```

The `test_id` parameter also accepts multiple test IDs separated by semicolons.[4] If multiple IDs are specified, results from each test ID will be plotted on the same graph(s). For time series graphs, produced by analyse_rtt or analyse_throughput etc., the resulting graphs' file name(s) will be the *first* test ID specified followed by the string "_comparison" to distinguish from graphs where only one experiment is plotted, created in the sub directory of the test ID specified first.

Instead of the `test_id` parameter some tasks, such as analyse_cmpexp and analyse_2d_density, have a parameter that allows specifying a file that contains a list of experiment IDs.

2) Data series selection: All analyse tasks, except analyse_dash_goodput, provide the `source_filter` parameter that gives each extract and analysis function a rudimentary mechanism to identify the flows to be plotted. Note, `source_filter` only limits what is plotted. Intermediate data files are always extracted for all flows in an experiment. However, the list of files passed by the extract task to the analysis task will only contain files for the selected flows. Section III-B5 describes how to use the parameter.

[4]Since TEACUP v0.5. No trailing semicolon for only one test ID.

3) Re-using previously extracted data: All analysis functions above have a parameter `replot_only`. This parameter allows to replot the graphs without extracting the data again (e.g. from tcpdump, SIFTR or Web10G files). If `replot_only` is set to '1' data is still extracted for experiments where data has not been extracted before, but for experiments with already extracted data the graph(s) are created based on the existing extracted data. For example, the following command recreates the graphs without extracting already extracted data again:

```
> fab analyse_all:replot_only=1
```

4) Specifying the location of intermediate data files: By default all interim data files are generated in a sub directory (with the directory name being the test ID prefix) inside the directory where fab is executed. To put the output files into a specific directory the `out_dir` parameter can be specified (and it will be passed to the extract tasks(s)). If `out_dir` is an absolute path, the output files will be put in the specified directory. If `out_dir` is a relative path, it is treated as relative *to the location of the log files* when determining the directory for the output files. The directories are created automatically if they do not exist. The following is an example where the output files are put in sub-directories named 'results' inside each test ID prefix sub directory with experiment data:

```
> fab analyse_all:out_dir=./results/
```

Assuming we executed the last command in a fabfile directory <fabfile_dir> with two sub directories, one for test ID prefix `20131206-102931_exp` and one for test ID prefix `20131206-124510_exp`, the output files will be put in the directories:

```
<fabfile_dir>/20131206-102931_exp/results/
<fabfile_dir>/20131206-124510_exp/results/
```

5) Specifying the location of final graphs: By default, the final PDF files (graphs) will be created in `out_dir`. This can be adjusted using the `pdf_dir` parameter. Like `out_dir` the specified directory is either an absolute path or a relative path which is relative to the location of the experiment's log files. The directory is automatically created if it does not exist. The parameter can be used as follows:

```
> fab analyse_spp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
out_dir=./results,pdf_dir=./pdfs
```

6) Suppressing small or unchanging datasets: In many experiments we may have TCP flows where data only/-mostly flows in one direction and TCP statistics in the other direction are basically constant. The `omit_const` parameter can be used to suppress any completely constant series (i.e. all values are identical). It can be used as follows:

```
> fab analyse_all:omit_const=1
```

Any flows that have only very few data points (less or equal than `min_values`) are excluded from the plot (by default `min_values` = 3). The min_values parameter can be changed on the command line, for example the following command omits any flows with 20 data points or less from the plots:

```
> fab analyse_all:min_values=20
```

7) Adjusting the y-axes: All analyse tasks have the parameters `ymin` and `ymax`. These parameter can be used to set the y-axis limits to specific values, for example to produce multiple plots with the same scale (by default `ymin` is 0 and `ymax` is determined automatically). The parameters can be used as follows (here the y-axis range is set to 100–200 ms):

```
> fab analyse_spp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
ymin=100,ymax=200
```

8) Adjusting time (x-axes) scales: All tasks tasks that plot data over time have the parameters `stime` and `etime` to control the x-axis limits of the plots (by default `stime` is 0.0 and `etime` is the duration of the experiment). Note that using these parameters allows to zoom in, but the data outside the specified interval is not filtered out. The y-axis maximum is adjusted automatically to the maximum occurring in the specified x-axis interval, but the legend is not adjusted. To remove unwanted entries in the legend (e.g. flows not in the time window), one must use `source_filter` to filter out the the unwanted flows (see Section III-B5). The parameters can be used as follows (here the x-axis range is set to 5–10 seconds):

```
> fab analyse_spp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
stime=5,etime=10
```

9) User-supplied legend names: By default the legend entries are simply the flow tuples (source IP, source port, destination IP, destination port). The parameter `lnames` can be used to replace these with more informative names. One must specify the same number as names as there are data series. Names specified must be separated by semicolons. The parameter can be used as follows:

```
> fab analyse_spp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
lnames='TCP Reno;TCP Cubic'
```

10) Prefix for pdf file names: The `out_name` parameter allows to change the name of the PDF files produced. If `out_name` is specified, the prefix for the PDF files is `out_name` followed by the test ID (followed by "_comparison" for comparison graphs based on multiple test IDs). The parameter can be used as follows:

```
> fab analyse_spp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
out_name='ExperimentA'
```

11) Filtering SIFTR log lines: The analyse_all, analyse_cwnd, analyse_tcp_rtt and analyse_tcp_stat tasks have a parameter `io_filter` that allows specifying whether TCP statistics are plotted based on incoming (set value to 'i') outgoing (set value to 'o') or incoming and outgoing packets (set value to 'io') for SIFTR log files. The default value is 'o'. Currently, the parameter is only effective with SIFTR log files, it does nothing with web10g log files.

Note: this parameter only takes effect if `replot_only` is set to '0' (i.e. you must re-extract intermediate data if you want to change io_filter)

12) Correcting timestamps: All analyse tasks provide a `ts_correct` parameter that can be used to correct timestamps in the measurement data (e.g. timestamps in tcpdump files) based on estimated clock offsets (see Section III-B4). This parameter is passed to the extract task(s) and Section III-B4 describes in more detail how it works.

13) Burst-separation (incast experiments): The tasks analyse_rtt, analyse_ackseq and analyse_incast can extract and plot data on a per-burst basis for incast experiments, where we have bursts of TCP requests sent by a querier to multiple responders interspersed by idle times. The parameter `burst_sep` allows to specify the time between bursts. Any idle time longer than `burst_sep` signals the start of a new burst. If `burst_sep` is set to 0, data will not be separated according to bursts. The parameters `sburst` and `eburst` can be used to select the bursts to be plotted (they start with an index of 1). Setting `eburst=0` means to end with the last burst.

14) Total throughput/goodput: The tasks analyse_throughput and analyse_goodput by default plot throughput and goodput for each direction of each traffic flow. The parameter `total_per_experiment` can be used to plot the total throughput or goodput. Note that this parameter will cause the tasks to sum up all the traffic in both directions unless a `source_filter` is specified. Usually, we want the total only in one direction, so a `source_filter` should be specified that filters on a particular source or destination.

C. Common plotting environment variables

Plotting behaviour can be further controlled by a number of shell environment variables. Here we explain the parameters that work for many analyse functions. Task-specific variables are described in the task sections. How to set these environment variables depends on your Unix shell.

1) Space for the legend: The variable YMAX_INC controls the space for the legend. It assumes the legend is plotted at the top, which is the default. The actual y-axis maximum for the plot is $y_{\max}\,(1 + \mathrm{YMAX\_INC})$, where $y_{\max}$ is the maximum based on the data (or the maximum specified by the user).

2) Smoothing the throughput calculations: Throughput always needs to be computed over some time interval. The variables AGGR_WIN_SIZE and AGGR_INT_FACTOR allow you to specify the window size and interpolation for throughput plots (including comparison plots for throughput using the task described in Section IV-P). Interpolation is useful to 'fill in the gaps' when a long window size is chosen to mitigate noise in the data.

AGGR_WIN_SIZE is specified in seconds, with fractional values allowed. Setting AGGR_INT_FACTOR to 1 means no interpolation, whereas setting it to an integer value n greater than 1 means you will get n times the number of data points (where $n - 1$ points are interpolated points). Effectively the interpolation creates overlapping time windows, with the gaps between windows being AGGR_WIN_SIZE/AGGR_INT_FACTOR seconds. By default AGGR_WIN_SIZE=1 and AGGR_INT_FACTOR=4.

3) Point thinning: By default all plot functions will plot every point for each data series. If the data series' are very large, the resulting PDF files will be large and the figures will take a long time to display. To reduce the size of the plots and the time for opening them without loosing important information TEACUP implements "point thinning", which can be controlled with the variable PTHIN_DIST or PTHIN_DIST_FAC.

Using PTHIN_DIST one can set the minimum (Euclidean) distance between plotted data points. Any data points within the minimum distance are not plotted. For example, PTHIN_DIST=0.25 means the minimum (Euclidean) distance between two plotted points is 0.25 and any data points in-between are not plotted. By default PTHIN_DIST=0, which means point thinning is disabled.

Using the absolute Euclidean distance is problematic if the scales of x-axis and y-axis differ significantly. Then, we are likely to have either too few points in one dimension or too many points in the other dimension. Also specifying an absolute distance means we may need to adjust the distance if we create zoomed-in graphs.

Since version 0.8 TEACUP also allows to specify the variable PTHIN_DIST_FAC which controls the minimum distance of points in x-dimension and y-dimension separately, relative to the x-range and y-range plotted. A point is plotted if either the distance in x-dimension is larger or equal PTHIN_DIST_FAC*<xrange> or the distance in y-dimension is larger or equal PTHIN_DIST_FAC*<yrange>, where <xrange> and <yrange> are given by the data to plot and the settings applied by the user through specifying ymin, ymax, stime, and etime.

4) Size of plotted points: The point size in graphs can be controlled with a variable POINT_SIZE. Note that POINT_SIZE does not specify an absolute point size, but it is a scaling factor that is multiplied with the actual default point size. Hence, if POINT_SIZE is set to 1.0 the size of points will be the default size, if POINT_SIZE is set to 0.5 the size of points will be half the default size and so on. By default POINT_SIZE is 0.5.

5) Filter out inactive flows in time interval: By default TEACUP does not filter out flows that were not active within the time interval specified by the user with `etime`, `stime`. Legend entries for these flows are still plotted, although they can be supressed by using `source_filter`. If FILTER_FLOWS is set to 1, TEACUP will filter out all flows for which there is no data in the time interval specified by the user. FILTER_FLOWS only works for

the tasks that plot time series, such as analyse_rtt, analyse_cwnd, analyse_throughput, with the exception that it does not work for analyse_dash_goodput.

6) Sorting flows by start time: By default, for time series plots TEACUP sorts flows by the 5-tuple or in the order of specified source filters. If SORT_FLOWS_BY_START_TIME is set to 1, the default order is ignored and flows are sorted in the order of their start times. Ordering flows by their start times can be useful in some situations, for example, if more than 12 flows are plotted as series of PDF files. The variable works for all time series plots (except analyse_dash_goodput).

7) Examples: The following shows an example where we plot the throughput with a modified aggregation time window and enabled point thinning (using the BASH shell):

```
> AGGR_WIN_SIZE=2 AGGR_INT_FACTOR=8
PTHIN_DIST=0.25 fab analyse_throughput:
test_id=20131206-102931_dash_2000_tcp_newreno
```

D. analyse_throughput – Plot throughput over time

This task plots the total or per-flow throughput over time. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the `total_per_experiment_parameter`.

1) Calculating IP-layer or link-layer throughput: The analyse_all and analyse_throughput tasks have a `link_len` parameter. If set to '0' (default) throughput is based on IP-layer packet length, if set to '1' throughput is based on link-layer frame length. Note that the bandwidth limits specified on the router are link-layer limits.

2) Examples: The following command generates a throughput graph:

```
> fab analyse_throughput:test_id=
20131206-102931_dash_2000_tcp_newreno
```

The following shows an example where we plot the throughput based on the length of link layer frames:

```
> fab analyse_throughput:test_id=
20131206-102931_dash_2000_tcp_newreno,
link_len=1
```

E. analyse_rtt – Plot RTT (SPP) over time

This task plots RTT estimated by SPP for each bidirectional flow over time. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the burst separation parameters `burst_sep`, `sburst`, `eburst` and the `udp_map` parameter (explained in Section III-D).

F. analyse_cwnd – Plot TCP CWND over time

This task plots the TCP CWND from SIFTR or web10g logs for each direction of a TCP flow over time. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the `io_filter` parameter described in Section IV-B11.

G. analyse_tcp_rtt – Plot TCP RTT over time

The analyse_tcp_rtt task plots a graph of TCP's estimate of the RTT (from SIFTR or web10g log files). The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the `io_filter` parameter described in Section IV-B11.

1) Smoothed or unsmoothed TCP RTT: By default the TCP RTT graphs generated are for the smoothed RTT estimates (and in case of SIFTR this is not the ERTT estimates). If the parameter `smoothed` is set to '0', non-smoothed estimates are plotted (and in the case of SIFTR this is the ERTT estimates). Note, the `smoothed` parameter can also be used with analyse_all, analyse_cmpexp and analyse_2d_density.

2) Examples: The following command generates the TCP RTT graph for the non-smoothed estimates:

```
> fab analyse_tcp_rtt:test_id=
20131206-102931_dash_2000_tcp_newreno,
smoothed=0
```

H. analyse_all – Combines basic analysis tasks

This task combines analyse_throughput, analyse_rtt, analyse_cwnd and analyse_tcp_rtt and has the superset of all the parameters of these tasks.

The easiest way to generate the four basic graphs (see Section IV-A) for all experiments is to run the following command in the directory containing the sub directories with experiment data:

```
> fab analyse_all
```

This command will generate results for all experiments listed in the file `experiments_completed.txt`. However, the `test_id` parameter can be used to only generate the plots for a specific experiment. Furthermore, we can pass a custom list of experiment IDs (see below).

1) Modifying the list of test IDs to analyse: For analyse_all the parameter `exp_list` allows to change the file used as list of test IDs (by default `experiments_completed.txt`), which makes it possible to adjust the list of experiments we generate results for. The following shows an example:

```
> fab analyse_all:exp_list=myexp_list.txt,
out_dir="./results"
```

2) Resuming an interrupted analyse_all: If an analysis_all was interrupted (e.g. because a log file was corrupted) we can resume the analysis after the experiment with the corrupted files. First, one needs to look up the next test ID after the corrupted test ID in `experiments_completed.txt`. Then, one can resume at this test ID using the `resume_id` parameter. For example, if for a test ID 20131206-102931_dash_2000_tcp_newreno_run_0 we cannot do the analysis because of corrupted data files and the next test ID is 20131206-102931_dash_2000_tcp_newreno_run1, we can continue the analysis with this command:

```
> fab analyse_all:resume_id=
20131206-102931_dash_2000_tcp_newreno_run_1
```

I. analyse_tcp_stat – Plot arbitrary TCP statistic over time

The analyse_tcp_stat function can be used to plot any TCP statistic from SIFTR or Web10G logs. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the `io_filter` parameter described in Section IV-B11.

1) Selecting statistic: The parameter `siftr_index` defines the index of the column of the statistic to plot for SIFTR log files. The parameter `web10g_index` defines the index of the column of the statistic to plot for Web10G log files. The start index for both parameters is 1, which selects the statistic in the first column. If one has only SIFTR or only Web10G log files the other index does not need to be specified. But for experiments with SIFTR *and* Web10G log files both indexes must be specified. By default both indexes are set to plot CWND. The lists of available statistics (including the column numbers) are in the SIFTR README [9] and the Web10G documentation [10].

2) Adjusting y-axis label and scaler: The parameter `ylabel` can be used to set the y-axis label to a user-defined string. The parameter `yscaler` can be used to scale the extracted data values by a user-defined factor.

3) Examples: We can plot the number of kilo bytes in the send buffer at any given time with the command:[5]

```
> fab analyse_tcp_stat:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
siftr_index=22,web10g_index=120, ylabel="Snd
buf (kbytes)",yscaler=0.001
```

J. analyse_ackseq – Plot acknowledged bytes or dupACKSs over time

This task plots the cumulative number of bytes acknowledged or the cumulative number of dupACKs for TCP flows. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`.

It also has the burst separation parameters `burst_sep`, `sburst`, `eburst`. If `burst_sep` is set to 0, data will not be separated according to bursts and the plot will show acknowledged bytes or dupACKs for the whole flow. If `burst_sep` is set to 1, each burst between idle times will

[5] This command works with web10g version 2.0.9 logs. Use web10g_index=116 when parsing web10g version 2.0.7 logs.

be plotted as separate data series and the acknowledged bytes or dupACKs are normalised to that they are zero at the start of each burst. The parameters `sburst` and `eburst` can be used to select the bursts to be plotted (assuming `burst_sep` is not set to 0). Both start with an index of 1. Setting `eburst=0` means to end with the last burst.

Note that since analyse_ackseq processes the acknowledgement number stream, `source_filter` needs to be specified in the opposite direction as for other tasks. For example, if we want to plot the throughput for a source 172.16.10.60 we set `source_filter=S_172.16.10.60_*`, but to plot the acknowledged bytes for this source we have to use `source_filter=D_172.16.10.60_*`.

1) Choosing between bytes and dupACKs: The parameter `dupacks` allows to choose whether acknowledged bytes (`dupacks`=0, default) or dupACKs (`dupacks`=1) are plotted.

2) Examples: The following shows an example of how to use the task to plot per burst acknowledged bytes omitting the first burst (assuming the gaps between bursts are at least 1 second long):

```
> fab analyse_ackseq:test_id=
20131206-102931_tcp_newreno,out_dir=./results,
burst_sep=1.0,sburst=2,eburst=0
```

K. anlyse_dash_goodput – Plot DASH-like client goodput over time

The analyse_dash_goodput task allows to plot the goodput for DASH-like traffic over time. The plot is based on data from the httperf log files of the DASH-like clients (named <test_id>_httperf_dash.log.gz). The task has the common parameters `etime`, `lnames`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. Note that `ts_correct` has no effect for this task as the timestamps are not taken from log files, but are generated based on the nominal request times given by the configuration of the DASH-like client.

1) Extracting data for specific experiments: By default the task extracts data from all client log files for all test IDs specified. However, if `dash_log_list` is specified the task extracts the data from the log files listed in a text file (the value of `dash_log_list` is the file name). This allows to explicitly list which dash client logs are used, and these files can also be from experiments with different test IDs. The format of the log list file is one file name per line. The paths do not need to be specified, as TEACUP will automatically find the files assuming they are in a sub directory below the fabfile.py directory.

Note that the number of legend names specified with `lnames` must be equal to the number of files names specified in the log list file (if `dash_log_list` is used) or equal to the number of log files with the specified test ID (if `dash_log_list` is not used).

2) Plot nominal goodput: By default analyse_dash_goodput will also plot the nominal goodput, i.e. goodput according to the DASH rate specified in the traffic generation configuration. The NO_NOMINAL environment variable can be used to tell plot_dash_goodput to not plot this. If NO_NOMINAL=1 the nominal goodput will not be plotted. Note that the task can only plot a maximum of 11 data series in a single graph.

3) Examples: The task can be used as follows:

```
> fab analyse_dash_goodput:
dash_log_list=dash_logs.txt,out_dir=./results/,
lnames="newreno;cdg;vegas"
```

L. analyse_goodput – Plot TCP goodput over time

This task plots goodput for TCP flows. This task plots the total or per-flow goodput over time. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the `total_per_experiment_parameter`.

Note that this task is work in progress. Currently, the goodput is computed based on the acknowledged bytes over time (extract_ackseq). This has the side effect that `source_filter` needs to be specified in the opposite direction as for other tasks (see Section IV-J). In the future it would be better to compute the goodput from the TCP sequence numbers directly.

1) Examples: The following command generates a graph of total goodput for the source 172.16.11.60:

```
> fab analyse_goodput:test_id=
20131206-102931_dash_2000_tcp_newreno,
source_filter=D_172.16.11.60_*,
total_per_experiment=1
```

M. analyse_incast – Plot response times over time (incast)

The task analyse_incast can be used to plot response times for incast experiments over time. It plots the response times as reported by httperf (httperf log files are present if incast traffic was generated). The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`.

It also has the burst separation parameters `burst_sep`, `sburst`, and `eburst` (explained in Section III-D). These allow to select only a subset of the queries, aka bursts, to be plotted. `sburst` specified the first burst included in the plot (starting from index of 1) and `eburst` specifies the last burst included in the plot. By default, the plot will include the response time of all bursts. Note, that `sburst` and `eburst` are currently only supported for the default case of `tcpdump='0'` (see below).

1) Filtering on flows: Note that for analyse_incast flows are bidirectional, and the source of an incast flow is always the querier and the destination of an incast flow is always one of the responders. This needs to be taken into account when using `source_filter`. For example, to filter the response times for responder 172.16.11.61 one needs to specify `source_filter=D_172.16.11.60_*`.

2) Boxplots instead of lines: The task has a `boxplot` parameter. If this parameter is set to '1', instead of plotting one line for the response times of each flow, a boxplot is generated for each point in time that captures the distribution of response times over all flows (default is '0').

3) Selecting data source: By default analyse_incast extracts the data from the httperf log files. Alternatively, if `tcpdump` is set to '1' the task will extract the response times from the tcpdump files collected at the querier (time between seeing the GET and the last packet of the response). If `tcpdump` is set to '1', the parameter `query_host` must be set to the host name of the querier (as specified in the config file).

4) Slowest response only: The task has a parameter `slowest_only` (default is '0'). If `slowest_only` is set to '1', instead of plotting separate data for each flow, the task will only plot the slowest response time over all flows for each burst. If `slowest_only` is set to '2', instead of plotting separate data for each flow, the task will only plot the time between the first request sent and the time the last response was finished for each burst (default is '0').

5) Examples: The following shows an example where we plot the slowest response time for each burst based on the times in the httperf logs:

```
> fab analyse_incast:
test_id=20140704-181632_incast_tcp_newreno,
,out_dir=./results/, slowest_only='1'
```

The following shows an example where we plot the response times based on tcpdump data (assuming testhost1 was the querier):

```
> fab analyse_incast:
test_id=20140704-181632_incast_tcp_newreno,
,out_dir=./results/, tcpdump='1',
query_host=testhost1
```

N. analyse_incast_iqtimes – Plot inter-query times (incast)

This task plots the times between requests sent by the querier (for checking whether requests were sent close together in time). The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. It also has the parameter `query_host` that must be used to specify the querier (name as specified in TPCONF_hosts) and the parameter `burst_sep` that must be used to specify the time between request bursts (by default this is set to 1 second).

1) Definition of inter-query time: By default inter-query time is defined as the time difference between a request and the first request in the same burst of requests. This allows to view how far query times are spread within each burst. To get a better idea of the timing jitter, the parameter `diff_to_burst_start` can be set to '0', to compute the time differences between each request and the previous request within the same burst.

2) No grouping by responders: By default response times are extracted and plotted separately (in different colours) for each responder. The parameter `by_responder` can be set to '1' to extract only one file with all response times and plot the response time for all responders in the same colour.

3) Cumulative response times: By default inter-query times are plotted against the time they occurred. The task also supports a cumulative plot mode to be able to analyse whether inter-query times significantly differ between different responders or different experiments. Setting the parameter `cumulative` to '1' means the inter-query times will be plotted in a cumulative fashion. Instead of each y-value being the inter-query time, each y-value will be either the current inter-query time plus the duration of all previous bursts (if `by_responder`=0) or the current inter-query time plus all previous inter-query times (if `by_responder`=1). Note that if `cumulative` is set to '1', `diff_to_burst_start` should be set to '1' (the default), otherwise the resulting graph is hard to interpret.

4) Examples: The following command plots the inter-query times as computed in different colours for each host (assuming the querier was testhost1):

```
> fab analyse_incast_iqtimes:
test_id=20140704-181632_incast_tcp_newreno,
,out_dir=./results/, query_host=testhost1,
cumulative=0
```

The following command plots the inter-query times in a cumulative fashion separately for each responder (again assuming the querier was testhost1):

```
> fab analyse_incast_iqtimes:
test_id=20140704-181632_incast_tcp_newreno,
,out_dir=./results/, query_host=testhost1,
cumulative=1,by_responder=1
```

O. analyse_pktloss – Plot packet loss rate

This task plots the packet loss rate in percent for emulated FPS game traffic flows. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `test_id`, `ts_correct`, `ymin`, and `ymax`. Packet loss is computed from the raw data by computing the percentage of lost packets for each time window. The environment variables AGGR_WINDOW_SIZE and AGGR_INT_FACTOR can be used to control the window size and oversampling.

1) Examples: The following command plots the packet loss rate for two game traffic flows originating from the same source:

```
> fab analyse_pktloss:
test_id=20141003-111821_game_traffic_aqm_pfifo,
,out_dir=./results/,
source_filter="S_172.16.10.2:10000;
S_172.16.10.2:10001"
```

P. analyse_cmpexp – Plot metric depending on experiment variables

The task analyse_cmpexp allows to plot the metrics 'throughput', 'spprtt', 'tcprtt' (unsmoothed/ERTT), 'cwnd', 'tcpstat', 'ackseq', 'restime' and 'iqtime' depending on the different experiment parameters for different selected flows. It can show the metric distribution as boxplots (default), or plot the mean or median. The task has the standard parameters `etime`, `lnames`, `min_values`, `omit_const`, `out_dir`, `out_name`, `pdf_dir`, `replot_only`, `source_filter`, `stime`, `ts_correct`, `ymin`, and `ymax`. However, a few of these parameters are handled slightly differently (explained below).

1) Specifying test IDs: The task has no `test_id` parameter. The `exp_list` parameter allows to specify a text file containing the list of experiments as for analyse_all (one experiment ID per line), making it possible to precisely select which combinations of parameters should be considered. For example, we can remove certain parameter values by removing all test IDs with these values from the list passed via `exp_list`. Note that all experiments to be compared must have the same test ID prefix and the same parameters varied.

2) Specifying the test ID prefix: Prior to version 0.9 TEACUP identified which variables were used and what their names are in the log file names from the config.py file (TPCONF_vary_parameters and TPCONF_parameter_list). This meant that at the time of running analyse_cmpexp, the TPCONF_vary_parameters in config.py had to be the configuration used for the experiment(s) specified in `exp_list`.

Since version 0.9 TEACUP identifies which variables were used and what their names are from the experiment file names directly and does not need to access config.py. In order to reliable detect the param-

eter names, the user must specify the test ID prefix format (as regular expression) using the parameter `test_id_prefix`. By default the parameter is set to '[0-9]{8}\-[0-9]{6}_experiment' (to match the default `YYYYMMDD_HHMMSS_experiment` format for test ID prefixes).

3) Intermediate files and final graphs: Intermediate data must already exist from previous executions of extract or analyse tasks, or analyse_cmpexp will trigger extraction of the data itself using the appropriate extract task(s). This extraction process is controlled by the parameter `res_dir`.

If `res_dir` is set and points to the existing intermediate files[6], analyse_cmpexp will search for and use the existing files. If files are missing analyse_cmpexp will abort. If `res_dir` is an empty string or unspecified, analyse_cmpexp will run the necessary extract tasks and the extracted data will be placed in `out_dir` (if set) or in the experiment directory (default). If you want analyse_cmpexp to explicitly (re-)run the data extraction, leave `res_dir` unset and (optionally) set `out_dir` with the desired destination for the extracted intermediate files.

4) Controlling which data to plot: The `variables` parameter can be a semicolon-separated list of variable names (names as used in the log file names) with associate values (separated by an equal sign). This provides a simple filter, as only experiments are considered where the variable(s) had the value(s) specified. Note that the equals (=) must be escaped with backslashes, otherwise Fabric will parse these.

The parameter `source_filter` works as explained in Section III-B5 and must be specified to control for which flows the metrics will be plotted.

5) Plot type: The `metric` parameter specifies the metric to plot: 'throughput' (default), 'spprtt', 'tcprtt' (unsmoothed/ERTT), 'cwnd', 'tcpstat', 'ackseq', 'restime' and 'iqtime'. The `ptype` parameter specifies the plot type: 'box' (default), 'mean', or 'median'.

6) Plot scaling, legend and labeling: The `ymin` and `ymax` parameters allow to specify custom minimum and maximum values for the y-axis (as for the other analyse tasks). The `lnames` parameter allows to specify the legend names used as list of semicolon-separated strings (as for the other analyse tasks). Note, the number of legend strings must the same as the number of source filters. By default the legend names are the source filters specified.

The `stime` and `etime` parameters allow to plot data for selected time windows only. In contrast to the other tasks this is not zooming (as there is no time axis), but all the data series are actually filtered to only contain values from inside the time window prior to plotting the data.

Currently it is not possible to reorder the different parameters for plotting other than by generating a custom experiment ID list. The default order is the order specified in the config (which is the same as the order in the file names). By default the x-axis labels contains all variable parameters, even the ones that had only one value (and were de-facto constant). The Boolean parameter `omit_const_xlab_vars` allows to automatically exclude constant variables from the x-axis labels. If set to '1' any variables that always had the same value in all experiments do not appear in the x-axis labels.

7) Grouping flows: By default analyse_cmpexp groups experiments by traffic flow, meaning each group in the plot is for one traffic flow, identified by a unique flow tuple (source IP, source port, destination IP, destination port), regardless of the series of experiments (test ID prefix) in which the traffic was produced. The Boolean parameter `group-by-experiment` allows to group by test ID prefix instead by setting `group-by-experiment=1`. With this each group in the plot relates to a particular series of experiments (identified by a test ID prefix) and the actual flow tuples can differ for different test ID prefixes. Of course in this case the number of (filtered) flows must be the same for all series of experiments and also the flows must be comparable across different series, i.e. the same type of test traffic was used in all series of experiments.

8) Merging data: The parameter `merge_data` specifies whether to merge the extracted data over all flows/responders per experiment. By default (`merge_data=0`) data for each flow/responder is plotted as separate box/median/mean. If `merge_data=1` only one box/median/mean will be plotted for each experiment. Note, TEACUP simply merges all types of data – the user must decide whether merging actually makes sense.

9) Metric specific parameters: The task has a number of metric specific parameters that are passed to the appro-

[6]E.g. The location given by the `out_dir` parameter to the previously run extract or analyse.

priate extract functions and/or select the data columns in the interim data files needed.

The parameter `link_len` defines whether to use IP layer packet length or link layer packet length for metric 'throughput' (see Section III-C). The parameter `smoothed` defines the type of RTT for metric 'tcprtt' (see Section IV-G). The parameter `stat_index` defines the statistic to be extracted and plotted for metric 'tcpstat' (see Section III-G). The parameter `dupacks` defines whether to plot acknowledged bytes or dupACKs for metric 'ackseq' (see Section IV-J), and the parameter `cum_ackseq` specifies whether to plot the cumulative values (`cum_ackseq=1`) or averages per time window (`cum_ackseq=0`). The parameters `sburst` and `eburst` can be used to select the bursts for metrics 'ackseq' and 'restime'. The parameter `slowest_only` can be used to plot the slowest response time for each burst, or the time between the first request sent and the last response finished (see Section III-K).

10) Response time special modes: If metric is set to 'restime', the parameter `res_time_mode` controls special plot modes for response time plots. If `res_time_mode` is set to 0 the plot is generated as normal.

If `res_time_mode` is set to 1 in addition to the observed response times the nominal response time is plotted as points. This only works if the parameters 'incSz' (block size), 'down' (bandwidth), and 'responders' (number of responders) are present in all the interim data file names.[7] The nominal response time is computed as: block_size * number_of_responders / (bandwidth / 8).

If `res_time_mode` is set to 2 *and* `ptype` is set to 'median' or 'mean', the ratio of observed median/mean response time and nominal response time is plotted. This only works if the nominal response time can be computed as described above.

11) Environment variables: The plotting can be further controlled by task-specific environment variables.

The variable OUTLIER_QUANT removes any points in the lowest OUTLIER_QUANT and highest OUTLIER_QUANT quantiles. For example, specifying OUTLIER_QUANT=0.01 will remove all data points that fall in the <0.01 quantile and all data points that fall in the >0.99 quantile.

[7]We assume responder to querier traffic was considered the 'downstream' direction during the actual TEACUP experiment

Setting NICER_XLABS=1 will modify the plotting of x-axis labels so that variable names are plotted once (on the left side) and only the variable values will be plotted at each tick. The environment variables YMAX_INC, AGGR_WIN_SIZE, and AGGR_INT_FACTOR can also be used and work as explained in Section IV-C.

The variable DIFF=1 can be set to convert cumulative data into non-cumulative data. Effectively the data vector of non-cumulative values is generated by taking the differences of subsequent values of the original cumulative data vector.

The variable NO_BARS=1 can be set to plot points instead of the default bars when `ptype` is set to 'median' or 'mean'.

12) Examples: The following command shows an example, where we plot TCP RTT as boxplots:

```
> fab analyse_cmpexp:exp_list=
myexp_list.txt,res_dir="./results/",
variables="run\=0", source_filter=
"D_172.16.10.2_5001;D_172.16.10.3_5006",
metric=tcprtt, lnames="CDG;Newreno"
```

Q. analyse_2_density – Plot two metrics against each other depending on experiment variables

The analyse_2_density task can create 2-dimensional density plots or ellipse plots allowing to explore the relationship between two metrics depending on different experiment parameter combinations. It supports the same metrics as analyse_cmpexp: 'throughput', 'spprtt', 'tcprtt' (unsmoothed/ERTT), 'cwnd', 'tcpstat', 'ackseq', 'restime' and 'iqtime'. The task has the standard parameters `out_dir`, `replot_only`, `source_filter`, `ymin`, `ymax`, `lnames`, `out_name`, `min_values`, `ts_correct`, and `pdf_dir`. A few of these parameters are handled slightly differently (explained below).

Please note that this task is somewhat experimental and still under development.

1) Specifying test IDs: The task has no `test_id` parameter. The `exp_list` parameter allows to specify a text file containing the list of experiments as for analyse_cmpexp. Also, as for analyse_cmpexp, the parameter `test_id_prefix` must be specified in case the test ID prefix used is not the standard prefix.

2) Intermediate files and final graphs: Intermediate data must already exist from previous executions of extract or analyse tasks, or analyse_2d_density will

trigger extraction of the data itself using the appropriate extract task(s). This extraction process is controlled by the parameter `res_dir` as described for analyse_cmpexp.

3) Controlling which data to plot: The `variables` parameter can be a semicolon-separated list of variable names (names as used in the log file names) with associate values (separated by an equal sign). This provide a simple filter, as only experiments are considered where the variable(s) had the value(s) specified. Note that the equals (=) must be escaped with backslashes, otherwise Fabric will parse these.

The parameter `source_filter` works as explained in Section III-B5 and must be specified to control for which flows the metrics will be plotted.

4) Plot type: The `xmetric` parameter specifies the metric for the x-axis and the `ymetric` parameter specifies the metric for the y-axis. Both parameter values must be selected from the set of available metrics.

5) Axis limits: Instead of the usual parameters `stime`, `etime`, analyse_2d_density has the parameters `xmin`, `xmax` for limiting the x-axis values. Note that due to the internals of a different plotting function used (R ggplot2), the parameters `xmin`, `xmax`, `ymin`, `ymax` do not only zoom-in on the data, they actually filter out any data points outside of the specified limits.

6) Grouping: The parameter `group_by` can be used to specify the different groups. Each group will be plotted in a different colour and will appear as a separate entry in the legend. The value must be set to the name of a parameter varied during the experiment (the name of the variable in the file names). By default `group_by` is set to 'aqm'.

7) Merging data: The parameters `merge_xdata` and `merge_ydata` specify whether to merge the extracted data over all flows/responders per experiment for the x-axis and/or y-axis. If set to '0' data is extracted per flow/responder and for the plot a single flow needs to be selected with `source_filter`. If set to '1' `source_filter` can be used to select multiple flows, i.e. by using a wildcard for the port number, and the data for all flows will be merged. Note that TEACUP simply merges all types of data, it is the user's responsibility to decide whether it makes sense to merge the data.

8) Metric specific parameters: The task has a number of metric specific parameters that are passed to the extract functions and/or select the data columns in the interim data files needed. It has all the metric-specific parameters of analyse_cmpexp, except `stat_index`. Instead of `stat_index` analyse_2d_density has the parameters `xstat_index` and `ystat_index` to specify the indexes for both axes. `xstat_index` must be set if `xmetric=tcpstat` and `ystat_index` must be set if `ymetric=tcpstat`.

9) Environment variables: The plotting can be further controlled by task-specific environment variables.

The variable OUTLIER_QUANT removes any points in the lowest OUTLIER_QUANT and highest OUTLIER_QUANT quantiles. For example, specifying OUTLIER_QUANT=0.01 will remove all data points that fall in the <0.01 quantile and all data points that fall in the >0.99 quantile.

The variable ELLIPSE controls whether a 2D density plot or a ellipse plot will be created. By default 2D density plots are created (ELLIPSE=0). To create ellipse plots instead set ELLIPSE=1. For 2D density plots the PDF files have a '2d_density_plot.pdf' extension whereas for ellipse plots the PDF files have a 'ellipse_plot.pdf' extension.

The variable BINS can be used to set the number of bins for the 2D density estimation (only effective if ELLIPSE=0). By default the number of bins is set to 4. The variable ADD_RAND can be used to add randomness to the data (noise is drawn from a uniform distribution within the interval of (-0.5, 0.5)). By default no randomness is added. If 2D density should be plotted (ELLIPSE=0) and the data consists of discrete values, the resulting graphs can look better with added randomness (ADD_RAND=1).

The variable MEDIAN can be set to 1 to plot the median for each group. By default the task does not plot the median. The variable NO_LEGEND can be set to 1 to suppress the legend, which by default is shown to the right of the graph.

By default only the estimations or ellipses are plotted. The variable SCATTER can be set to 1 to superimpose a scatter plot of all points on the 2D density or ellipse plot.

The variable DIFF=1 can be set to convert cumulative data into non-cumulative data. Effectively the data vector of non-cumulative values is generated by taking the dif-

ferences of subsequent values of the original cumulative data vector.

10) Examples: The following example command plots a 2D density plot for traffic going to a particular destination (172.16.11.68 port 5000) with throughput as metric on the x-axis and TCP RTT as metric on the y-axis, and three different groups based on the AQM method that was used (FIFO, Codel, PIE):

```
> ADD_RAND=1 fab analyse_2d_density:
exp_list= myexp_list.txt,res_dir="./results/",
variables="run\=0",source_filter=
"D_172.16.11.68_5000",xmetric=throughput,
ymetric=tcprtt,link_len=1,
group_by=”aqm”,lnames="fifo;codel;pie"
```

The following example command is the same as the previous, except that now we plot an ellipse plot instead of a 2D density plot:

```
> ELLIPSE=1 ADD_RAND=1
fab analyse_2d_density:
exp_list=myexp_list.txt,res_dir="./results/",
variables="run\=0",source_filter=
"D_172.16.11.68_5000",xmetric=throughput,
ymetric=tcprtt,link_len=1,
group_by=”aqm”,lnames="fifo;codel;pie"
```

R. Limits on series/groups per graph

Currently the underlying plot function for all time series graphs, such as analyse_throughput, analyse_spp_rtt, analyse_cwnd, analyse_tcp_rtt, and analyse_tcp_stat, can only plot 12 different time series on a single graph. If the number of data series to plot is larger than 12, multiple graphs are generated with a _<graph_number> at the end of each file name to indicate the number of the graph in the series of graphs (graph number starting from 1).

The number of groups (corresponding to legend entries) that can be plotted in one graph with analyse_cmpexp and analyse_2d_density is limited to 12.

V. Customise Plotting

The previous section explained some environment variables that a user can use to customise the plotting. Internally, TEACUP also sets a variety of other environment variables and passes them to the plotting script. The plotting script reads these environment variables and generates the plot based on their values. For example, TEACUP sets a variable YLAB with the value being the label string plotted on the y-axis. All existing environment variables are explained at the beginning of each default plot script and the parameters passed are also explained in the TEACUP code calling the script (analyse.py).

All analyse tasks have a parameter called `plot_params` that can be used to override any of the environment variables. This allows the user to customise things for which there is no TEACUP task parameter, such as the y-axis label. For example, we can set environment variables to plot the throughput in Megabits per second instead of the default kilobits per second as follows (note that any '=' must be escaped, so they are not interpreted by Fabric):

```
> fab analyse_throughput:test_id=
20131206-102931_dash_2000_tcp_newreno,
plot_params="YLAB\=\"Throughput (mbps)\"
YSCALER\=0.000008"
```

Furthermore, all analyse tasks have a parameter called `plot_script` that can be used to specify the script used for plotting. If this parameter is set, the specified script will be executed instead of the default script. The default plotting scripts are R scripts located in the teacup source directory, so by default `plot_script` looks like this:

```
R CMD BATCH --vanilla
    <TPCONF_script_path>/<R_script>
```

The simplest way of customising things is to create a modified R script (say <my_R_script>) based on the original plotting script (<R_script>).Then one can simply set `plot_script` to:

```
R CMD BATCH --vanilla
    <TPCONF_script_path>/<my_R_script>
```

However, one could also specify an entirely different script written in an entirely different scripting language.

The plot script gets its input from TEACUP via the environment variables. The plot script is called with a single command line argument, which is a file name for logging output from the script, e.g. log messages, warning, or errors. The output file is only created if TPCONF_debug_level is greater than 0 (see [4]). The name of the log file is currently hard-coded and corresponds to the name of the default R script used for plotting.

VI. Utility Functions

This section describes some analysis and plotting related utility functions TEACUP provides. For example, TEACUP provides some functionality to combine multiple graphs into a single graph, so results can be compared more easily.

A. Combining graphs

There are two simple shell scripts to combine different result graphs (different PDF files) on a single page for easy comparison.[8] The assumption is that the TCP congestion control algorithm is the innermost parameter varied (i.e. the last parameter in the file name).

The script `tcp_comparison.sh` can be used to combine throughput, RTT, CWND or SPP RTT graphs for up to four different TCP congestion control algorithms on one page. For example, the following command creates four PDFs, each with four graphs for each TCP congestion control algorithm (assuming the test ID is 20131220-182929_del_<delay>_tcp_<tcp_algo>):

```
> tcp_comparison.sh 20131220-182929_del_10_tcp
test
```

The output files are:

```
test_cwnd_different_tcps.pdf
test_spprtt_different_tcps.pdf
test_tcprtt_different_tcps.pdf
test_throughput_different_tcps.pdf
```

The second script `tcp_comparison_allinone.sh` creates the same PDFs as above but in addition creates one single-page PDF with all throughput, RTT, CWND and SPP RTT graphs for up to four TCP congestion control algorithms (up to 16 graphs in total). It can be used as follows:

```
> tcp_comparison_allinone.sh
20131220-182929_del_10_tcp test
```

The output files are:

```
test_cwnd_different_tcps.pdf
test_spprtt_different_tcps.pdf
test_tcprtt_different_tcps.pdf
test_throughput_different_tcps.pdf
test_different_tcps_allinone.pdf
```

The two scripts are not strictly limited to combining the results for different TCP congestion control algorithms. They can be used with any last parameter as long as there are no more than four values. However, part of the output file names is hard-coded.

To combine graphs with more flexibility one can use the script `combine_graphs.sh`, which is used by the scripts for TCP comparison. The script allows to combine an arbitrary number of graphs one one page. For example, if we want to compare the CWND graphs for two different delay values and four different TCP congestion control algorithms we can do this with the following command:

```
> combine_graphs.sh -c 4x2 -o test.pdf
`find . -name
20131220-182929_del_*_tcp_*cwnd*.pdf | sort`
```

Here the `find` command is used with wild-cards to select the PDF files to combine on one page and the `-c` parameter is used to specify that the graphs are organised in a layout with 2 rows and 4 columns. Instead of using `find` one can specify all file names explicitly. This allows for full control of the location of graphs on the single page, but is cumbersome if there are many graphs. Note that `combine_graphs.sh` puts the graphs on the page row after row, i.e. for a 4x2 layout the first four graphs go in the first row, the second four graphs go in the second row, etc.

VII. Use Case – Analysing an Incast Experiment

In this use case we analyse the data from incast experiments carried out with TEACUP where one Linux querier sent quasi-simultaneous queries to multiple Linux responders in fixed five-second time intervals. Over successive experiments we have increased the number of responders from 4 to 12. Upstream/downstream bandwidth was 50 Mpbs, emulated delay was 1 ms, the buffer size of the bottleneck was 120 packets, and the size of the responses was 512 kB. TCP was Linux Cubic and the AQM was varied.

A. Analysing throughput and RTT

First, we use analyse_throughput to plot throughput. We filter so that only the flows from the responder are included as the flows from the querier are mainly ACKs after the initial query (using source_filter). We reduce the point size in the plot with POINT_SIZE and tweak the window for throughput calculations with

[8] These scripts require that the pdfjam package is installed (on FreeBSD this can be installed from the ports tree).

AGGR_WIN_SIZE and AGGR_INT_FACTOR (the default window settings are inappropriate for short sub-second traffic bursts). We use plot_params to change from the default kbps to Mbps. We only plot the first two bursts (by setting etime=6), since all bursts look relatively similar and we are interested to see more details for single bursts. We set link_len=1 to plot throughput including link-layer headers. The following shows the command:

```
> POINT_SIZE=0.25 AGGR_WIN_SIZE=0.05
AGGR_INT_FACTOR=10 fab analyse_throughput:test_id=
20150218-135735_experiment_aqm_pfifo_responders_4,
out_dir=../result,link_len=1,source_filter=
"S_172.16.11.61_80; S_172.16.1
1.62_81;S_172.16.11.63_82;S_172.16.11.64_83",
etime=6,plot_params='YLAB\="Throughput (mbps)"
YSCALER\=0.000008'
```

Next, we use analyse_rtt to plot the RTT. The parameters used are a subset of the above:

```
> POINT_SIZE=0.25 fab analyse_rtt:test_id=
20150218-135735_experiment_aqm_pfifo_responders_4,
out_dir=../results,source_filter="S_172.16.11.61_80;
S_172.16.11.62_81;S_172.16.11.63_82;
S_172.16.11.64_83",etime=6
```

Figure 2 shows the throughput and RTT figures for the experiment with 4 responders. The throughput graph shows that the throughput varies depending on the responder. For example, in the first part of the first burst responders 1, 3 and 4 have roughly 12–15 mpbs whereas responder 2 has 7–10 mbps, and only when the first three responders are finishing responder 2 is catching up. The second burst is even more unequal with responders 1 and 3 getting a higher share of the bandwidth at the start, and responder 2 and 4 catching up later. The RTT estimated by SPP reaches up to 29 ms with is the theoretical limit (1500 byte packet size multiplied with 120 buffer slots divided by 50 mbps).

B. Analysing time between queries

For incast experiments the goal is that the querier in each round/burst sends out the queries as closely together in time as possible. We now investigate the time gaps between queries for the experiment with 4 responders. We use the analyse_incast_iqtimes task to plot the times between a query sent to a responder and the first query sent in each burst separately for all responders (by setting by_responder=1 and cumulative=1). The querier must be specified with query_host, in this case it was host newtcp20. The following shows the command:

```
> POINT_SIZE=0.5 fab analyse_incast_iqtimes:test_id=
20150218-135735_experiment_aqm_pfifo_responders_4,
out_dir=../results,query_host=newtcp20,
by_responder=1,cumulative=1
```

Figure 3 shows the plot of inter-query times for the experiment with 4 responders. It shows that for the first 7 bursts the requests were sent very closely together within 2 ms. However, from burst 8 onwards, the requests were sent more and more spread out with the request to responder 1 falling behind more and more. Still, even for the last burst all requests were sent within 9 ms.

C. Analysing response times

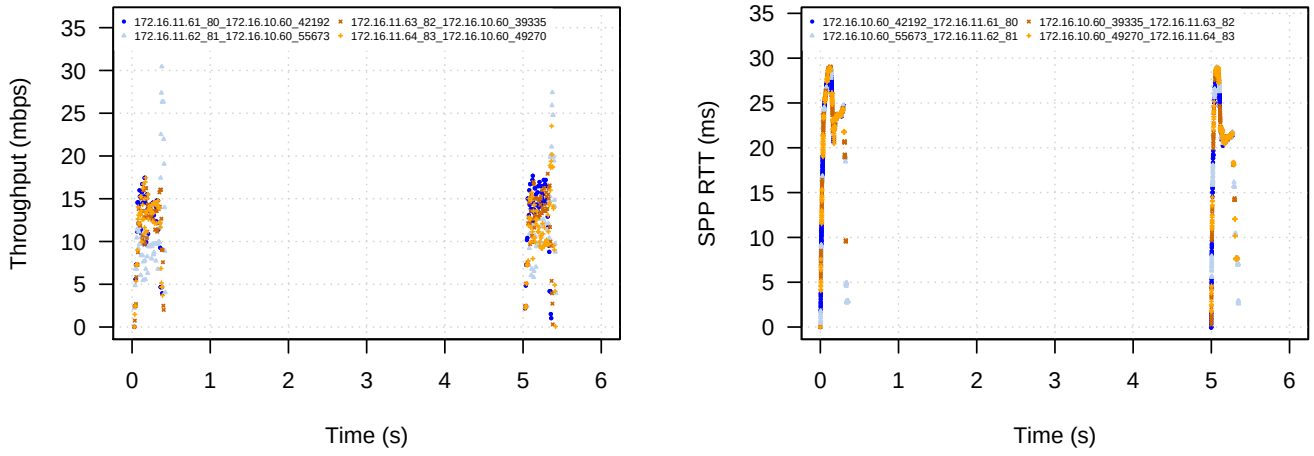
Now let us have a look at the actual response times for the experiment with 4 responders. To plot the response times over time for each responder separately, we use the analyse_incast task without any parameters (except that we use out_dir to put the interim data files and the PDF files in the desired directory):

```
> POINT_SIZE=0.5 fab analyse_incast:test_id=
20150218-135735_experiment_aqm_pfifo_responders_4,
out_dir=../results
```

Figure 4 shows the response times over time for the different responders for the experiment with 4 responders. Response time varies for the different responders but in general is between 0.3 and 0.35 seconds for all bursts. For the first two bursts we can see that response times are exactly as expected from the throughput graphs, e.g. in burst one responder 2 has the highest response time since it has the lowest throughput initially.

D. Analysing acknowledged bytes and dupACKs

To plot the acknowledged bytes and dupACKs we use the analyse_ackseq task. We set burst_sep to 1 second and select only the first and second burst with sburst=1 and eburst=2. We select the ACK flows corresponding to the data flows from the four responders using source_filter. Since the ACK flows travel in the opposite direction as the data, our source filters from Section VII-A (which filtered the data coming *from* the various responders) now become filters that filter the ACKs going *to* the various responders.



(a) Throughput

(b) RTT

Figure 2: Throughput and RTT statistics for experiment with 4 responders

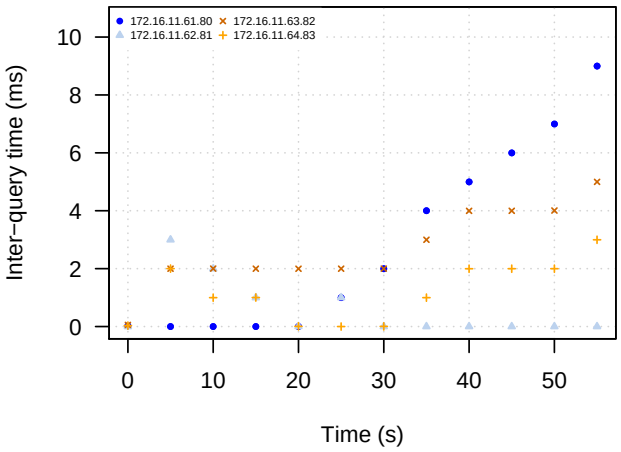


Figure 3: Incast inter-query times for experiment with 4 responders

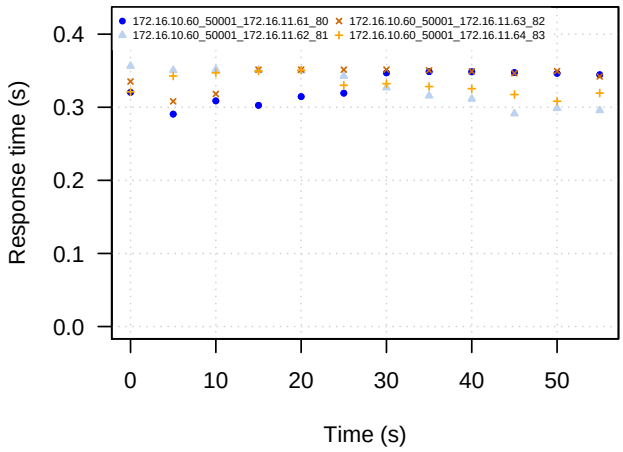


Figure 4: Incast response times for experiment with 4 responders

```
> POINT_SIZE=0.25 fab analyse_ackseq:test_id=
20150218-135735_experiment_aqm_pfifo_responders_4,
out_dir=../results,source_filter=
"D_172.16.11.61_80;D_172.16.11.62_81;
D_172.16.11.63_82;D_172.16.11.64_83",
burst_sep=1,sburst=1,eburst=2
```

Figure 5 shows the acknowledged bytes per burst for each responder for the experiment with 4 responders. Again we can see that the download speed varies for the different responders with responder 2 being the slowest.

E. Response times versus number of responders

Finally, let us compare the response times for an increasing number of responders. To plot comparison graphs we use the analyse_cmpexp task. We create a file called explist.txt that contains the list of experiment IDs for all the experiments with different responders. The file has the following content:

```
20150218-135735_experiment_aqm_pfifo_responders_4
20150218-135735_experiment_aqm_pfifo_responders_5
20150218-135735_experiment_aqm_pfifo_responders_6
20150218-135735_experiment_aqm_pfifo_responders_7
20150218-135735_experiment_aqm_pfifo_responders_8
20150218-135735_experiment_aqm_pfifo_responders_9
20150218-135735_experiment_aqm_pfifo_responders_10
20150218-135735_experiment_aqm_pfifo_responders_11
20150218-135735_experiment_aqm_pfifo_responders_12
```

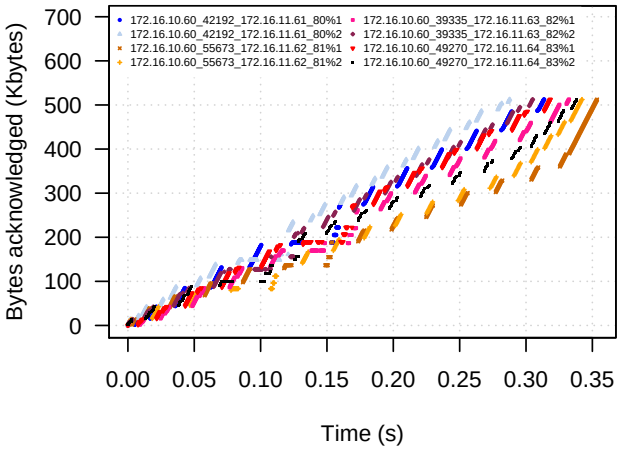


Figure 5: Acknowledged bytes per burst and responder for experiment with 4 responders

First, let us have a look at the first four responders. We specify source_filter to select the first four responders. We set metric to response time ('restime') and the type of plot to boxplot ('box'). The PDF file name prefix is set to "FIRST_FOUR" to get a file name we can identify later.

To make the x-axis look nicer we set NICER_XLABS=1 and omit_const_xlab_vars=1. We decided to change the first default colour, as the default dark blue makes the median lines inside the boxes harder to read (and as excuse to introduce the plot_script parameter). We created a slightly modified version of the default plot script named `my_plot_cmp_experiments.R`, in which the first colour is changed from the default dark blue to a lighter blue (by redefining the cols array defined in plot_func.R after the point where plot_func.R is included).[9] Then we instructed TEACUP to use the modified script for plotting with the plot_script parameter.

The command used for plotting is:

```
> NICER_XLABS=1 fab analyse_cmpexp:exp_list=
explist.txt,out_dir=../results,
source_filter="D_172.16.11.61_80;D_172.16.11.62_81;
D_172.16.11.63_82;D_172.16.11.64_83",
metric=restime,ptype=box,omit_const_xlab_vars=1,
out_name="FIRST_FOUR",plot_script="R CMD BATCH
--vanilla ./my_plot_cmp_experiments.R"
```

Figure 6 shows the response time distribution as boxplots for the first four responders over all the experiments (the boxplots are the distributions of the response times over all burst in each experiment). We can see that response time is increasing with the total number of responders as expected, but depending on the experiment and responder there variations.

Next, let us simply plot the response time distributions over all responders in each experiment. We specify source_filter to select all flows to the querier and instruct TEACUP to merge all flows in each experiment into one box instead of plotting a box for each flow (by setting merge_data=1). The other parameters used are the same as before.

The command used for plotting is:

```
> NICER_XLABS=1 fab analyse_cmpexp:exp_list=
explist.txt,out_dir=../results,
source_filter="S_172.16.10.60_*",merge_data=1,
metric=restime,ptype=box,omit_const_xlab_vars=1,
out_name="ALL",plot_script="R CMD BATCH --vanilla
./my_plot_cmp_experiments.R"
```

Figure 7 shows the response time distributions as boxplots, where each box shows the distribution of response times over all bursts of all responders in each experiment. The figures shows that response time is increasing with an increasing number of responders as expected. It also shows that the variance of the response times increases with an increasing number of responders.

VIII. Conclusions and Future Work

TEACUP is a Python-based software tool we developed to run automated TCP performance tests in a controlled testbed [4]. In this report we described TEACUP's functionality to extract statistics and plot graphs based on data collected during the experiments.

Acknowledgements

TEACUP versions prior to 1.0 were developed as part of a project funded by Cisco Systems and titled "Study

[9] Note that `my_plot_cmp_experiments.R` includes other R scripts, such as plot_func.R. The included scripts also need to be copied in the same directory as `my_plot_cmp_experiments.R` even if they are not modified. Alternatively, one needs to modify the include paths to point to the original scripts from the TEACUP distribution.

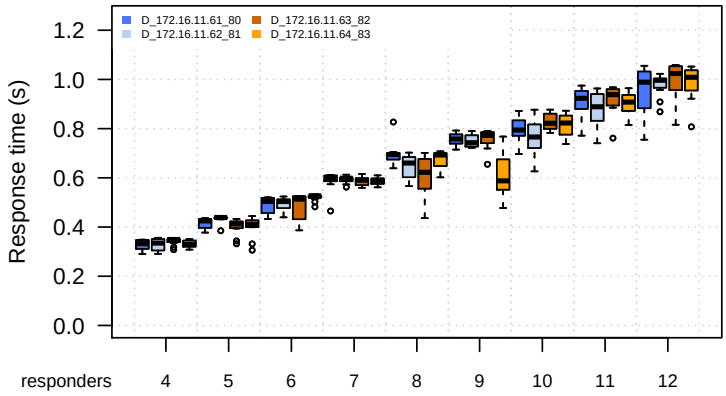


Figure 6: Response time distributions for the first four responders depending on the number of responders

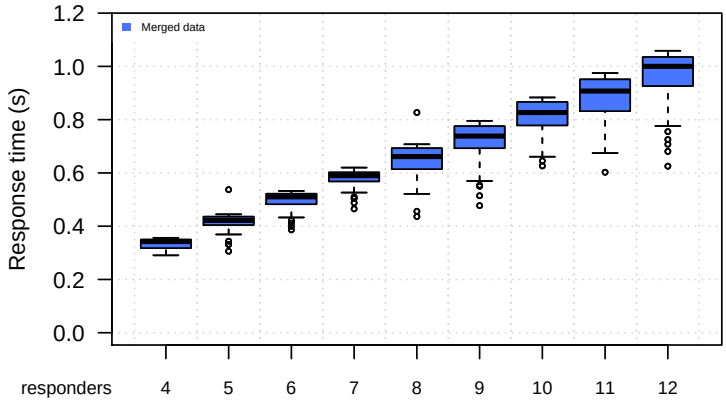


Figure 7: Response time distributions for all responders depending on the number of responders

in TCP Congestion Control Performance In A Data Centre". This was a collaborative effort between Swinburne University of Technology's Center for Advanced Internet Archectures and Fred Baker of Cisco Systems. These may be found at http://caia.swin.edu.au/tools/teacup/downloads.html.

Starting with TEACUP v1.0, development will be community supported and publicly hosted at http://sourceforge.net/projects/teacup/.

REFERENCES

[1] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011, pp. 345–360.

[2] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11, 2011, pp. 25:1–25:12.

[3] "Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," ISO, 2012, iSO/IEC 23009-1:2012. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623.

[4] S. Zander, G. Armitage, "TEACUP v1.0 – A System for Automated TCP Testbed Experiments," Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 150529A, 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf

[5] S. Zander and G. Armitage, "Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs," in *The 38th IEEE Conference on Local Computer Networks (LCN 2013)*, 21-24 October 2013.

[6] A. Heyde, "SPP Implementation," August 2013. [Online]. Available: http://caia.swin.edu.au/tools/spp/downloads.html

[7] D. Hayes, "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 100219A, 19 February 2010. [Online]. Available: http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf

[8] S. Zander, "TEACUP v1.0 – Command Reference," Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 150529C, 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529C/CAIA-TR-150529C.pdf

[9] L. Stewart, "SIFTR v1.2.3 README," July 2010. [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools/siftr-readme-1.2.3.txt

[10] M. Mathis, J. Semke, R. Reddy, J. Heffner, "Documentation of variables for the Web100 TCP Kernel Instrumentation Set (KIS) project." [Online]. Available: http://www.web100.org/download/kernel/tcp-kis.txt