Flent: The FLExible Network Tester

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

Carlo Augusto Grazia
carloaugusto.grazia@unimore.it
University of Modena and Reggio Emilia

Per Hurtig
per.hurtig@kau.se
Karlstad University

Anna Brunstrom
anna.brunstrom@kau.se
Karlstad University

ABSTRACT

Running network performance experiments on real systems is essential for a complete understanding of protocols and systems connected to the internet. However, the process of running experiments can be tedious and error-prone. In particular, ensuring reproducibility across different systems is difficult, and comparing different test runs from an experiment can be non-trivial.

In this paper, we present a tool, called Flent, designed to make experimental evaluations of networks more reliable and easier to perform. Flent works by composing well-known benchmarking tools to, e.g., 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, collect relevant metadata and interactively plot and explore datasets.

We showcase Flent's capabilities by performing a set of experiments evaluating the new BBR congestion control algorithm, using Flent's capabilities to reproduce experiments both in a controlled testbed and across the public internet. Our evaluation reveals several interesting features of BBR's performance.

CCS CONCEPTS

• **Networks** → **Network experimentation**; **Network performance analysis**; **Network measurement**;

ACM Reference format:
Toke Høiland-Jørgensen, Carlo Augusto Grazia, Per Hurtig, and Anna Brunstrom. 2017. Flent: The FLExible Network Tester. In *Proceedings of 11th EAI International Conference on Performance Evaluation Methodologies and Tools, Venice, Italy, December 5–7, 2017 (VALUETOOLS 2017),* 6 pages.
https://doi.org/10.1145/3150928.3150957

1 INTRODUCTION

Running properly managed experimental evaluations of networks and network services can be tedious and error-prone, which is part of the reason why new network technologies are often primarily (or even exclusively) evaluated by simulation. In this work we present a tool that is designed to aid researchers in increasing reliability and making it easier to run tests on real network hardware.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
VALUETOOLS 2017, December 5–7, 2017, Venice, Italy
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6346-4/17/12.
https://doi.org/10.1145/3150928.3150957

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 be inaccurate; real-world systems simply behave *differently* than simulations. 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 years, in many aspects completely changing its behaviour. These changes mean that the assumptions underlying new research need to be verified against actual systems running on 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 ("the FLExible Network Tester"), works by composing well-known benchmarking tools to (for example) run tests consisting of several bulk data flows combined with simultaneous latency measurements or measure application traffic while loading the link with background flows. Tests are specified in source code, and several common tests are included with the tool. In addition, Flent contains features to automate test runs, collect metadata and auxiliary data sets, and to interactively plot data collected from experiments.

To showcase Flent's capabilities, we perform an experimental evaluation of the BBR congestion control algorithm [6] for TCP. BBR was released recently, and has only been subject of a few independent evaluations. As such, we consider an evaluation of BBR both timely, and appropriate for showing the capabilities of Flent. We evaluate the performance of BBR in a controlled testbed with different Active Queue Management (AQM) algorithms on the bottleneck link, as well as over a link on the public internet. The evaluation reveals several interesting features of BBR, including its ability to function without using drops as a sign of congestion.

The rest of the paper is structured as follows: Section 2 elaborates on the difficulties that we seek to alleviate, and Section 3 describes Flent and how it addresses them. Section 4 showcases Flent's capabilities by performing an experimental evaluation of the BBR congestion control algorithm. Section 5 summarises related work and, finally, Section 6 contains the concluding summary.

2 EXPERIMENTAL CHALLENGES

The key difficulties that the Flent testing tool is designed to address, are reproducibility of experiments, testbed configuration and automation, and storage and analysis of measurement data. The rest of this section outlines each of these in turn.

2.1 Reproducing experiments

Reproducing experiments is important for verifiability, for both the researcher, and for independent reproduction by others. However,

actually creating reproducible experiments is challenging [12].

Part of the reason for this is the complexity of coordinating different test tools. Since many network benchmarking tools are single-purpose, running different tools at the same time can be necessary when creating complex test scenarios. 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. The more ad-hoc the test configuration and setup is, the harder it is to ensure reproducibility of the tests.

2.2 Testbed configuration and test automation

Experiments often involve testbeds comprising several physical devices set up 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, configuration can fail, either from human error or from (unchecked) failures in the configuration process, so automation is important in both application and subsequent collection.

Finally, being able to automatically run a test series, including applying configuration between runs, significantly reduces the difficulty of experimental work, and makes it practical to test a larger set of variables in a single setting.

2.3 Storing and analysing measurement data

As the number of experiments grows, storing the measurement data and relating it to the right tested configuration 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 benchmarking tool outputs, and extracting the meaningful data points for analysis.

In addition, gathering relevant metadata can be extremely helpful in verifying the test setup and avoiding spurious errors from faulty configurations that make the test data invalid.

3 HOW FLENT HELPS

Flent is developed specifically to address the difficulties mentioned in the previous section, while also being extensible to address other future use cases. Flent is written in Python and can drive several other well-known network benchmarking tools, most notably Netperf. This section describes how Flent seeks to address each of the challenges presented in the previous section. A design diagram for Flent is available in the online documentation.[1]

3.1 Reproducing experiments

Flent works by running one or more tests, each defined by a configuration file that specifies which benchmarking tools to run, leveraging well-known tools such as Netperf and Iperf. 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). Each test defined in Flent has a name and a separate configuration file. This greatly aids reproducibility, as the named tests are available along with the source code, and can be referenced reliably across different systems running Flent. The tests are quite versatile, since larger test suites can be composed of the available named tests. This also allows Flent to work well with other tools that manage tests: Anywhere there's a Python environment and the required underlying benchmarking tools, Flent can run.

When running a test, the output of each testing tool is parsed, and the output 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 Configuration and automation

Flent manages configuration of the test environment by including support for running arbitrary scripts at the start and end of each test in a series. We have found that in the small to medium-sized testbed environments targeted by Flent, scripting is the most flexible choice for configuring nodes. To ensure correctness, these scripts must run before each test invocation, and any failures must be detected. To this end, 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 is an effective configuration management and verification facility, without the need to include platform-specific configuration code in Flent itself.

In addition, 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.

3.3 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 transferred to another system for further analysis. In addition, auxiliary data sets can be captured along with the main data series, including queueing statistics from the Linux qdisc layer, socket statistics reported by the operating system during the test, CPU usage, WiFi rate statistics and more. As with the metadata, these auxiliary data sets can be captured from both the local machine running the test, and from instrumented remote devices such as intermediate routers.

Flent also contains an extensive analysis facility, which reads already produced test files and produces plots of the data. A graphical user interface makes it easy to flip between plots of different test runs. Tests can define detailed 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.

[1] Flent (and its online documentation) is available at https://flent.org/.

The interactive plotting feature makes for a powerful analysis tool in the exploratory phases of experimental work. Additionally, the tool can also produce final high-quality graphs for publication: The example plots in the next section are all produced by Flent's built-in plotting facilities.

Many plot types are supported, allowing exploration to be performed directly from within Flent. 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 SHOWCASING FLENT: A LOOK AT THE BBR CONGESTION CONTROL

In this section, we showcase the capabilities of Flent to effectively run experiments, by evaluating the BBR congestion control algorithm [6] for TCP. This algorithm was proposed recently, and has garnered significant interest in the research community. BBR is designed to ensure high utilisation at all bandwidths without inducing unnecessary queueing latency (also known as bufferbloat). It does this by continuously estimating delivery rate and path RTT and adjusting its sending bandwidth accordingly. Every eight roundtrips, BBR will enter a one-RTT probing phase to probe for more bandwidth, followed by a drain phase where sending bandwidth is temporarily lowered to allow buffers to drain. BBR also employs packet pacing, where packets are sent at a constant rate over the whole RTT, instead of being bursted out in window-sized bursts as traditional TCP does.

We believe it is relevant to evaluate how BBR performs in the presence of another technology that has been shown to be effective against bufferbloat: Active Queue Management (AQM) algorithms installed at the bottleneck link. Performing such an evaluation is exactly what Flent excels at, since it involves applying well-known testing methodology (bufferbloat tests) to a new scenario (BBR).

To also showcase how we can perform the same tests in different environments and compare the results, we repeat out tests over the public internet. In both evaluations, we compare the performance of BBR with the well-known CUBIC congestion control. We focus on the application level goodput and the end-to-end latency under load as our metrics of interest. In the testbed experiments, we apply the two state of the art AQM algorithms CoDel [13] and PIE [14], which both work by dropping packets before the queue fills up, to signal TCP to slow down. We also apply the FQ-CoDel [11] hybrid AQM and packet scheduling algorithm, which combines CoDel with a flow scheduler to also provide flow isolation, fairness and low latency for sparse flows.

4.1 Experimental setup

For our testbed evaluation, we re-use the testbed from a previous study of AQM algorithms [10]. The testbed consists of five nodes, connected as depicted in Figure 1. All nodes are regular x86 PCs with Intel Core 2 CPUs and Intel 82571EB network controllers, running Debian Jessie with a backported Linux kernel version 4.11. We configure the bottleneck to be 10 Mbps and the baseline RTT between client and server to be 50 ms. This emulates a connection to a server on the internet over a residential internet connection.

Flent runs at the node labelled 'Client' and runs tests against

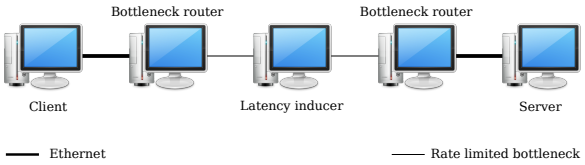

Figure 1: Testbed setup.

the 'Server' node. We run three different tests included in the Flent source distribution: A TCP download test and a TCP upload test (each running one TCP flow in the respective direction combined with a periodic ICMP Ping to measure latency), and the Realtime Response Under Load (RRUL) test [16], which is designed by the bufferbloat community as a stress test to show the presence of bufferbloat. The latter test runs a total of eight simultaneous TCP flows (four in each direction) while simultaneously measuring latency using both UDP and ICMP packets.

For the test running over the public internet, we run the same suite of tests, but between a test machine located at the University of Modena and Reggio Emilia in Italy and another machine located at Karlstad University in Sweden. The path characteristics between the two endpoints are unknown before the experiment is conducted.

We use the batch facility of Flent to repeat all tests 15 times (for the internet tests) and 30 times (for the testbed tests). We use the interactive plotting interface to explore individual test runs and point out interesting features, and the aggregate plotting facilities to combine all the test runs into meaningful metrics for the tested algorithms. Detailed instructions for replicating the experiments are available online.[2]

4.2 Testbed results

In the testbed results, we first examine the behaviour of a single flow with a FIFO queue, using the TCP upload test.[3] We use Flent's default timeseries graph of throughput and ping latency over the duration of the test to get an idea of how the two different congestion control algorithms work. These graphs are shown in Figure 2. We clearly see how the latency increases to around a full second when using CUBIC; and when the queue overflows we see spikes in both latency and goodput as delivery of date to the application first slows down, then catches up after packets have been retransmitted.

In contrast to this, BBR manages a level throughput and low latency for most of the time, punctuated by spikes in latency every 10 seconds when the algorithm probes for more bandwidth. This shows BBR working as intended and preventing most of the bufferbloat on the bottleneck link from affecting the application. This comes at only a small cost in bandwidth: BBR achieves an average of 9.35 Mbps, while CUBIC achieves 9.55 Mbps.

Turning to the AQM algorithms, one feature of BBR's behaviour is that since it does not interpret loss as a sign of congestion, it is not being controlled only by the AQM. This can be seen in the data in two ways: In the drop behaviour of the AQMs, and in the relative performance of the different AQM algorithms. The number of packets dropped by the CoDel AQM is seen in Figure 3. This is

[2] See https://www.cs.kau.se/tohojo/flent/.

[3] Since the bottleneck is symmetrical, the TCP download test shows identical behaviour in this scenario.

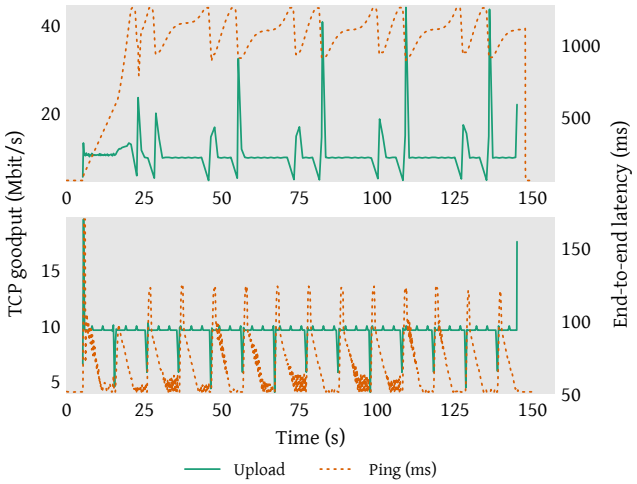

Figure 2: Single flow and ping, FIFO queue. CUBIC (top) and BBR (bottom).

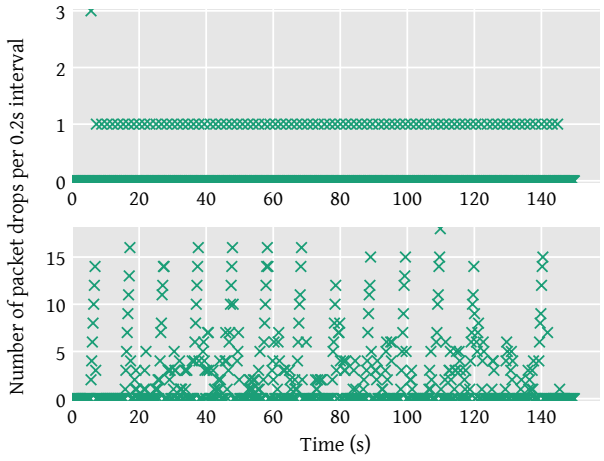

Figure 3: Number of packets dropped by CoDel per measurement interval (0.2 seconds). CUBIC (top) and BBR (bottom).

captured from the bottleneck router by the auxiliary data collection facility in Flent. From the figure, it is very clear that CoDel works as intended for the CUBIC flow: After a small burst of drops during TCP slow start (seen at top-left of the graph), CoDel tunes itself to a drop rate that keeps CUBIC oscillating around the bottleneck bandwidth: around one drop per two seconds in this case.

In contrast to this, for the BBR flow, CoDel keeps increasing the drop rate in an attempt to get the queue under control, whenever BBR increases the rate in the probe phase. This is seen in the crosses on the lower part of Figure 3 that look like they are aligned vertically above one another. Actually, these are not completely vertical, but rather show a sharply increasing drop rate over a very short time (this is barely visible in the figure, but can be clearly seen when zooming in on the plot). Between the probe episodes, periods of quiescence appear, where the AQM does not drop any packets.

When looking at the latency distribution, it is clear that BBR is less influenced by the AQM than CUBIC is. To show this, we turn to one of the aggregate plots available in Flent. Figure 4 shows a CDF of the latency measurements across all test repetitions for the CoDel and PIE AQMs for both CUBIC and BBR. Consistent with earlier evaluations of the AQMs [10], CoDel achieves lower delay than PIE, since it tends to drop more aggressively. However, because this difference is more pronounced when using CUBIC (the leftmost and rightmost lines), it means that the relative performance of BBR and CUBIC is reversed depending on the AQM: With CoDel, the queueing delay is lower when using CUBIC that when using BBR, while the reverse is true for PIE. Another interesting feature is the shape of the tail of latency measurements: For CUBIC, the AQMs have a few but quite high spikes of latency at the 99th percentile. For BBR, these do not appear, but a thicker tail, starting at the 80th percentile, of latency measurements up to 125 ms is clearly visible.

The FQ-CoDel hybrid AQM/fairness queueing algorithm shows almost no induced latency for either congestion control, consistent with earlier results. This is not shown in Figure 4 as it would obscure the lines of the other algorithms; however, the FQ-CoDel results are included for the RRUL test considered next.

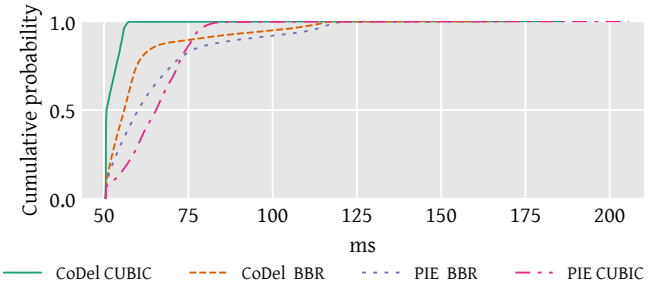

Figure 4: Latency distributions with a single flow for the CoDel and PIE AQM algorithms.

To evaluate a scenario with higher load on the link, the RRUL test is an excellent tool. An overview of the aggregate behaviour for the different combinations of AQMs and congestion control algorithms using the RRUL test is shown in Figure 5.[4] This plot is a so-called "ellipsis plot", which is another of the plot types included in Flent.[5] This plot is excellent for summarising a lot of data in a compact representation. The latency axis is flipped to make "better" values be up and to the right, to fit the intuition of laymen. The dots for each data series is the median values, while the ellipses are 1-σ ellipses of the values on the two axes (so larger ellipses indicate higher variance and the angle of the ellipsis shows the direction of covariance between the variables).

From the figure, the same trends are visible as before: The behaviour of BBR does not change significantly depending on the AQM. As the RRUL test has a larger number of active flows, the

[4]The CUBIC FIFO case is omitted as that shows latency so high that it would squeeze the rest of the figure to the point of making it illegible.

[5]The use of this type of graph has not been invented for Flent; it was pioneered for visualising bandwidth/latency tradeoffs by Winstein in [17].

latency induced by BBR is higher; and so CUBIC induces less latency than BBR with both PIE and CoDel, but BBR achieves slightly higher throughput. Finally, the superior performance of the FQ-CoDel AQM is clearly visible.

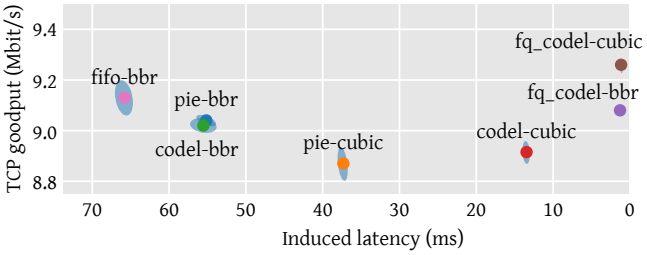

Figure 5: Ellipsis plot of throughput and latency for the RRUL test.

In summary, our testbed evaluation clearly shows several interesting features of BBR: It achieves a significantly lower latency than CUBIC on a FIFO queue, although the intermittent probing behaviour gives significant latency spikes. In addition, the induced latency is higher when many flows are active, and AQM algorithms do not impact the performance as much as they do for CUBIC. However, the AQMs will sharply increase their drop rates in response to BBRs probing behaviour. While this does not impact the behaviour of BBR itself in this test, it has the potential to impact other flows sharing the bottleneck negatively.

4.3 Public internet results

To test how BBR performs outside a controlled environment, we set out to repeat out experiments "in the wild". Because the tests we performed are built-in to Flent, repeating the tests was simply a matter of installing the tool, transferring the configuration file for batch run to another machine, and making a few adjustments (such as the target host for the tests). We performed these repeated tests over the public internet between our two universities. The path characteristics were not known in advance, but the test data allows us to infer several things of interest, as we will see below.

The initial one-flow test is shown in Figure 6. BBR shows consistent performance around 110Mbps of throughput, while CUBIC has an initial spike at around the same speed, but quickly becomes erratic and achieves low overall throughput (20 Mbps mean throughput for CUBIC over the whole test, 110 Mbps for BBR).

Since the CUBIC throughput decreases over a series of events, we hypothesise that the bottleneck router is shallowly buffered, and other users sharing the bottleneck causes our flow to experience a lot of packet drops. To confirm that the drops in throughput correspond to window reduction events, we examine the TCP congestion window data (as reported by the operating system and captured as an auxiliary data series by Flent).

This is shown in Figure 7 for both the flow depicted above, and for another flow from a separate test we conducted at night (where the network load is likely to be lower, which should lead to fewer overflow-induced drops). From this figure, we clearly see the characteristic CUBIC window increase and decrease. We also see significantly better performance for the experiment conducted at night

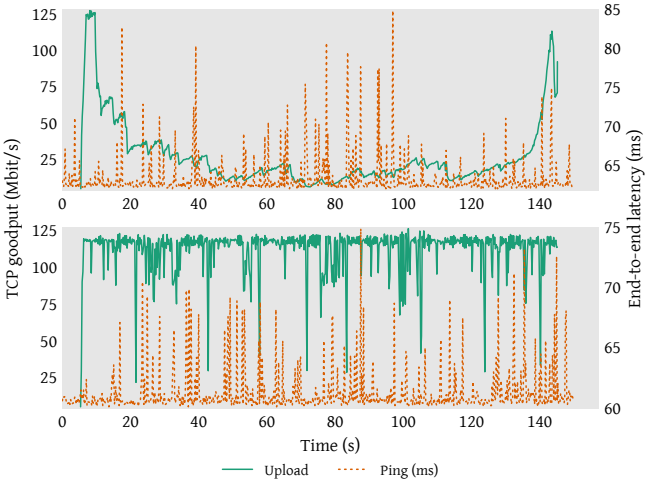

Figure 6: Initial test run over the internet. CUBIC (top) and BBR (bottom).

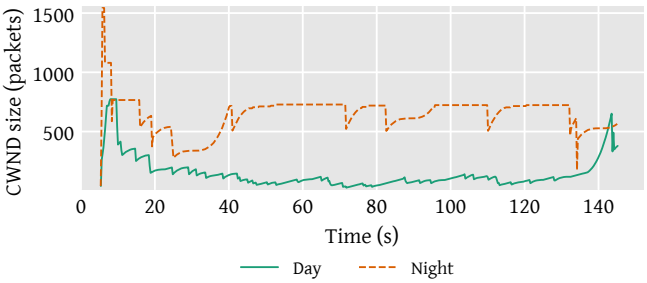

Figure 7: TCP CUBIC congestion window size, daytime and nighttime tests.

(dashed line); the average throughput for this run is 115 Mbps, on par with what BBR achieves. However, we still see several consecutive loss events, which strengthens our shallow buffer hypothesis.

Next, we turn to the RRUL test. Figure 8 shows the aggregate results of the whole test series for both BBR and CUBIC, and showcases the box plot type also featured in Flent. We also added a third contender: CUBIC with packet pacing (labelled "P. CUBIC" in the graph), to see how much of BBR's performance comes from pacing.

From the figure, we see that having more than one active flow significantly increases the total bandwidth available. We also see that pacing does indeed help CUBIC achieve better throughput in one direction, while it is less conclusive in the other. In either case, BBR consistently performs significantly better, keeping aggregate throughput steady for all tests, at a slight cost in latency.

In summary, we have repeated our tests from the testbed over the public internet, and confirmed several features of BBR. In particular, not treating every drop as a sign of congestion allows BBR to achieve better performance in the face of what appears to be a congested bottleneck link with shallow buffers, although possibly to the detriment of competing flows. We have also shown how to use the data gathered by Flent to infer properties of the bottleneck link, such as the congestion levels at different times of day.

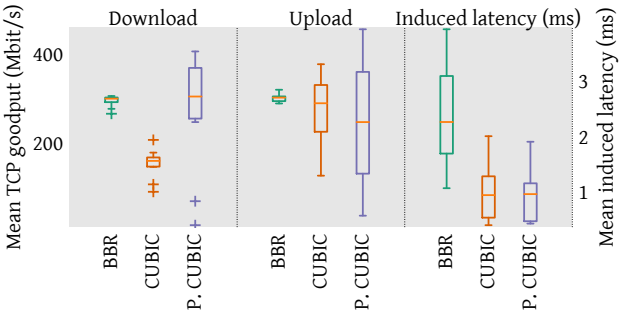

Figure 8: Aggregate values for all RRUL tests over the internet. "P. CUBIC" is CUBIC with packet pacing enabled.

5 RELATED WORK

The difficulties of properly constructing and performing experiments, and of reporting accurately on the results of them are not limited to experiments conducted on real hardware. For instance, Kurkowski et al [12] 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).

Turning to test tools, the TEACUP system [18] 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. This means that it has more features for configuration, but also makes more assumptions on topology and the nature of the testbed than Flent does. Additionally, while TEACUP offers graphing and analysis of test results, these are more limited, and there is no interactive GUI to explore the data. Netesto [4] is closer in function to Flent, but also focuses on orchestration of several nodes when running a test, and supports fewer types of traffic.

D-ITG [3] is a traffic generation and test platform with an extensive list of supported traffic profiles. It is geared towards running in a managed testbed and emphasis is on remote management with a separate control network to transfer logging data. 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.

TEMPEST [7] is a simulation framework that makes it possible to run packet scheduling code from operating system kernels in a simulated environment and evaluate various performance metrics with an accessible graphical user interface. As such, it shares the goal of Flent of making it easier to evaluate real networking code, but takes the approach of porting the code to a simulation environment instead of running experiments in a live environment.

Dummynet [5] and netem [9] are emulation modules that can be used in a real network topology to emulate network features not available in the physical hardware. Flent can be used to run experiments in topologies that include emulated dummynet or netem links. Indeed, we use netem for the purpose of adding latency to the bottleneck link in our own testbed.

Finally, several platforms are available for researchers to run their experiments in extensive testbeds, either isolated or across the public internet [1, 2, 8, 15]. Flent can be used to drive experiments on these platforms, further increasing the ease of running experiments.

6 CONCLUSIONS AND FUTURE WORK

We have presented Flent, a tool to facilitate experimental evaluations of networks, specifically designed to deal with commonly encountered issues with running experiments. These issues include creating reproducible tests, storing and analysing data, and test automation and configuration management. Flent tackles each of these issues, and is flexible enough to be widely applicable.

We have showcased Flent through an analysis of the BBR congestion control algorithm, both in a controlled testbed with various active queue management algorithms installed, and in an uncontrolled setting over the public internet. This analysis has shown how BBR functions without reacting to drops as a sign of congestion, and revealed several interesting consequences of this behaviour.

Development of Flent is ongoing, and future plans include improving the ability to run tests from the graphical user interface, as well as adding support for more data sources and low-level test tools. This development is guided by feedback from the online community as well as our own needs when running experiments.

REFERENCES

[1] V. Bajpai, A. W. Berger, P. Eardley, J. Ott, and J. Schönwälder. 2016. Global Measurements: Practice and Experience (Dagstuhl Seminar 16012). *Dagstuhl Reports* 6, 1 (2016), 15–33. https://doi.org/10.4230/DagRep.6.1.15
[2] V. Bajpai and J. Schönwälder. 2015. A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts. *IEEE Comm. Surveys Tutorials* 17, 3 (2015), 1313–1341. https://doi.org/10.1109/COMST.2015.2418435
[3] A. Botta, A. Dainotti, and A. Pescapè. 2012. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks* 56, 15 (2012), 3531–3547.
[4] L. Brakmo. 2017. Netesto, a Network Testing Toolkit. In *NetDev 2.1 - The Technical Conference on Linux networking*. https://netdevconf.org/2.1/session.html?brakmo
[5] M. Carbone and L. Rizzo. 2010. Dummynet Revisited. *ACM SIGCOMM Computer Communication Review* 40, 2 (March 2010), 12–20.
[6] N. Cardwell, Y. Cheng, C S. Gunn, S. H. Yeganeh, et al. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.
[7] M. Casoni, C. A. Grazia, and P. Valente. 2014. TEMPEST: a new Test EnvironMent for Performance Evaluation of the Scheduling of packeTs. *Simulation Modelling Practice and Theory* 49 (2014). https://doi.org/10.1016/j.simpat.2014.10.005
[8] E. Eide, L. Stoller, and J. Lepreau. 2007. An Experimentation Workbench for Replayable Networking Research. In *Proceedings of the Fourth USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.
[9] S. Hemminger. 2017. netem. (2017). https://goo.gl/25x8zV
[10] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom. 2015. The Good, the Bad and the WiFi: Modern AQMs in a residential setting. *Computer Networks* 89 (Oct. 2015), 90–106. https://doi.org/10.1016/j.comnet.2015.07.014
[11] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. 2016. FlowQueue-Codel. Internet Draft. (March 2016). https://goo.gl/qXsvFE
[12] S. Kurkowski, T. Camp, and M. Colagrosso. 2005. MANET Simulation Studies: The Incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.* 9, 4 (Oct. 2005), 50–61.
[13] K. Nichols and V. Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (July 2012), 42–50. https://doi.org/10.1145/2209249.2209264
[14] R. Pan, P. Natarajan, C. Piglione, M.S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*. 148–155. https://doi.org/10.1109/HPSR.2013.6602305
[15] L. Peterson, A. Bavier, M. E Fiuczynski, and S. Muir. 2006. Experiences building planetlab. In *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 351–366.
[16] D. Taht. 2012. RFC: Realtime Response Under Load (rrul) test specification. (Nov 2012). https://github.com/dtaht/deBloat/blob/master/spec/rrule.doc?raw=true
[17] K. Winstein. 2014. *Transport Architectures for an Evolving Internet*. Ph.D. Dissertation. Massachusetts Institute of Technology. https://goo.gl/tkLf49
[18] S. Zander and G. Armitage. 2015. *TEACUP v0.8 – A System for Automated TCP Testbed Experiments*. Technical Report 150210A. CAIA, Swinburne Univ. of Tech.