ttprobe v0.1: Packet-Driven TCP Stack Statistics Gathering for TEACUP

Rasool Al-Saadi, Grenville Armitage, Jason But
Centre for Advanced Internet Architectures, Technical Report 150911A
Swinburne University of Technology
Melbourne, Australia
ralsaadi@swin.edu.au, garmitage@swin.edu.au, jbut@swin.edu.au

***Abstract*—TEACUP is an automated software tool that is used to run TCP/UDP experiments in a testbed with different operating systems including FreeBSD, Linux and Microsoft Windows. Internally, TEACUP uses many tools to emulate different networks conditions, generate network traffics, log different system and traffic statistics, analyse data, plot graphs and many other functions. To collect internal TCP information such Congestion Window Size (CWND) and smoothed TCP Round Trip Time (RTT), TEACUP uses operating system specific tools such as SIFTR in FreeBSD and Web10g in Linux. We developed ttprobe (TEACUP TCP probe), an alternative packet-driven TCP state logger for TEACUP experiments under Linux. At high packet rates ttprobe provides significantly more samples than web10g, with lower CPU overhead in many scenarios. ttprobe code is based on a loadable kernel module called TCP Probe.**

***Index Terms*—TCP Instrumentation, ttprobe, TCP probe, Web10g, TEACUP**

I. Introduction

Analysis and understanding network protocols are very important to improve network performance. Researchers are looking forward for tools that simplify these processes and to get more accurate and realistic statistics. TEACUP is one of these tools that makes studying and enhancing network protocols easier and required less time than before [1]. TEACUP provides a controlled environment for doing TCP experiments on a testbed. Providing testbed with a controlled environment is very important for doing TCP experiments, such as analysis congestion control algorithms, to get accurate and comparable results. TEACUP uses TCP loggers to capture internal TCP statistics and information from inside TCP/IP stack such as congestion window size (CWND) and smoothed TCP Round Trip Time (RTT). In FreeBSD, TEACUP uses SIFTR logger which can collect per packet statistics and it has been included in FreeBSD kernel since version 8.2-RELEASE [2]. On the other hand, in Linux environment, TEACUP uses Web10g [3] which is a kernel patch and user space tools that allows users to capture TCP statistics per time interval of 1 millisecond or more.

As an improvement to TEACUP and Linux TCP logger, we developed ttprobe logger (TEACUP TCP probe) which is a per-packet TCP statistics logger that can capture detailed and accurate TCP statistics. Moreover, we adapted TEACUP code to be compatible with our logger. ttprobe brings many benefits to TEACUP when it is used with Linux hosts including detailed statistics capturing and reasonable CPU overhead. ttprobe code is based on a loadable kernel module called TCP Probe [4].

II. TCP Probe

TCP probe is a Linux loadable kernel module that able to capture different information of TCP connections on each incoming packet. TCP probe hooks `tcp_rcv_established` kernel function by using kprobes framework [5], so the internal TCP/IP stack structures that contain TCP/IP variables, such as congestion window, slow start threshold (ssthresh) and acknowledge number (ACK), can be accessed and captured [6]. TCP probe also provides a basic filter that can be set up to capture only data of a specific TCP port number and/or when CWND is changed. When the module is loaded, it creates a virtual file, named tcpprobe, in `/proc/net` directory which is used to transfer TCP probe output to user space processes. Basically, TCP probe can be started using the following shell commands:

```
> modprobe tcp_probe port=0 full=1
> cat /proc/net/tcpprobe >/tmp/out.log
```

These commands will log TCP statistics for all TCP flows on each TCP packet arrive.

An indication of the data format as output by TCP probe can be seen in Table I. Figure 1 is a sample of the

TCP probe output for a small subset of data from a real experiment.

Table I
TCP PROBE OUTPUT HEADINGS

Column	Contents
1	Kernel Timestamp
2	Source_IP:port
3	Destination_IP:port
4	Packet Length
5	Send Next
6	Send Unacknowledged
7	Send window
8	Receive window
9	Congestion windows
10	ssthresh
11	Smoothed RTT

III. OVERVIEW OF TTPROBE V0.1

ttprobe (TEACUP TCP Probe) is a packet-driven Linux TCP instrumentation that can collect per-packet TCP statistics on each incoming and/or outgoing packet. ttprobe code is based on TCP probe source-code with many improvements to meet TEACUP requirements.

Getting accurate and detailed statistics are very important for protocols analysis. As TEACUP uses Web10g framework which is a time interval based method of a minimum of 1 millisecond, this causes TEACUP to miss many useful samples especially when link speed is high.

Another issue with Web10g is that it requires a patched kernel to be functional, and patching a kernel is highly depending on its version. The latest (at the report writing time) stable Web10g kernel patch is for Linux kernel 3.17. This means, currently[1], there is no applicable Web10g kernel patch for newer stable Linux kernels versions such as 4.0.9 and 4.1.3. Moreover, Web10g puts a high impact on CPU usage due to the time interval even when there is a very low traffic flow in the network.

On the other hand, ttprobe's per-packet data collection ensures virtually[2] no TCP state changes are missed. Furthermore, ttprobe consumes lower processing power than Web10g and it does not require patching the core kernel to work. However, some parts of ttprobe source code should be modified if there are substantial changes in the kernel socket structure and the hooked functions prototypes.

As ttprobe v0.1 is built on top of kprobes framework, ttprobe requires Linux kernel compiled with kprobes support to be functional. New versions of many Linux distributions, such as Ubuntu, Debian, CentOS and openSUSE come with kprobes enabled kernels.

IV. TTPROBE DEVELOPMENT

The main features that exist in ttprobe but not in TCP probe are:

1) The output timestamps are date/time timestamps (`timeval`) while in TCP probe are relative kernel timestamps (`k_time`). This change is very important to make TEACUP works properly with the output.
2) Hooking `tcp_v4_do_rcv` and `tcp_v6_do_rcv` (for incoming packets) and `tcp_transmit_skb` (for outgoing packets) instead of just `tcp_rcv_established` (for incoming packets). This change makes ttprobe able to log TCP information in all TCP connection states and on every incoming and/or outgoing packet.
3) Collecting additional TCP states such as maximum segment size (MSS) and TCP connection state.
4) Providing three output formats which are ttprobe, binary and web10g format.
5) Changing the module name and virtual file name to make ttprobe works without any interference with the original TCP probe module.
6) Implementing a buffer flushing function to flush TCP probe kernel buffer to user space buffer. This is very important to make sure that all the data inside ttprobe buffer is logged to user space buffer at the end of the experiment without any lost.

ttprobe provides many parameters that can be set up before starting the logger. A list of available ttprobe parameters is shown in Table II. The parameters have a format of *parameter=value* and must be written in the same command that is used to load the module as follow:

```
> modprobe ttprobe parameter1=value1
   parameter2=value2 ....
```

Table III shows columns descriptions of ttprobe output format. Binary format can be used to reduce log file size and CPU load. Our TEACUP patch all TEACUP to read binary format and ttprobe format natively. Moreover, we developed a small tool, called `ttpb2ascii`, that can

[1]This report was written on 2nd of September 2015

[2]When ttprobe's buffer size is too small and there is a very high traffic, ttprobe silently drops some samples.

```
1.622631348 192.168.1.101:22 192.168.1.100:52680 84 0x3e4fafe 0x3e4fafe 42 2147483647 65024 25300 34688
1.822848521 192.168.1.101:22 192.168.1.100:52680 20 0x3e4fb3e 0x3e4fafe 10 2147483647 65024 25300 34688
1.950667701 192.168.1.101:22 192.168.1.100:52680 84 0x3e4fb3e 0x3e4fb3e 10 2147483647 65024 47144 34688
1.957206390 192.168.1.101:22 192.168.1.100:52680 20 0x3e52d1a 0x3e4fb3e 10 2147483647 65024 47144 34688
```

Figure 1. A sample of TCP Probe output

Table II
TTPROBE PARAMETERS

Option	Description
`omode`	Output mode '0' for ttprobe format (default). '1' for binary output. '2' for web10g format.
`direction`	Capturing direction '0' on every outgoing packet. '1' on every incoming packet (default). '2' on every incoming and outgoing packet
`port`	Source or destination port number to match default: '0' (capture all port)
`bufsize`	ttprobe buffer size in packet default: 8192
`full`	Full log or just when CWND changed '0' ttprobe will capture samples just when CWND value is changed. '1' ttprobe will log on every packet (default).

Table III
TTPROBE OUTPUT HEADINGS

Column	Contents
1	Direction (i or o)
2	Timestamp
3	Source IP addr.
4	Source port no.
5	Destination IP addr.
6	Destination port no.
7	Packet counter
8	MSS
9	Smoothed RTT
10	Congestion windows
11	ssthresh
12	Send window
13	Receive window
14	Socket state
15	Send unacknowledged
16	Send next
17	Packet size

decode binary format log file and produce ttprobe ASCII format. The disadvantage of using binary mode is it increases the execution time of data analysis functions.

After loading ttprobe module, it will create a virtual file with path-name `/proc/net/ttprobe`. This file is used to read the log data from the kernel to a user space process, and to send commands to ttprobe module. ttprobe commands can be sent to the module from a shell using `echo` command.

```
> echo "command" > /proc/net/ttprobe
```

Currently, there are two commands that can be used with ttprobe which are `flush` command that used to flush ttprobe kernel buffer, and `finish` command that is used to send end of file signal to the reader process.

ttprobe was tested on Linux kernel 3.18 and 4.1, but it should work fine on any kernel version higher than 3.0.

Example of using ttprobe from Linux shell:

```
> modprobe ttprobe direction=2 omode=0
    port=5000 bufsize=16384 full=1
> cat /proc/net/ttprobe > /tmp/ttprobe.
    log&
> # waiting for an experiment to complete
> echo "flush" > /proc/net/ttprobe
> sleep 0.5
> echo "finish" > /proc/net/ttprobe
> rmmod ttprobe
```

V. EXPERIMENTAL COMPARISONS OF WEB10G AND TTPROBE V0.1

We did practical experiments to compare the details of captured data and the CPU load when Web10g or ttprobe is used. TEACUP tool was used in all our experiments.

The testbed that we used in the experiments includes three hosts, one bottleneck and a control host. The hosts and the bottleneck are normal PCs with Intel Core 2 Due @ 3GHz processes, 4GiB RAM and Gigabit and Fast Ethernet cards, and the control host is a Virtualbox virtual machine. Host 1 is connected directly to the

bottleneck while host 2 and host 3 are connected through a Gigabit Ethernet switch, and the switch is connected to the bottleneck. All machines having additional Ethernet cards to be connected to the control network. The host machines and bottleneck run Linux 3.17.4 while the control host runs FreeBSD 10.1. Figure 2 shows the network topology of the testbed that was used in our experiments.

A. Comparing the Details of the Captured TCP Data Samples

Firstly, we did two identical experiments (two individual runs) except that one has 1 ms Web10g poll interval and the other has 10 ms Web10g poll interval. In these experiments, the bottleneck shapes the traffic to 100 Mbps and emulates 6 ms RTT. Both Web10g and ttprobe ran together, iperf was used as a traffic generator and the congestion control algorithm is TCP CUBIC.

Tacking CWND values as a sample, we notice that web10g misses many samples in TCP rapid actions such as slow start stage and TCP congestion back-off event. These samples losses happen because TCP stack changes CWND size many times in one web10g poll time interval. As Web10g poll time interval is set to a high value, Web10g samples losses will become even bigger.

Figure 3 and Figure 4 show the first 1.4 second of CWND graphs for the first experiment (1 ms Web10g poll interval) and second experiment (1 ms Web10g poll interval) respectively. These figures illustrate that web10g missed many CWND values during the experiment even when the time interval was very short as opposite as ttprobe which captured all CWND samples. As a note, Web10g data points are plotted on top of ttprobe data points in all the graphs in this report.

Figure 5 and Figure 6 show zoomed in CWND plot of the first 30 ms of these experiments. These figures show more clearly the details of ttprobe CWND plot and how Web10g missed many data points especially during the slow start stage.

B. Comparing CPU Overhead of Web10g and ttprobe

In this section, our goal is to compare CPU overhead of Web10g and ttprobe. We utilised TEACUP tool with additional function that logs CPU utilization to get CPU usage in different scenarios. Additionally, `tcpdump` was disabled in TEACUP during the experiment to get CPU utilisation that related to the loggers as much as possible.

For both loggers, there is no way to get CPU overhead specifically for the logger. This because a big part of

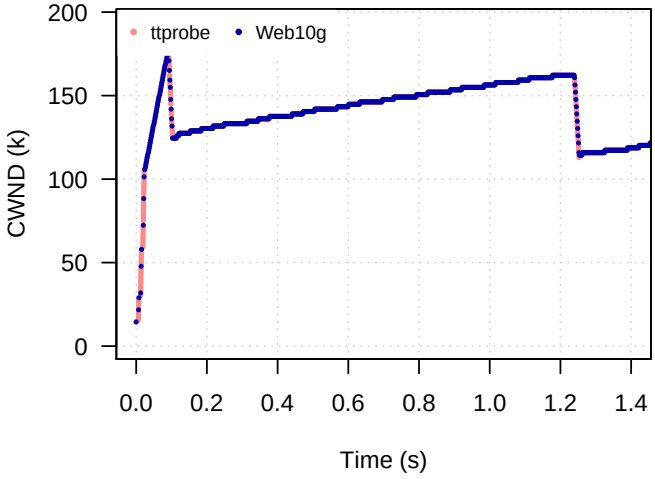

Figure 3. A CWND plot for a TCP flow captured using ttprobe and Web10g loggers at the same time (Web10g poll interval is 1 ms)

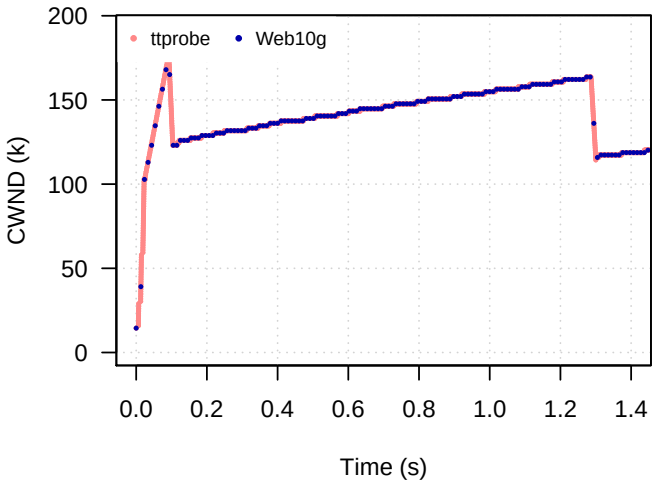

Figure 4. A CWND plot for a TCP flow captured using ttprobe and Web10g loggers at the same time (Web10g poll interval is 10 ms)

Web10g code is injected inside TCP/IP stack which is run as a part of the kernel, and the largest part of ttprobe is a kernel module that creates hooks to kernel functions (similar to a callback).

Obtaining comparative CPU usage from specific sections within the kernel is more than we require to do a simple comparison of how the choice of Web10g or ttprobe impacts on aggregate end-system load during TEACUP experiments.

Using averaged CPU load every one second is sufficient for comparison purpose as there are no big changes in the CPU load during the actual experiment load. System Activity Reporter utility (`sar`) , which it is part

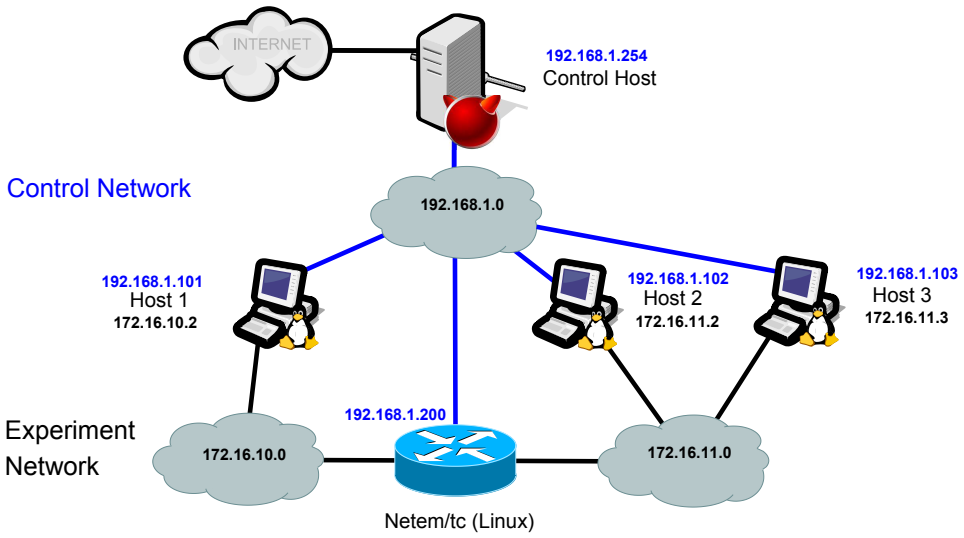

Figure 2. TEACUP Testbed topology

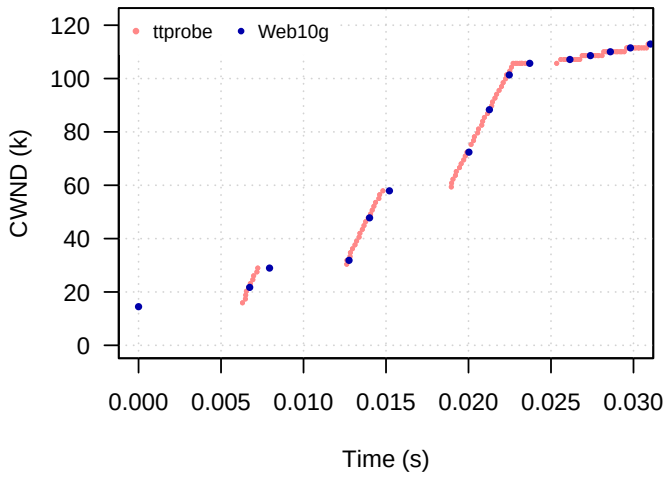

Figure 5. A zoomed in CWND plot for a TCP flow captured using ttprobe and Web10g loggers (Web10g poll interval is 1 ms)

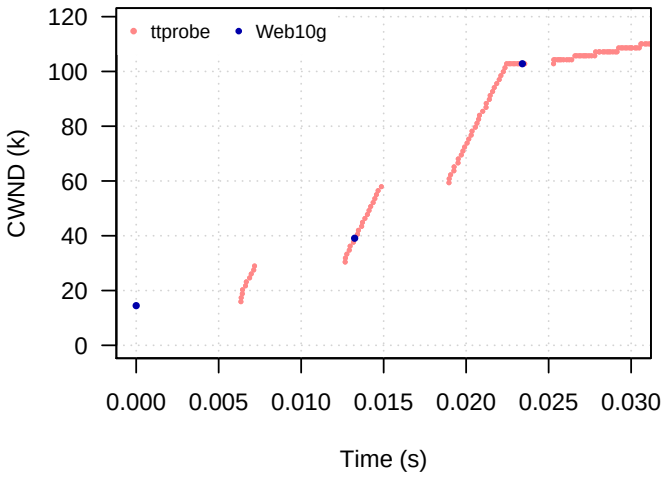

Figure 6. A zoomed in CWND plot for a TCP flow captured using ttprobe and Web10g loggers (Web10g poll interval is 10 ms)

of Sysstat Utilities [7], was used to log CPU usage during the experiments.

First, we did a TEACUP experiment to understand CPU load behavior during TEACUP experiment. This experiment ran for 60 second and it included one TCP flow transmitted between two computer (host 2 and host 3) connected through 1Gbps link.

Figure 7 shows the percentage of CPU usage relative to maximum CPU capacity of the sender machine when Web10g logger is used. CPU usage is including traffic generator and other default system processes. In this figure, we can see that there are three regions, TEACUP initialisation, experiment load and TEACUP finalisation regions.

During the initialisation stage, TEACUP configures the host, collects different information about the host, starts loggers and then starts traffic generators. During the finalisation stage, TEACUP stops traffic generators, stops loggers and collect log files. The important stage for comparing TCP loggers is the experiment load stage.

To compare CPU overhead of using Web10g and ttprobe, we did five TEACUP experiments of 60 seconds duration and 10 runs each. The first experiment was run without TCP logger, the second one with ttprobe logs just on receiving packets, the third one with ttprobe

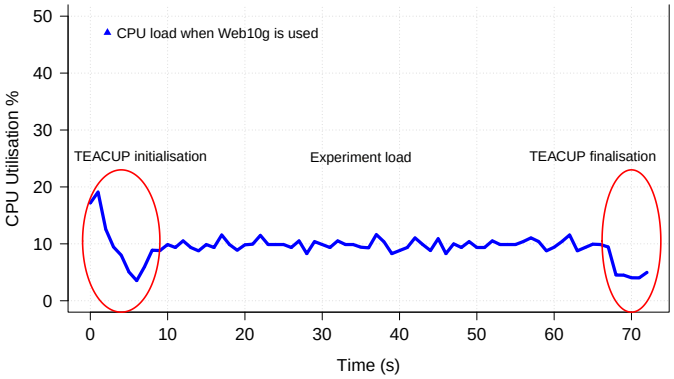

Figure 7. CPU usage % relative to maximum CPU capacity of the sender machine including traffic generator and other default system processes CPU utilisation when Web10g is used

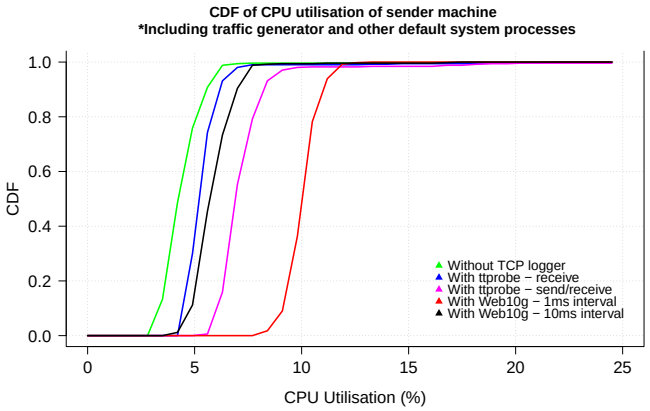

Figure 8. CDF plot of CPU utilisation for five experiments with different TCP logger in each experiment (link speed is 1Gbps)

logs on sending/receiving packets, the fourth one with Web10g logger and 1 ms poll interval and the last one with Web10g logger and 10 ms poll interval.

In these experiments, there were two PCs (host 2 and host 3) which having 1Gbps Ethernet cards and connected through 1Gbps network switch. This is considered the maximum throughput that can be achieved in our testbed. For each experiment, we extracted CPU utilisation for the experiment load period (t=15 s to t=65 s) of each run.

We then calculated cumulative distribution function (CDF) for each experiment using the extracted CPU utilisation data of the ten runs.

Figure 8 shows CDF of CPU utilisation for the four experiments. This figure illustrates that ttprobe (logging on sending/receiving packets) consumes less processing power than Web10g when Web10g poll interval is 1 ms, and slightly more than Web10g when Web10g poll interval is 10 ms at the same testbed condition. The figure also shows that ttprobe (logging on receiving packets) consumes less processing power than Web10g in all cases.

To understand how link speed affects CPU overhead, we replicated the five experiments but with traffic shaping of 100 Mbps and emulated base RTT of 6 ms. Figure 9 shows CDF of the five experiments with different loggers when the link speed is 100 Mbps. This figure illustrates that in this scenario, ttprobe overhead is highly depends on traffic speed.

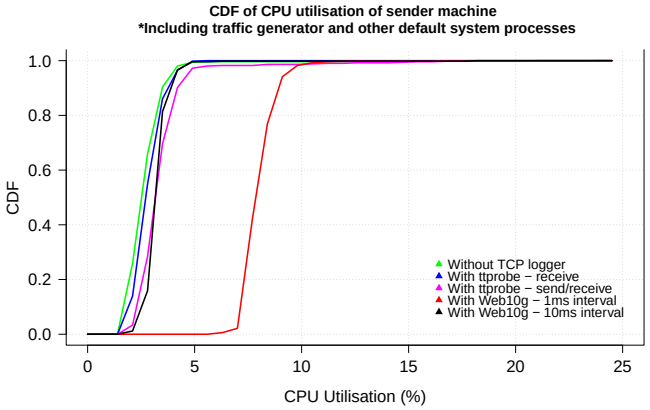

Figure 9. CDF plot of CPU utilisation for five experiments with different TCP logger in each experiment (link speed is 100 Mbps)

VI. Installation Procedures

To make TEACUP working properly with ttprobe logger, ttprobe module should be install in all TEACUP hosts and ttprobe TEACUP patch must be applied to TEACUP code in the control host.

A. *ttprobe Installation Procedure*

Before starting the compilation and installation process, the system must be updated and all software dependencies must be installed. The required packages to be installed are:

1) make
2) gcc
3) kernel-devel and/or linux-headers

The procedure of updating the system and installing the dependencies are depending on Linux distribution. Table IV shows dependencies installation commands for some Linux distributions. All commands must be executed by super user (root) account.

ttprobe module can be installed by using the following procedure:

Table IV
TTPROBE LINUX DEPENDENCIES INSTALLATION COMMANDS

Linux Distribution	Tested on	Commands
Ubuntu	14.04.2-desktop-amd64[a]	`>apt-get update` `>apt-get install make gcc linux-headers-$(uname -r)`
Debian	8.2.0-amd64	`>apt-get update` `>apt-get make gcc install linux-headers-$(uname -r)`
CentOS	7-x86_64	`>yum update` `>yum install make gcc kernel-devel kernel-headers`
openSUSE	13.2-x86_64	`>zypper update` `>zypper install make gcc kernel-devel`

[a]This version comes with all required dependencies installed by default.

1) Extract `ttprobe-0.1.tar.gz` archive file.
```
> tar xzvf ttprobe-0.1.tar.gz
```
2) Compile the module.
```
> cd ttprobe-0.1
> make
```
3) If all the dependencies are installed correctly, ttprobe.ko file should be created in the current directory.
```
> ls ttprobe.ko
```
4) Copy ttprobe kernel module to Linux kernel modules directory.
```
> mkdir -p /lib/modules/$(uname -r)/extra
> cp -r ttprobe.ko /lib/modules/$(uname -r)/extra/
```
5) Update modules dependency descriptions.
```
> depmod $(uname -r)
```

If all TEACUP hosts have same Linux kernel version, it is easier to compile ttprobe kernel module on one machine and copy it to all hosts machines (use steps 4 and 5 of ttprobe installation procedure).

B. Applying ttprobe's TEACUP patch

We created a patch for TEACUP v1.0 to support ttprobe logger natively. This patch allows TEACUP to start/stop ttprobe logger, and makes `analys_*/extract_*` functions working properly with ttprobe output file (file name ends with `_ttprobe.log.gz`). The procedure of applying the patch to TEACUP code is as follow:

1) Install TEACUP v1.0, if it is not already installed, by following the instructions in CAIA technical reports [1] [8].
2) Extract `ttprobe-0.1.tar.gz` archive inside the TEACUP directory <teacup_directory>[3] using the following commands:

[3]<teacup_directory> is the full path to your TEACUP installation.

Table V
TEACUP TPCONF VARIABLES FOR TTPROBE LOGGER

Option	Description
TPCONF_linux_tcp_logger	TCP logger to be used in Linux hosts 'web10g' web10g only (default). 'ttprobe' ttprobe only. 'both' to use ttprobe and web10g together.
TPCONF_ttprobe_direction	Capturing direction 'o' on every outgoing packet. 'i' on every incoming packet. 'io' on every incoming and outgoing packet (default).
TPCONF_ttprobe_output_mode	ttprobe output mode '0' for ttprobe format (default). '1' for binary output.

```
> cd <teacup_directory>
> tar xzvf ttprobe-0.1.tar.gz
```

3) Apply the patch to TEACUP.

```
> patch -p1 <  ttprobe-0.1/teacup-
   ttprobe-0.1.patch
```

VII. TEACUP CONFIGURATION

TEACUP configuration file (config.py) should include some additional TPCONF variables to setup ttprobe options. Table V lists TEACUP TPCONF variables that are used with our TEACUP patch.

If both loggers are chosen to be used in an experiment, LINUX_TCP_LOGGER environment variable must be set to either 'ttprobe' or 'web10g' in order to select which logger output will be used in TEACUP analyse_*/extract_* functions.

VIII. CONCLUSIONS AND FUTURE WORK

ttprobe module has many benefits over Web10g with respect to the details of the captured information, kernel patching and CPU overhead load. Moreover, the installation process of ttprobe is much easier than Web10g. ttprobe can easily integrate with virtually any Linux kernel with version higher than 3.0 compiled with kprobe support. However, Web10g collects more TCP statistics than ttprobe. For this reason, our update to TEACUP gives the choice to the users to select the desired TCP logger depending on their needs, as well as the option to use Web10g and ttprobe together. As a future work, ttprobe requires more development to add more TCP statistics to its output and improve its filters.

IX. ACKNOWLEDGEMENTS

We would like to thank Jonathan Kua for his help with testing ttprobe logger and providing valuable comments.

REFERENCES

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

[2] L. Stewart, "SIFTR - Statistical Information For TCP Research." [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools.html

[3] "The Web10G Project." [Online]. Available: http://web10g.org/

[4] "TCP Probe Kernel Module." [Online]. Available: https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/net/ipv4/tcp_probe.c?id=refs/tags/v4.1.6

[5] "KProbes." [Online]. Available: https://www.kernel.org/doc/Documentation/kprobes.txt

[6] Linux Foundation, "tcpprobe," 2009. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe

[7] "Sysstat Utilities." [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/

[8] 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, May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf