Flent: The FLExible Network Tester

Toke Høiland-Jørgensen
Karlstad University
toke.hoiland-jorgensen@kau.se

We present a tool designed to make experimental evaluations of networks more reliable and easier to perform. This tool, called Flent, works by composing well-known benchmarking tools to, for example, run tests consisting of several bulk data flows combined with simultaneous latency measurements. Tests are specified in source code, and several common tests are included with the tool. In addition, Flent contains features to automate test runs, and to interactively plot and explore data collected from experiments.

1 Introduction

There can be many difficulties in running properly managed experimental evaluations of networks. Hence, oftentimes new network technologies are primarily (or even exclusively) evaluated by simulation. Using simulation can make sense because many difficulties disappear when running the experiment is as easy as launching a computer program. However, we wish to make the case that running actual experiments on real-world hardware is an important element of network research. And we believe that experimental evaluation should not be shunned because of the inherent difficulties in running the experiments; but rather that it is important to work towards increasing reliability and making it easier to run tests.

There are several reasons why it is valuable to run experiments instead of simulations. The main reason is also the most obvious: simulations are necessarily idealised, and may not be accurate. In particular, interactions between networking hardware, drivers and the operating system can give rise to effects that would otherwise be missed. Whether it is hidden buffers in the networking hardware, device drivers or the operating system network stack; hardware offloads that amplify quantification effects by making packets appear bigger to the algorithms deployed higher in the stack; or plain and simple bugs: real-world systems simply behave *differently* than the simulation might indicate.

Another reason to prefer experiments on real systems is the sheer pace of development of, especially, open source operating systems. Linux in particular has seen sweeping changes to its network stack over the last several years, in many aspects completely changing its behaviour. Having more scrutiny on this process by the academic community can be valuable in itself. But more importantly, these changes mean that the assumptions underlying new research needs to be checked against the actual systems running the internet.

We present a tool designed to work towards the goal of making testing more reliable and easier to carry out. This tool, called Flent[1], works by composing well-known benchmarking tools to (for example) run tests consisting of several bulk data flows combined with simultaneous latency measurements. Tests are specified in source code, and several common tests are included with the tool. In addition, Flent contains features to automate test runs, and to interactively plot and explore data collected from experiments.

The rest of this paper describes Flent in more detail. Section 2 elaborates on the difficulties of running experiments and Section 3 describes Flent and how it addresses these difficulties. Section 4 gives some examples of the kind of tests that can be run with the tool. Finally, Section 5 outlines some future development directions, Section 6 summarises related work and Section 7 concludes.

2 Difficulties when running experiments

This section outlines some key difficulties that we believe are important to address for successful real-world experimental work, and that we seek to solve in the Flent testing tool.

2.1 Coordinating different test tools

Many network benchmarking tools are single-purpose, or can only run one test at a time. Thus, many tests require running several instances of the same tool in concert. Similarly, running different tools at the same time can be necessary when a test scenario requires testing (e.g.) different types of traffic that cannot all be generated by the same tool.

Often, ad-hoc scripting is the tool of choice when combining test tools, but that can be error-prone and tedious, and usually results in duplication of effort between different test scenarios and deployments.

2.2 Reproducing experiments

Reproducing experiments is important for verifiability, both for the researcher herself, but also for independent

[1] Short for the FLExible Network Tester. Additionally, it is well known that 'flent' is the sound a network makes when subjected to rigorous testing that exposes the flaws (such as bufferbloat) preventing it from performing adequately.

reproduction by others. The more ad-hoc the test configuration and setup is, the harder this is. A common format to describe tests will significantly ease the process of sharing, and allows others to provide feedback. This can also facilitate the collaborative development of best practices for how to test different phenomena.

2.3 Managing test-bed configuration

Experiments often involve test-beds comprising several physical devices setup to emulate the desired network topology and characteristics, and the configuration of these devices must be managed. This includes correctly configuring network interfaces, applying the algorithm(s) under test, etc. Additionally, the configuration must be verifiable after the tests have run, so that it is possible to certify that a data set corresponds to a particular configuration.

This process can be error-prone, especially as the number of configuration parameters that vary between test runs increase. In addition, application of configuration can fail, either from human error or from (unchecked) failures in the application process, so automation is important in both application and subsequent collection.

2.4 Storing and analysing measurement data

As the number of experiments grow, storing the measurement data and relating it to the tested configurations becomes harder. This is exacerbated by the previous issue of coordinating several benchmarking tools with possibly different output formats. A standardised way is needed to manage these different benchmark tool outputs, and extracting the meaningful data points for further analysis. This also ties into the previous point on configuration management, in that the configuration must be matched to the stored data; and preferably this configuration information should not be lost as the data is further processed.

3 Running tests with Flent

Flent [2] is a testing tool developed specifically to address the difficulties mentioned in the previous section, while also being extensible to address other future use cases. It is a Python wrapper[2] around several other well-known network benchmark tools, most notably *Netperf* [3]. This section describes how Flent seeks to address each of the difficulties presented in the previous section.

3.1 Coordinating different test tools

Flent works by running one or more *tests*, each defined by a configuration file that specifies which benchmarking tools to run. Several tools can be run simultaneously, or in series, and dependencies can be specified between them (e.g., run one tool once another has finished). This mode of operation is inspired by the Unix philosophy of composing separate tools that each does one thing well. Flent simply adds coordination, and thus is able to leverage the capabilities of existing tools and achieve the same high confidence in the benchmark results afforded to well-known tools such as Netperf.

The output of each test tool is parsed, and the interesting data is stored in a common JSON-based format. This makes it easy to create composite tests comprising several different tools, and afterwards directly compare the data collected by the tools. A common example employed in many of the tests included with Flent is running one or more instances of Netperf to produce bulk flows, while simultaneously measuring the end-to-end latency by means of the regular `ping` command.

3.2 Reproducing tests

Each test defined is Flent is named and has a separate config file. This greatly aids reproducibility, as the named tests are available along with the source code, and can thus be referenced reliably. Additionally, this makes Flent tests vary versatile, since more extensive test suites can be composed of the available tests. This also allows Flent to work well with other tools that manage test setups: Anywhere there's a Python environment, the required underlying benchmarking tools and a network connection, Flent can run.

Indeed, running tests under varying conditions is a major goal of Flent. As such, care is taken to avoid external dependencies where possible, and to support different alternatives for test tools where appropriate. In particular, it is an explicit design goal to be able to run a test from, say, a regular laptop connected to a network that is not necessarily under the control of the experimenter (the "test my hotel wifi" scenario), as well as working in a fully instrumented testbed environment.

3.3 Managing test-bed configuration

Flent has built-in batch run capabilities, making it possible to specify a series of test runs to be run in sequence, while supporting inheritance and recursive expansion of variables and config sections to facilitate configuration reuse. By means of this facility, extensive test suites can be built from the named tests.

Flent does not include functionality to directly manage node configuration in a testbed. This is a deliberate design choice: we have found that in the small to medium-sized testbed environments targeted by Flent, simple scripts are the most flexible choice for configuring nodes. However, these scripts must run reliably before each test invocation, and any failures be detected detected. To achieve this, the batch feature simply provides a facility to run arbitrary commands before and after each test, and (optionally) abort the test if any of the commands fail. Combined with the metadata gathered by Flent at each test-run, this comprises an effective configuration management and verification facility.

[2] And indeed, Flent has until recently been named *netperf-wrapper*.

3.4 Storing and analysing measurement data

Flent automatically gathers metadata from the host running the test (and optionally from remote hosts via SSH), and stores the metadata in the data file along with the test data. This means that a single data file can capture a complete test run and be easily distributed or moved for further analysis.

Flent also contains an extensive analysis facility, which reads already produced test files and produces plots of the data. A GUI makes it easy to flip between plots of different test runs. Tests can define detail plots (such as the raw timeseries data of throughput during the test run), as well as aggregate plot types (CDFs, box plots, etc), and it is possible to show several test runs side by side, as well as to combine them into aggregate plots. Together, the interactive plotting features make for a powerful analysis tool in the exploratory phases. Additionally, the tool can also produce final high-quality graphs for publication: The example plots in the next section are produced by the built-in plotting facilities.

Many plot types are included, so data exploration can be done directly from the data files. Should this not be sufficient, it is also possible to export the data to other formats: there's a CSV export feature in Flent itself, and the JSON data format is readily parsable by other tools.

4 Examples of effective tests enabled by Flent

This section describes two example tests that have been developed in conjunction with the Flent test tool, and are included in the source distribution of the tool. They are the Real-Time Response Under Load (RRUL) test, and the RTT fairness test.

4.1 The RRUL test

The RRUL test was developed by the bufferbloat community [6] specifically to stress-test networks and weed out undesirable behaviour. It consists of running four concurrent TCP flows in each direction, while simultaneously running UDP and ICMP flows to measure latency. The goal is to saturate the connection fully, to better observe the behaviour in this scenario. This is valuable for exposing bufferbloat in particular, but also works well as a general stress-test of queue management schemes.

An example of the results that can be obtained with the RRUL test is shown in figure 1. This figure shows the result of running the RRUL test over a bottleneck link managed by a series of different queue management schemes. The plot shows the mean induced latency vs the mean TCP goodput for each of the queue management schemes, with the dots showing the median values and the ellipses showing the variance. This gives an overview of several test results, while other available plot types serve well in the exploration phase, or when examining details of the test performance.

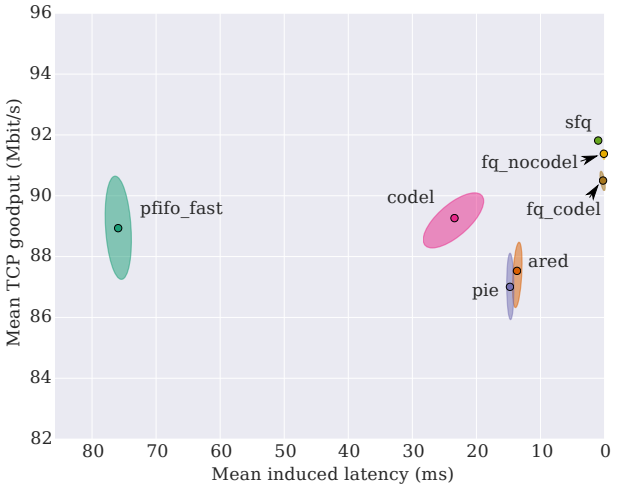

Figure 1: A bandwidth/delay plot of results of running the RRUL test. The dots show the median values and the ellipses the variance of the per-test-run mean goodput and mean induced latency.

4.2 RTT fairness measurements

The RTT-fairness test examines the RTT fairness properties of TCP. It is well-known that the TCP goodput is affected by the RTT [5], because the congestion control algorithm reacts to feedback that is on an order of the RTT.

The purpose of the RTT-fairness test is to evaluate whether the queue management schemes make this effect worse, or whether they help alleviate it. The test consists of running four concurrent TCP streams from the client to four different servers. Flent includes several variants of the RTT-fairness test that can be run against four arbitrary hosts, making it easy to examine the behaviour in any scenario where suitable test endpoints (with varying path characteristics) are available.

Figure 2 shows an example of the results against the same set of queue management schemes as the RRUL test above, run against four endpoints with path RTTs of 10, 50, 200 and 500 ms, respectively. The figure shows the mean goodput of each of the four TCP streams for each of the queue management schemes.

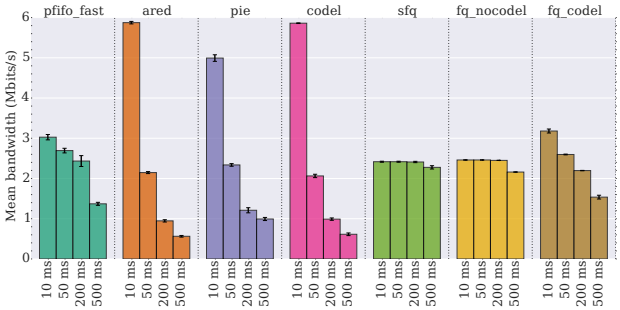

Figure 2: RTT fairness results. The graphs show the aggregate mean goodput of each of the four TCP flows for each queue management scheme.

5 Future planned development

Flent is developed as free software and used in the bufferbloat community to diagnose bufferbloat and develop and test solutions to it. An on-going effort is to make the tool easier to use, especially on first use, and so make it applicable for more people and in more situations. Additionally, expanding the scope of the tool both in terms of the tests included and the analysis capabilities is planned.

5.1 Ease of use and wider applicability

Flent neatly packages many things that would otherwise need to be done manually, and so lower the bar for running experiments significantly. However, the tool still needs to be run from the command line, and it needs to have the test name and target endpoints supplied.

A development goal is to make it easier to run tests out of the box, and support a straight-forward "download and run" use case. For this, we plan to establish a publicly available set of endpoint servers and add geo-based discovery of the nearest available server to Flent. In addition, we plan to develop a suite of common tests targeted at measuring internet access link behaviour.

Longer term, running tests directly from the GUI will make the tool accessible to users for whom using the command line is not an option.

5.2 Expanding test coverage and analysis capability

Developing test coverage as more interesting scenarios are discovered is an ongoing effort. Collecting these and distributing them as part of the Flent source code is an important part of future development.

As more test data is accumulated, storage and indexing can be a challenge. A future goal is to develop a solution for storing and indexing data, to ease comparison over both time and different test scenarios.

6 Related work

The difficulty of properly constructing and performing experiments, and of reporting accurately on the results of them is not limited to experiments conducted on real hardware. For instance, Kurkowski et al [4] found that many simulation studies in the MANET research community suffered from a series of common errors, many of which are related to those discussed here (e.g. lack of reproducibility and ambiguous initial configuration).

The TEACUP system [7] is a test automation framework created specifically to test TCP implementations. It differs from Flent in that its focus is on managing an entire testbed infrastructure, including managing rebooting of machines and configuring the whole testbed. As such, it is more full-featured in this respect, but also makes more assumptions on topology and configuration of the machines than Flent does, and has more dependencies. Additionally, while TEACUP offers graphing and analysis of test results, these are more limited, and there is no interactive GUI to explore the data.

D-ITG [1] is a traffic generation and test platform with an extensive list of supported traffic profiles. It is targeted towards running in a managed testbed and emphasis is put on remote management and having a separate control network through which logging data can be transferred. As such, unlike Flent, D-ITG does not include facilities to interoperate with other tools, and does not offer integrated analysis and plotting tools. Flent can use D-ITG as a benchmarking tool in test definitions.

7 Conclusions

We have presented Flent, a tool to facilitate experimental evaluations in networks, specifically designed to deal with commonly encountered issues with running experiments. These issues include coordinating different test tools, creating reproducible tests, managing test-bed configuration and storing and analysing measurement data. Flent tackles each of these issues, while seeking to be flexible and widely applicable. An explicit design goal is the ability to function without an extensive management infrastructure, to e.g. be runnable from a laptop over a network not under the experimenter's control.

We hope that Flent can be useful for others in the research community, and will continue to develop it as free software. Future development will have a special focus on ease of use and wide applicability, and on expanding the included suite of tests and the analytical capabilities of the tool.

8 References

[1] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. 'A tool for the generation of realistic network workload for emerging networking scenarios'. In: *Computer Networks* 56.15 (2012), pp. 3531–3547.

[2] *Flent source code repository*. URL: `http://flent.org`.

[3] Rick Jones. *Netperf*. Open source benchmarking software. 2015. URL: `http://www.netperf.org/`.

[4] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. 'MANET Simulation Studies: The Incredibles'. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 9.4 (Oct. 2005), pp. 50–61. ISSN: 1559-1662. DOI: `10.1145/1096166.1096174`.

[5] Jitendra Padhye et al. 'Modeling TCP throughput: A simple model and its empirical validation'. In: *ACM SIGCOMM Computer Communication Review*. Vol. 28. 4. ACM. 1998, pp. 303–314.

[6] Dave Taht. *RFC: Realtime Response Under Load (rrul) test specification*. Nov. 2012. URL: `https://github.com/dtaht/deBloat/blob/master/spec/rrule.doc?raw=true`.

[7] Sebastian Zander and Grenville Armitage. *TEACUP v0.8 – A System for Automated TCP Testbed Experiments*. Tech. rep. 150210A. Centre for Advanced Internet Architectures, Feb. 2015.