Computer-Aided Reproducibility

Marcel Marek | Peyman Teymoori | Michael Welzl | Stein Gjessing
Department of Informatics
University of Oslo
Email: {marcelma|peymant|michawe|steing}@ifi.uio.no

***Abstract*—Computer networks research has been notoriously bad at reproducibility – a key aspect of making research results credible and convincing. This has been attributed to a lack of incentive for researchers to share the data underlying scientific results. We conjecture that this can be helped by reducing the amount of work that is required to make results reproducible. This paper introduces CAR – a system for "Computer-Aided Reproducibility". Similar to other forms of "Computer-Aided-*", our CAR tool facilitates the process of sharing the necessary data by partially automating it.**

I. Introduction

Writing a research paper is an arduous process. Once it is done, there is often not much incentive to make the artifacts produced in the process (code, input data, output data) available to others. While a paper is under submission, it may not always be adequate to openly share information related to it (e.g., when a venue has a double-blind review policy, such sharing may be deemed inappropriate and may be perceived as an effort to cheat). Once a paper is accepted, the major goal of its authors is reached and there is usually no immediate need for them to put extra effort into sharing the necessary data such that others can reproduce the work. Yet, making results openly available to ensure reproducibility is more than just good scientific practice—it is indeed a cornerstone of science. Reproducibility ensures that research results are credible and convincing.

The unwillingness to share data for the sake of reproducibility affects multiple fields of science, leading to what has been called a "reproducibility crisis" [1]. In the field of computer networks, it has recently been addressed by some publication venues that try to create an incentive—e.g. the ACM Internet Measurement Conference (IMC) has established an award for papers that contribute a novel dataset to the community. The problem has moved into the spotlight: a blog related to a Stanford course in which students try to reproduce results from research papers[1] has become quite well known, and the ACM SIGCOMM conference hosts a workshop on reproducibility in 2017.[2]

It seems that there are two major angles to attack the reproducibility problem. The first one is to create the right incentive structures—e.g. via awards, as in the case of the IMC conference, or via regulations put forward by publication venues, funding bodies or the research institutions and Universities themselves. The second one has, to the best of our knowledge, not been addressed much so far: *facilitating* the effort of putting the necessary data online and openly sharing it.

Filling this hole, in this paper, we introduce *Computer-Aided Reproducibility (CAR)*: a method to partially automate the process and involve the human as a guide in the process of auto-producing a website that contains the necessary data. We apply CAR to TEACUP, a system for network experimentation described in [2], [3], [4]. In section, II, we describe TEACUP and how we build the accompanied testbed. We also discuss how to design, run, and analyze new experiments. In Section III, we introduce CAR and describe how it facilitates publishing results. In section IV, we provide a step-by-step example of how a research result can be obtained and published; we perform experiments on the new "BBR" TCP congestion control mechanism [5] and publish its results. First, we extend TEACUP to support BBR, and then, compare it with some other congestion controllers using different Active Queue Management (AQM) methods. Then, we demonstrate how CAR auto-produces a website summarizing all the experiment results/artifacts. We conclude after an overview of related work in section V.

II. Testbed Setup

In this section, we describe how we build our testbed using TEACUP, and how to perform experiments on it. TEACUP - TCP Experiment Automation Controlled Using Python - is a tool to automate testing of various network parameters, mainly focused on TCP [2], [4]. An underlying testbed needs to be deployed to fully utilize TEACUP capabilities. The cookbook on how to build such an environment is described in [2]. We used it as a guide to build our testbed. The testbed slightly differs from the one described in [2]. The network topology of the testbed is depicted in Fig. 1.

Our aim was to use the latest version of necessary tools and libraries. One particular reason was that we wanted to test the new TCP congestion control BBR which is currently available only under Linux since kernel version 4.9. We had to extend TEACUP to add the support for BBR. TEACUP in its source code explicitly states which TCP congestion control variants can be used under each operating system. It also states the command to be used to load the appropriate kernel module. This was actually unnecessary for our testbed since

[1] https://reproducingnetworkresearch.wordpress.com

[2] http://conferences.sigcomm.org/sigcomm/2017/workshop-reproducibility.html

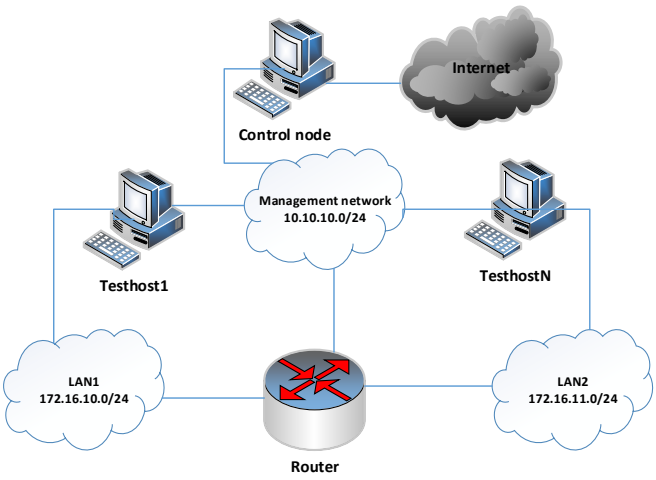

Fig. 1. TEACUP testbed network topology

we compiled our kernel with all available TCP congestion control variants.

A. Topology

The testbed environment consists of 7 PCs. Five machines serve as testbed hosts, one as a router and one as a control host. The control host is used to deploy OS images to testbed hosts and run the actual TEACUP code. It also serves as a network gateway to the Internet performing NAT with port overload (for software updates). The control node provides DHCP and TFTP server to support iPXE (network boot) and NTP as all nodes need to be synchronized.

Since the control host is reachable from the Internet the connection is possible via SSH with `publickey` – only authentication method.

B. Networks

Three networks are configured within the testbed. Management network is used to deploy OS images. TEACUP uses the management network to boot any chosen OS, configure router queues, initiate traffic generators and collect data (tcpdump, routing tables, interface names, etc.) from all active nodes in a particular experiment.

The other two networks are LAN1 and LAN2 subnets, which are connected through the router. Moving a testbed host from LAN1 to LAN2 or vice versa is just a matter of assigning its respective port in the switch to an appropriate VLAN. LAN1 is represented as VLAN 10 and LAN 2 as VLAN 20.

As the switch, Cisco 3650 series is used to interconnect all nodes. We had to disable STP; otherwise, the iPXE process would time out before the given switch port would transition to active state.

C. Testbed Hosts

We chose FreeBSD 11 and Ubuntu Linux 16.04.02 LTS dual boot for the testbed hosts. The version of the Linux kernel is 4.10 from the web10g project[3]. It is a patched vanilla kernel with enabled support for Extended TCP statistics (RFC4898 [6]) MIB. The web10g project also includes kernel module[4] and userland library[5]. The reason why we chose 4.10 instead of 4.9 is that the web10g patch for 4.9 is not available.

Unlike the control node, testbed hosts have the SSH `PermitRootLogin` option enabled. The root account is used by TEACUP to control the experiment.

Each testbed host is equipped with inbuilt 100BASE-T NIC for the control network and one PCI-e 1Gb card for the data network. Each node can, therefore, be connected to only one of the subnetworks.

1) TCP loggers: We had to create a small patch for the 4.10 web10g kernel since it would not compile due to changes in `ktime.h`. The fix is provided in the form of a commit[6] in our forked repository and also submitted as a pull request to the web10g official repository.

The second TCP logger is called `ttprobe`, and since TEACUP version 1.0, it has been provided in the tools of the official repository. It needs to be compiled and distributed to the testbed hosts.

2) Traffic generators: The patched version of `iperf` from the authors of TEACUP is used. This patched version enables the configuration of TCP's sender and receiver window buffer size. It also includes a modified version of `httperf`. We omit the description of FreeBSD as it is not used in our experiment of BBR.

D. Router

The router also runs FreeBSD 11 and Ubuntu Linux 16.04.02 LTS, but without the TCP loggers, traffic generators, etc. It performs routing between LAN1 and LAN2. The router is used to create artificial bottleneck throughput, which adds delay and loss to the crossing traffic.

E. Designing Experiments

When configuring the first experiment, a copy of `fab.py` and `run.py` from the TEACUP installation directory should be made. Then, custom configuration file(s) can be created. It is recommended to create separate configuration files for certain sections of the configuration so it can be easily used in future experiments. For example, you can use the following structure:

- Basic configuration user access credentials, TEACUP install dir location, maximum time allowed time difference between two testbed hosts (`config-fabric.py`).
- Testbed configuration includes tesbed hosts IP addresses, names, and the OS to be booted. A kernel name has to be provided in case the node uses Linux as OS. (`config-testbednodes.py`).
- Router configuration queue type (packet, byte-based) delay, etc (`cofig-router.py`).

[3]https://github.com/rapier1/web10g

[4]https://github.com/rapier1/web10g-dlkm

[5]https://github.com/rapier1/web10g-userland

[6]https://github.com/screw/web10g-dlkm/commit/87c5357218f5fb3b889c35738a6ff1797ce15af6

- Traffic generator configuration greedy (`iperf`), bursty (web traffic, DASH) (`config-router.py`).
- Parameter combination configures over which parameters the experiment iterates. The number of configurations is the result of multiplication of all choices for each parameter and the number of runs. All the previous files are included in the main configuration file (`config.py`) by using `execfile("<config-file.py>")`.

After the experiment is designed, we can either run it by executing `./run.sh` from the experiment directory or by executing `fab run_experiment_[single, multiple]`. When the experiment is finished, plots can be generated by running one of the `analyse_` commands to get a plot for a specific metric. Alternatively, `analyse_all` can be run to get plots for all metrics. The full set of parameters for each command can be found in [3].

1) Notes on Using TEACUP: During the process of building and using the testbed we encountered several problems. One particular was that we did not notice that TEACUP disables the `net.ipv4.tcp_moderate_rcvbuf`. Only later we realized that we had to specify the receiver buffer in the traffic generator section. It might have been the price of a novice user, but disabling the default functionality as this one should be strongly announced in the documentation.

III. CAR

We developed and integrated CAR into TEACUP in the form of added fabric `@task - publish` - by extending the `analyse.py`. The complete source code of TEACUP extended with the CAR functionality is found at our public repository https://github.com/screw/teacup.

Our tool assumes the user has already produced the set of graphs related to his experiment by running one of the `analyse_*` commands which produce a set of PDFs.

By running `fab publish`, our extension to the TEACUP library

- collects all PDF names according to the TEACUP default naming scheme,
- extracts the command that was used to produce this plot based on the file name,
- converts all PDFs to PNGs for the website, and
- generates the HTML page based on the provided template.

If more sophisticated ways were used to generate certain plots, the user can provide the full description in a file named `<file>.txt` where `<file>` represents the full name of the PDF. The `fab publish` task has a parameter (`overwrite`) that disables the *guess-the-command* feature and uses the matching file containing the command(s) instead. The full set of parameters for the `publish` task is presented in Table I. Currently, only one template is provided, but users can provide their own as long as they stick to the predefined names. The full set of template variables are found in Table II.

TABLE I
PUBLISH TASK PARAMETERS

Variable name	Description	Default value
`find_dir`	Directory to look for PDFs.	`./`
`out_dir`	Output directory where the final webpage with all necessary files will be placed.	`publish/`
`overwrite`	This flag specifies whether to replace an existing file with the description of how to reproduce this PDF if such file exists.	`True`
`title`	Title of the paper	`NULL`
`paper`	Path to the article PDF.	`paper.pdf`
`desc_file`	File containing the abstract.	`desc.txt`
`density`	Density for the convert command to get PNG from the PDF.	`450`
`source_link`	Link to source code (eg. Github repository).	`NULL`
`config_name`	File containing the experiment configuration.	`NULL`
`source_link`	Link to source code (eg. Github repository).	`NULL`
`author`	Authors' name and email.	`NULL`
`conclusion`	Text that concludes the paper or webpage.	`NULL`

TABLE II
WEBSITE TEMPLATE VARIABLES

Variable name	Description
`author`	Authors' name and email.
`title`	Title of the paper.
`abstract`	Abstract or opening text for the webpage.
`submenu`	Submenu to jump to reproducibility sub-sections.
`repro_content`	Block with figures and their description.
`source_link`	Link to source code (eg. Github repository).
`conclusion`	Text that concludes the paper or webpage.

IV. EXAMPLE

Using our extended TEACUP which is empowered with CAR, we conducted an experiment, and then, we published its results. In this experiment, the goal is to compare various TCP congestion control variants (NewReno, Vegas, Cubic) with BBR. We also evaluated the BBR performance with different Active Queue Management (AQM) methods, queue sizes, and delay. Queue size is varied to test the BBR behavior with queue sizes smaller or bigger than Bandwidth-Delay Product (BDP).

In the experiment, the bottleneck at the router is configured with 10 Mbps rate limit and 20 ms delay in each direction. The AQM is Fair Queue with a queue size of 50 packets.

Two connections are initiated. The first one, from `testhost2` to `testhost1`, starts at time 0 s, and uses BBR. The second one which uses Cubic starts at time 10 s, from `testhost3` to `testhost2` which shares the bottleneck. Both connections use `iperf` as the traffic generator. The data packet size is 1448 B.

Fig. 2 illustrates the throughput of the two TCP connections (4 unidirectional flows in total). The two flows are represented by blue circles and blue triangles in all throughput and congestion window figures. The acknowledgments going in opposite direction are shown using other colors. We left them in the plots to demonstrate the default behavior, but they can

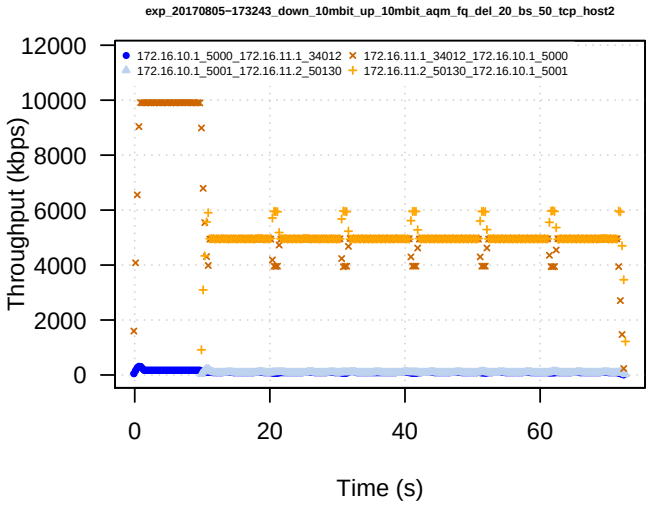

Fig. 2. Throughput of BBR and Cubic flows.

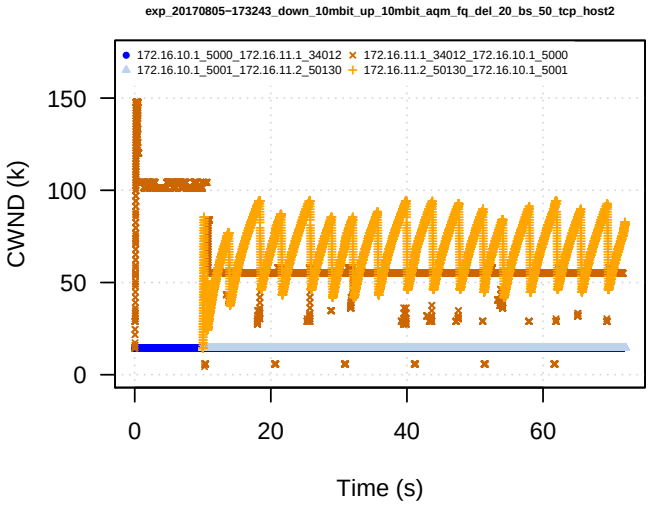

Fig. 3. CWND of BBR and Cubic flows.

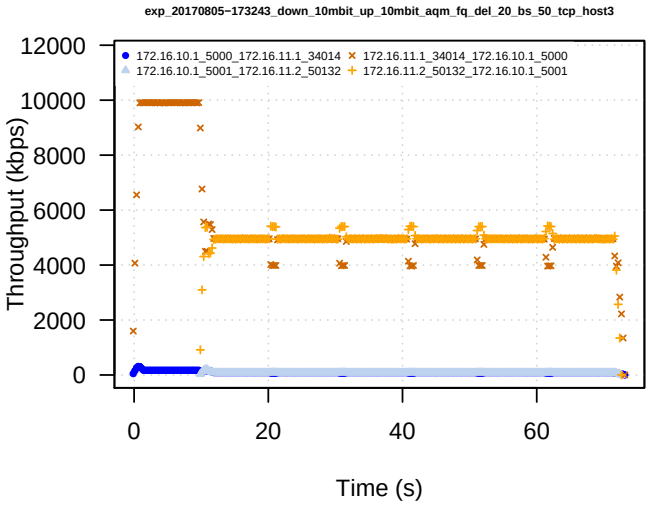

Fig. 4. Throughput of BBR and New Reno flows.

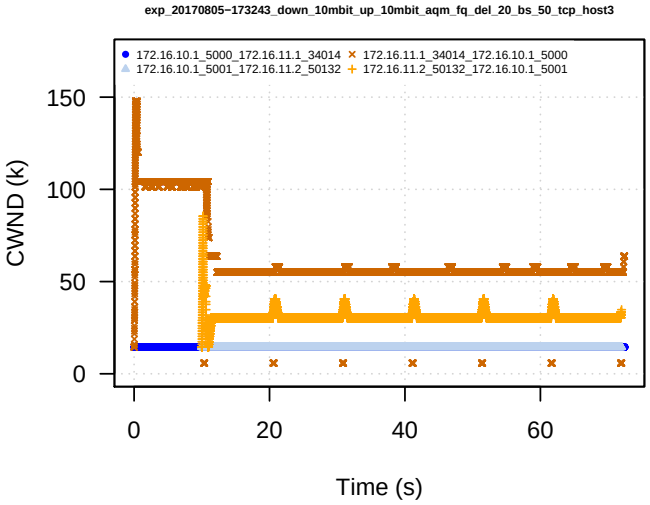

Fig. 5. CWND of BBR and New Reno flows.

be removed by specifying a `filter` to one of the `analyse` tasks.

First, the BBR flow is started and then after 10 s, the second one using Cubic, the current Linux default TCP congestion control, is initiated. BBR quickly reaches the bottleneck throughput of 10 Mbps. When both connections are started, they fairly share the available bandwidth. Every 10 s, there is a slight oscillation caused by the BBR's draining queue mechanism. In general, both algorithms achieve almost the same throughput.

A different situation is when we take a look at the congestion window (CWND) in Fig. 3. After overshooting at the start, BBR gets stabilized at 100 KB. After Cubic starts, BBR drops to 50 KB, and again it is fairly stable for the whole period of the connection. We can see subtle oscillation every 10 s when BBR is trying to adjust to the current situation by *draining the queue* mechanism. Cubic, on the other hand, oscillates with the typical *sawtooth* pattern.

The throughput of BBR versus New Reno in Fig. 4 is very similar to that of BBR versus Cubic in Fig. 2. Both algorithms share the available throughput fairly. Again, small oscillations caused by BBR appear every 10 s.

If we take a look at the corresponding CWND situation in Fig. 5, we can see that New Reno achieved almost the same throughput.

A completely different situation is depicted in Fig. 6, where the start of the Vegas flow is correlated with BBR's draining queue mechanism; it causes the BBR flow to heavily reduce the rate. During the period before reaching the subsequent iteration of draining queue, the BBR flow gets almost to a halt. Only after the next occurrence of draining queue, both flows get the same share of the bottleneck bandwidth. We cannot claim if this is caused purely by the Vegas algorithm or by timing. We would need to perform more runs, prove one or the other.

As for the congestion window with BBR and Vegas flows, both flows are fairly stable from 30 s.

In the second set of figures, we present the throughput and

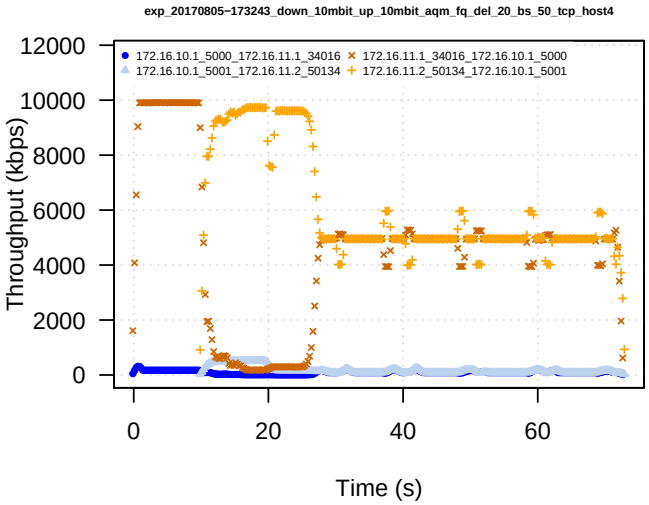

Fig. 6. Throughput of BBR and Vegas flows.

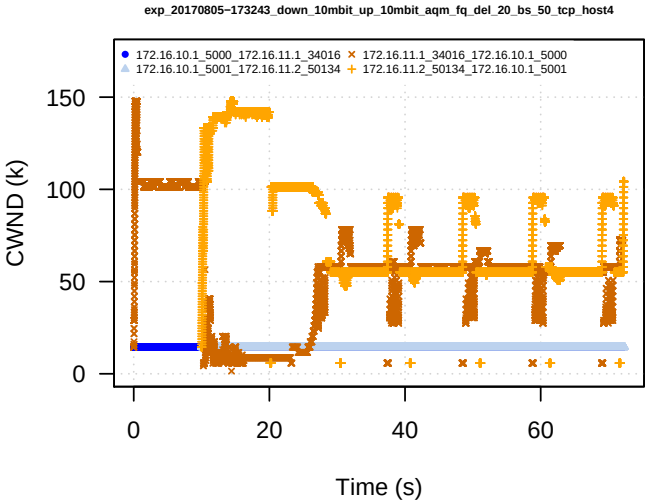

Fig. 7. CWND of BBR and Vegas flows.

congestion window plots with the same delay (20 ms) and queue size (50 packets) but with different AQM - `pfifo`. Only with Cubic (see Fig. 8 we can see a fair share of the bottleneck throughput. The other ones get either heavily outperform by BBR (New Reno), or Vegas outperforms BBR.

In Fig. 9, we can actually see that Cubic achieves a fair share throughput with much smaller congestion window compared to BBR. There is again the typical sawtooth-like pattern for Cubic flow. BBR congestion window is much more unstable than in the previous cases with Fair Queue.

In Fig. 10, BBR does not let the New Reno flow to get above 2 Mbps for the whole period of the connection.

And finally, Fig.12 shows BBR versus Vegas. Vegas outperforms BBR throughout during the whole connection but gets closer each time the draining queue occurs. It could be worth investigating if both flows get the same share if we run for a long enough period.

Fig.13 only illustrates the congestion window situation from the respective throughput figure. Once the Vegas flow is started

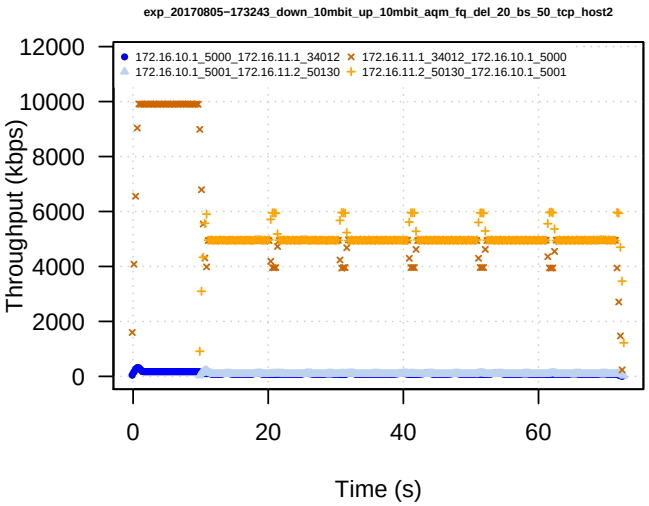

Fig. 8. Throughput of BBR and Cubic flows with PFIFO as AQM.

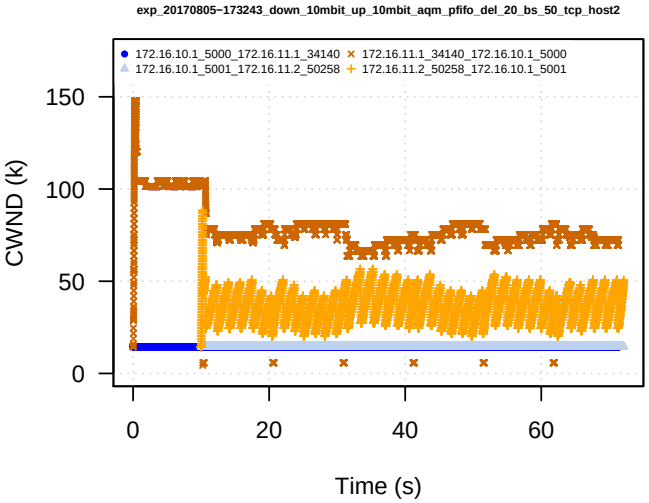

Fig. 9. CWND of BBR and Cubic flows with PFIFO as AQM.

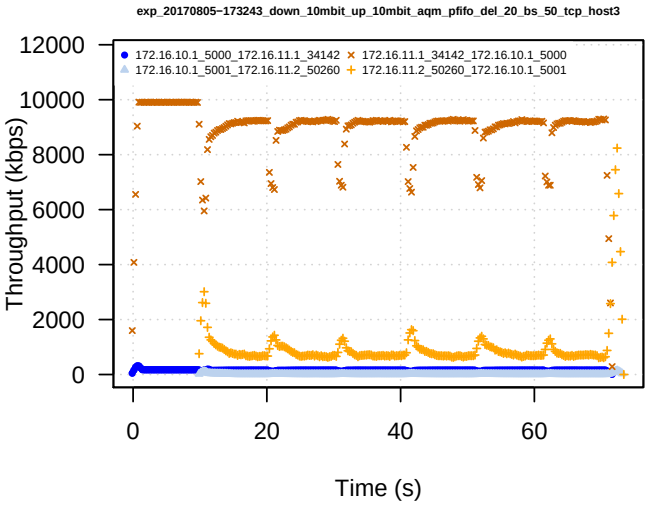

Fig. 10. Throughput of BBR and New Reno flows with PFIFO as AQM.

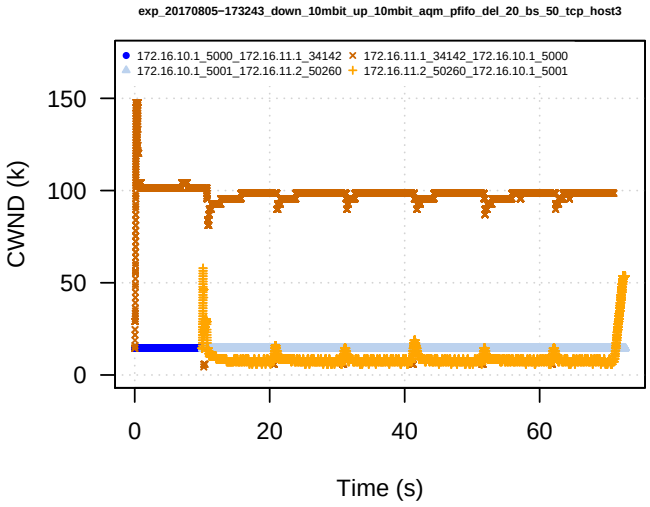

Fig. 11. CWND of BBR and New Reno flows with PFIFO as AQM.

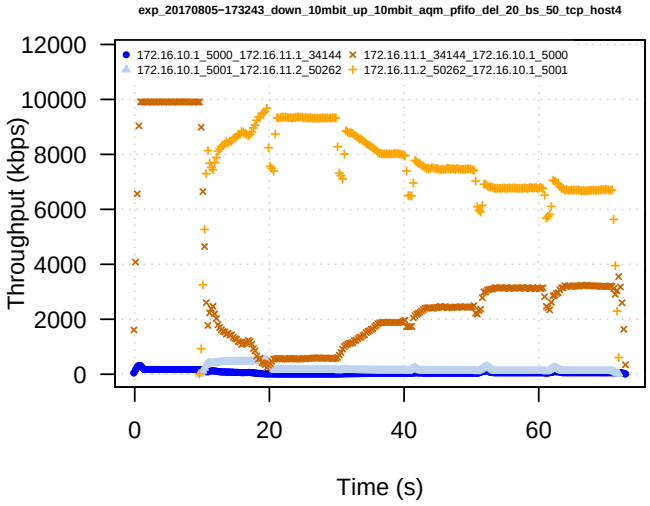

Fig. 12. Throughput of BBR and Vegas flows with PFIFO as AQM.

it uses a larger congestion window.

Because of the page limit, we included only Fair Queue and PFIFO AQMs. However, our tools support generating all of them.

We can conclude that BBR is very unstable with the use of PFIFO as AQM in comparison with Fair Queue although, for example, it performs fairly with the current default TCP congestion control variant - Cubic under Linux.

A. Generating the page

All the figures in the previous section were produced by `fab analyse_all`. After the analyses are finished, they are used in the paper, the unused plots can be removed, and the desired ones can be copied to a separate directory. To generate the webpage, the file containing the description of the paper (`description.txt`) should be prepared, or just a copy of the abstract could be written in the file. Then, a copy of the paper (e.g. `icncn2018.pdf`) should be prepared. To generate the complete webpage, simply the `publish` task is run as follows:

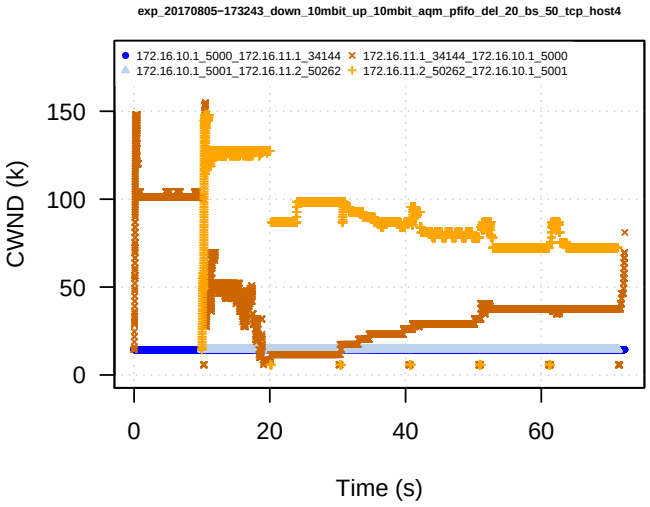

Fig. 13. CWND of BBR andd Vegas flows with PFIFO as AQM.

```
fab publish:find_dir=exp_20170804
    −173945/,out_dir="publish",paper="
    icnc2018.pdf",title="Computer−aided
    Reproducibility",desc_file="
    description.txt",source_link="https://
    github.com/screw/teacup",author="
    Marcel Marek marcelma@ifi.uio.no"
```

It contains a navigation menu at the left side and is separated into four main sections:

- Home navigates to the top of the page,
- Abstract contains the general description of the page or abstract from the paper,
- Reproduce this section contains all the figures with their description and the command to reproduce them,
- Conclusion provides space to summarize the results.

Moreover, the Reproduce section is further separated into up to four more subsections 1) TCP RTT 2) SPP RTT 3) Throughput 4) CWND. The number of subsections depends on the type of the figures found by the `publish` task.

The generated HTML page still requires manual input from the user. Each figure block in the HTML code contains a placeholder for its description. After editing these sections, the whole directory can be copied to user's www directory. Such an example can be found at the following address: http://heim.ifi.uio.no/marcelma/publish/page.html.

V. Related Work

Reproducible research ensures that scientific claims are published with their data and software code such that other researchers are able to verify the findings and build upon them [7]. However, there have been many challenges with reproducibility such as the lack of incentive for authors, double-blind review obfuscation, and anonymity [8]. As argued by [9], some additional challenges might rise such as author/artifact unavailability and lack of details during reproducing publications if reproducibility is not addressed well.

To address these challenges, some research work has been done; they mostly target other aspects of reproducibility rather than automation of generating reproducing artifacts. For example, what [8], [9] propose include motivating/obliging authors and venues to submit manuscripts with reproducible results; they try to unify the whole publication process with reproducibility requirements. Other types of motivating reproducibility, as claimed by [10], is to organize some regular international contest aiming at improving reproducibility skills of researchers and especially, students.

There is another type of work on reproducing research results in the literature: providing a (shared/public/customized) testbed. For example, the authors in [11] argue that shared, public testbeds are meant to produce reproducible results, but they might face challenges such as intolerance to outages and maintenances in the testbed, and they describe a methodology to overcome these challenges. WaIT [12] is another customized testbed aiming at providing a repeatable, low-cost platform for real-life experiments. The goal of [13] is to illustrate how reproducible research can be supported on distributed computing through resources selection, reservation, reconfiguring, monitoring and analyzing data.

VI. Conclusions and Further Work

In this paper, we introduced CAR – a facilitating tool to publish experiments results/artifacts. CAR is built upon TEACUP. We also described the necessary changes to the TEACUP testbed installation process to support BBR by both the TEACUP source code and TEACUP testbed. We also designed and extended the analysis part of TEACUP source code with `publish` task that makes it easier to publish results and enables easier reproducibility of them.

To show how CAR works, we conducted an experiment on BBR, and compared it with some other congestion controllers. CAR was shown a simple-to-use tool to collect results/artifacts and generate a website out of them. We believe that it could be used in more complex scenarios.

Apart from TEACUP, there is a web-based visualization tool called TEAPLOT (from version 1.0 integrated with TEACUP). It can produce 3D animated graphs via a user-friendly graphical interface. Unfortunately, it cannot save the result to a permanent file eg. pdf, png, etc. Next step for CAR would be to extend TEAPLOT functionality to be able to generate plots that could be used on a static HTML page. Another incremental step to extend the support of TEACUP capabilities could be to integrate it with all analyze tasks. Currently, generating time series graphs is only available. These all show us directions of our future work.

Acknowledgment

The authors were part-funded by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project (http://www.mn.uio.no/ifi/english/research/projects/ocarina/). The views expressed are solely those of the authors. Last but not least, we would like to thank our college Kristian Hiorth for his invaluable help with the debugging of throughput anomaly.

References

[1] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, no. 7604, pp. 452–454, 2016.

[2] S. Zander and G. Armitage, "CAIA Testbed for TEACUP Experiments Version 2," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150210C, 10 February 2015. [Online]. Available: http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf

[3] S. Zander, "TEACUP v1.0 - Command Reference," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529C, 29 May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529C/CAIA-TR-150529C.pdf

[4] S. Zander and G. Armitage, "TEACUP v1.0 - Data Analysis Functions," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529B, 29 May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529B/CAIA-TR-150529B.pdf

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct 2016. [Online]. Available: http://doi.acm.org/10.1145/3012426.3022184

[6] J. Heffner, M. Mathis, and R. Raghunarayan, "TCP Extended Statistics MIB," RFC 4898, May 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4898.txt

[7] V. Stodden, F. Leisch, and R. D. Peng, *Implementing reproducible research*. CRC Press, 2014.

[8] V. Bajpai, M. Kühlewind, J. Ott, J. Schönwälder, A. Sperotto, and B. Trammell, "Challenges with reproducibility," in *Proceedings of ACM SIGCOMM Reproducibility Workshop*. ACM, 2017.

[9] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, and G. Carle, "Towards an ecosystem for reproducible research in computer networking," in *Proceedings of ACM SIGCOMM Reproducibility Workshop*, 2017.

[10] M. Canini and J. Crowcroft, "Learning reproducibility with a yearly networking contest," in *Proceedings of ACM SIGCOMM Reproducibility Workshop*. ACM, 2017.

[11] S. Edwards, X. Liu, and N. Riga, "Creating repeatable computer science and networking experiments on shared, public testbeds," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 90–99, 2015.

[12] P. Brunisholz, E. Dublé, F. Rousseau, and A. Duda, "Walt: A reproducible testbed for reproducible network experiments," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE, 2016, pp. 146–151.

[13] L. Nussbaum, "Testbeds for reproducible research," in *REPPAR-2nd International Workshop on Reproducibility in Parallel Computing, held together with Euro-Par 2015*, 2015, p. 30.