Performance Evaluation of Tcpdump

Farhan Jiva
University of Georgia

Abstract

With the onset of high-speed networks, using `tcpdump` in a reliable fashion can become problematic when facing the poor performance of modern operating systems. Because of this limitation, the amount of packet loss a user is willing to face is an important factor when choosing an operating system for a specific task. In this paper, we evaluate the performance of tcpdump on five modern operating systems.

1 Introduction

The reliable usage of `tcpdump` as a network analyzing/debugging/monitoring tool is arguably a non-trivial task. With the onset of high-speed networks, using `tcpdump` in a reliable fashion can become problematic when facing the poor performance of modern operating systems. It is because of this limitation that the amount of packet loss becomes an inevitable factor when choosing an operating system for a specific task. For example, a researcher who needs to extract binary executable files from a live network interface will require there to be zero packet loss in `tcpdump` when choosing an operating system (because the loss of even a single packet can be severely detrimental to his work).

In this paper, we provide a performance evaluation of `tcpdump` on five, popular, out-of-the-box installations of modern operating systems. The operating systems we evaluate are Ubuntu Server 10.10, Debian 5.0, FreeBSD 8.1, Fedora 14, and Windows 7. In performing the evaluation, we have found that tcpdump on Ubuntu Server and Debian performed relatively similar with respect to packet loss. FreeBSD performed the worst out of the operating systems tested, while Windows usually displays seemingly random spikes of packet loss. For each of the datasets we tested, we found that Fedora performs the best. The rest of the paper is laid out as follows:

- In Section 2 we discuss some related work.
- Section 3 covers our evaluation, along with our experimental setup and experimental results.
- In Section 4, we discuss our conclusions.

2 Related Work

One approach to increasing the performance of packet capturing is explained by Deri et al. [1] In their paper, they introduce a new type of socket (`PF_RING`) based on circular buffers which is optimized for packet capturing. They show that by making certain modifications to the Linux kernel, they were able to decrease packet loss substantially, even on commodity hardware. Whereas Deri et al. provide insight on methods for increasing the performance for packet capturing, the work presented in this paper provides only a survey of how modern operating systems stand-up to packet loss, and makes no attempt to increase or decrease performance.

3 Evaluation

3.1 Experimental Setup

In the following section, we will describe our experimental setup. In addition, we will discuss the datasets that were used for the evaluation, as well as provide some statistics about them.

Testing platform. To carry out our evaluation, we chose a variety of operating systems. Included were Ubuntu Server 10.10, Debian 5.0, Fedora 14, FreeBSD 8.1, and Windows 7. Each operating system was installed on partitioned hard disks spanning across two machines. These machines contained Intel®Core™i7 processors (2.80Ghz) and contained 8GB of DDR3 memory. Each also had a 1 terabyte SATA hard disk running at 7,200 RPMs. After the operating systems were installed, no patches or updates were performed.

Datasets. For our evaluation, we used a total of 4 datasets. These datasets were in the form of packet-capture (pcap) files, each containing a different type of activity that a user would normally perform on the Internet. Each dataset was generated by us and was captured using `tcpdump`:

- **HTTP dataset.** This dataset contained a web browsing session. It contained mainly HTML and image content.
- **Flash dataset.** This dataset strictly contained flash content. The content was gathered while browsing Youtube and Trance.fm.
- **Skype dataset.** This dataset contained Skype Voice-over-IP traffic.
- **Mixed dataset.** This dataset was a combination of the above three datasets. The program `mergecap`[1] was used to combine these datasets.

Figure 1 displays the breakdown of TCP and UDP packets for each of the dataset. Most packets in the HTTP and Flash dataset contained TCP packets, whereas the Skype dataset consisted almost entirely of UDP packets. Much of the UDP packets in the HTTP dataset were related to DNS queries.

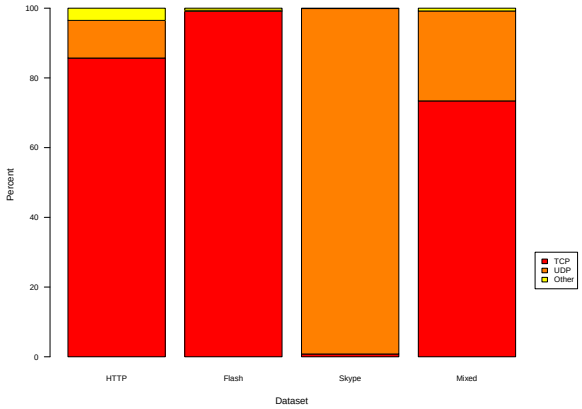

Figure 1: Breakdown of UDP and TCP packets for each dataset

About 75% of the Mixed dataset were made up of TCP packets while the other 25% percent were made up of UDP packets.

Distribution of packet lengths. To get a better idea of the frequency of packet lengths in each dataset, we provide frequency distribution graphs. Note that each of these frequency distribution graphs is binned at every 100th interval. Figures 2 and 3 show the frequency distribution for the HTTP and Flash dataset, respectively. As shown in these figures, these two datasets follow a similar distribution pattern with respect to UDP and TCP packets. In the TCP distribution, it can be seen that for both of these datasets, the packet lengths 1518 bytes (maximum length of an Ethernet frame) and 66 bytes occur most frequently.

Figures 4 and 5 show the frequency distribution for the Skype and Mixed dataset, respectively. As mentioned earlier, the Skype dataset is made up almost entirely of UDP packets. This is because Skype's protocol is based on UDP. In this dataset, UDP packets of length 1429 bytes occurred most frequently. The maximum UDP packet lengths for the Skype dataset are noticeably larger than the ones seen in HTTP and Flash dataset. These packets likely contain the voice data. It can also be seen from this distribution that there is a frequency spike in the 0-200 byte range. It is speculated that Skype's control traffic has packet

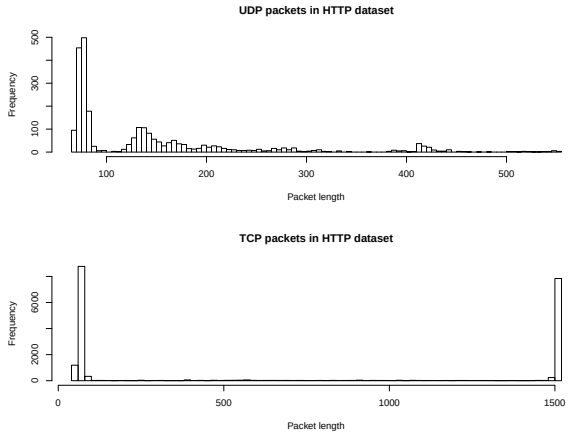

Figure 2: Frequency distribution of packet lengths in the HTTP dataset

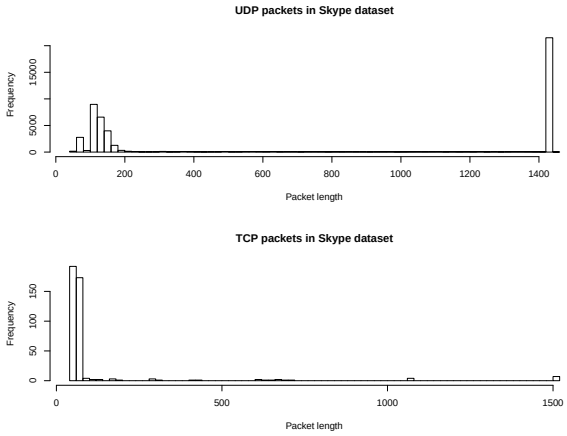

Figure 4: Frequency distribution of packet lengths in the Skype dataset

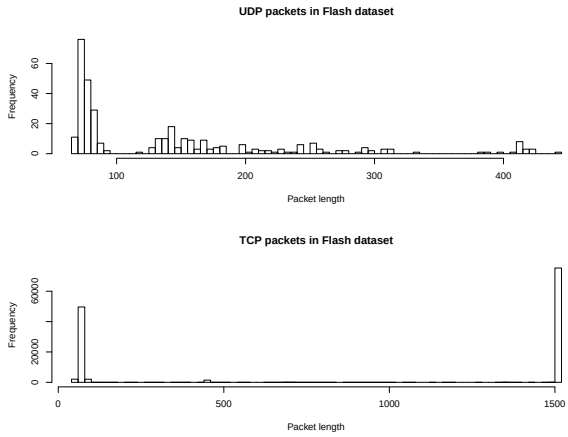

Figure 3: Frequency distribution of packet lengths in the Flash dataset

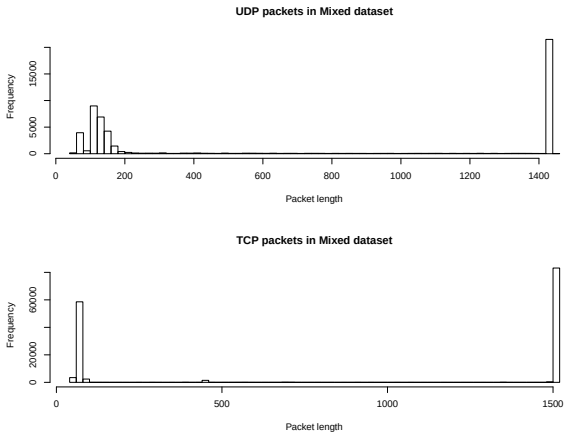

Figure 5: Frequency distribution of packet lengths in the Mixed dataset

lengths in this range. As shown in Figure 5, the frequency distribution of the Mixed dataset is a combination of the other frequency distributions.

Performance tests. To conduct the performance tests on `tcpdump`, we set up a mirror port on a Cisco 2950 Gigabit Switch. For each of the operating systems, we set `tcpdump`[2] to listen and capture[3] on this mirror port. On another port, we used `tcpreplay`[4][5] to replay each of the datasets over the network. `tcpreplay` can be configured such that it will replay the pcap file at a given bitrate. For our evaluation, we used bitrate intervals of {100Mbps, 200Mbps, ..., 900Mbps, topspeed}. `topspeed` is a command-line switch that can be set in `tcpreplay` which tells it to transmit the pcap file "as fast as it can." For each of these bitrate intervals, we run 10 rounds of tests and gather the statistics-output from `tcpdump`. This output contains information regarding the number of dropped packets, the number of packets captured, and the number of packets received by the filter. Using this information, we define a formula for the packet loss ratio (`PLR`):

$$PLR_i = \frac{NumberOfDroppedPackets_i}{NumberOfPacketsReceivedByFilter_i}$$

where *i* is a round from {1-10}. For each 10 rounds of tests for each bitrate interval, we calculate

the PLR. From these 10 PLRs, we select the median value along with its corresponding bitrate interval and use this information to report our results. The actual setup for this experiment is described as follows:

- Start `tcpreplay` on looped-mode at a particular bitrate interval on one machine.
- Shortly after, start `tcpdump`(with the -c flag set to 100000 and -s flag set to 1518)[67] on the second machine which is connected to the mirror port.
- `tcpdump` is run for 10 rounds at this particular bitrate interval, saving the captured packets onto disk.
- At the end of each of these 10 rounds, calculate the PLR.
- Continue for all of the bitrate intervals.

The following table provides information on the kernel version, `tcpdump` version, as well as the `libpcap` version for each of the operating systems.

Operating System	**Kernel**	**Tcpdump version**	**Libpcap version**
Ubuntu Server 10.10	2.6.35-22-server	4.1.1	1.1.1
Debian 5.0	2.6.26-2-amd64	3.9.8	0.9.8
Fedora 14	2.6.35.6-45.fc14.x86_64	4.1.1	1.1.1
FreeBSD 8.1	8.1-RELEASE	4.0.0	1.0.0
Windows 7	N/A	3.9.5*	4.1.2**
*This is the WinDump version. It is based on tcpdump version 3.9.5			
**This is the WinPcap version. It is based on libpcap version 1.0 branch 1_0_re10b			

Measuring discrepancies in tcpreplay data transfer rates. During our evaluation, we learned that the bitrate which is specified through `tcpreplay` is not the true bitrate at which it transfers data. To overcome this discrepancy, we conducted a small experiment to observe the differences between specified bitrate speeds in `tcpreplay` and actual bitrate speeds.

For this experiment, we measured the discrepancy in `tcpreplay`'s bitrate by using `tcpdump`. This experiment was done using a single network interface on one of the machines described above. By using some meta-information that `tcpdump` provides (byte size of captured pcap file, time it took to capture the file), we can use it as a tool to test the actual speed it took for the capture to take place. We explain the steps of the experiment below:

- Start `tcpdump` on the network interface.
- Start `tcpreplay` at a specified bitrate on the same network interface
- Once `tcpreplay` is done transmitting, exit `tcpdump`
- Using the byte size of the captured pcap file along with calculating how long the transfer took to finish, report the calculated bitrate.

A few important things to note with this experiment is that the process of starting and stopping `tcpdump` along with starting `tcpreplay` was scripted to remove delays caused by human error. Also, we determined that this method for determining bitrate was accurate because for `tcpdump` to capture packets sent by `tcpreplay` is just a matter of copying a buffer from kernel space (and the rate at which this buffer is copied is negligible). One last thing to note about this experiment is that `tcpdump` saved the captured packets on a ramdisk, to avoid delays caused by writing to the disk.

For each of the bitrate intervals mentioned above, we ran this experiment 5 times and averaged the bitrates inferred from `tcpdump`. We also averaged the bitrates which `tcpreplay` reports in its output. Using these two values along with the bitrate speed which we specified in `tcpreplay`, we show the results in Figure 6. As we can see from this graph, `tcpreplay` is accurate when

the specified bitrate reaches to about 300Mbps. After this point, the bitrate which is specified is no longer the true rate at which it transfers at. Another thing to note, however, is that the bitrate given in the output of `tcpreplay` is very close to the actual bitrate that it sends at. We speculate that this inconsistency in `tcpreplay` is due to the overheads involved of it trying to rate-limit itself when trying to match the specified bitrate.

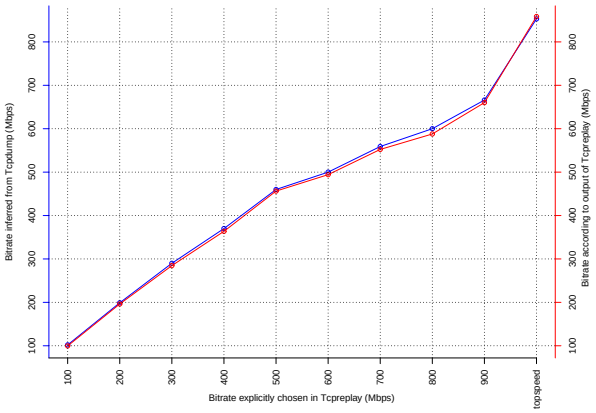

Figure 6: Comparison of bitrates between tcpreplay and tcpdump

3.2 Experimental Results

In this section, we discuss our findings. Namely, we will discuss how `tcpdump` fared on each of the operating systems we tested for each of the datasets. It is important to note that the graphs in this section are based on the bitrate speed which we specified in `tcpreplay`, and not the actual bitrate. The true bitrate can be found using Figure 6.

HTTP dataset. In Figure 7, we show our results for the HTTP dataset. As we can see, all of the operating systems maintained less than 1% packet loss until about the 300Mbps mark. After this mark, we see that Debian, FreeBSD and Ubuntu had a steadily increasing `PLR`. Windows encountered a relatively low `PLR`, while Fedora's `PLR` was negligible. We can see that at `topspeed`, nearly all of the operating systems witnessed a rapid-drop rate.

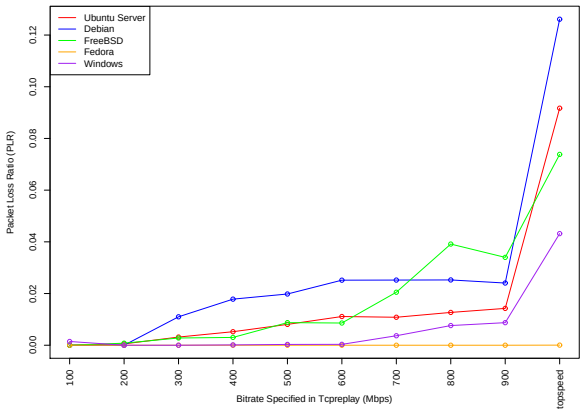

Figure 7: Tcpdump packet loss for HTTP traffic

Flash dataset. We can see the results for the Flash dataset in Figure 8. As we can see from this graph, Debian, Ubuntu, and Fedora performed relatively well. Windows had a substantial, seemingly anomalous drop-rate at the 600Mbps interval, but stabilized for the subsequent bitrate intervals. FreeBSD maintained a relatively low `PLR` until the 300Mbps interval, at which point it encountered a relatively rapid drop-rate.

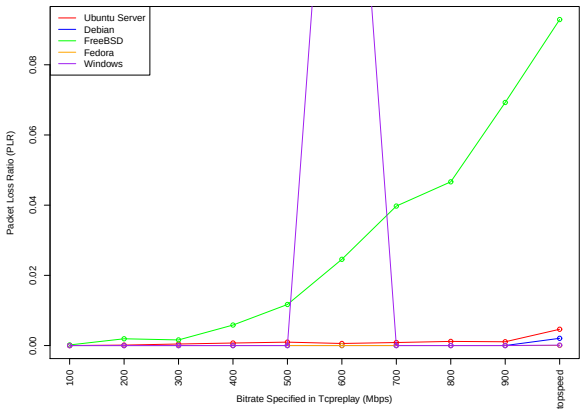

Figure 8: Tcpdump packet loss for Flash traffic

Skype dataset. Figure 9 shows the results for the Skype dataset. As we can see in this figure, nearly all of the operating systems performed relatively well until around the 400Mbps interval. Windows had another drop-rate at around the 200Mbps interval, however, this drop in `PLR` was

not nearly as substantial as it was in the Flash dataset. Windows remained relatively stabilized after this initial drop-rate. We can see the Debian and Ubuntu had a steady increase in PLR, while FreeBSD had a another relatively rapid increase in PLR after around the 400Mbps interval.

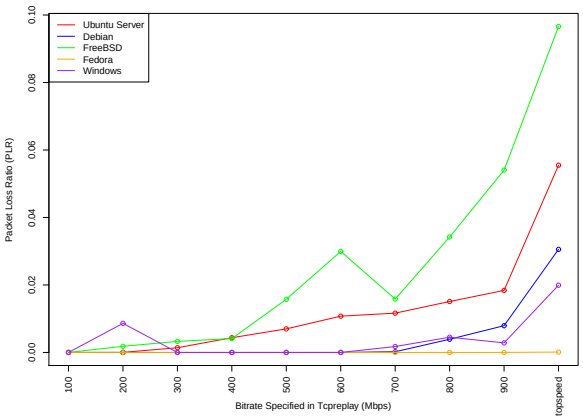

Figure 9: Tcpdump packet loss for Skype traffic

Mixed dataset. We look to Figure 10 to get a better idea of how well `tcpdump` will fare in the real world on these particular operating systems. This figure contains the results for the Mixed dataset. We can see that overall, Debian and Ubuntu performed relatively similar (most likely due to Ubuntu being based off of Debian). FreeBSD had a substantial amount of packet loss starting at around the 400Mbps interval, while Windows encountered another spike in drop-rate at around the 700Mbps mark. Fedora, again, had a near-negligible amount of packet loss.

4 Conclusions

In this paper, we have presented a performance evaluation on `tcpdump` using 5 popular operating systems (Ubuntu Server 10.10, Debian 5.0, FreeBSD 8.1, Fedora 14, Windows 7). From our experiments, we found that tcpdump on Ubuntu Server and Debian performed relatively similarly (likely because Ubuntu is based off of Debian) with respect to PLR. We have shown that out of these operating systems, FreeBSD performs the worst (usually exhibiting substantial packet loss at around 400Mbps),

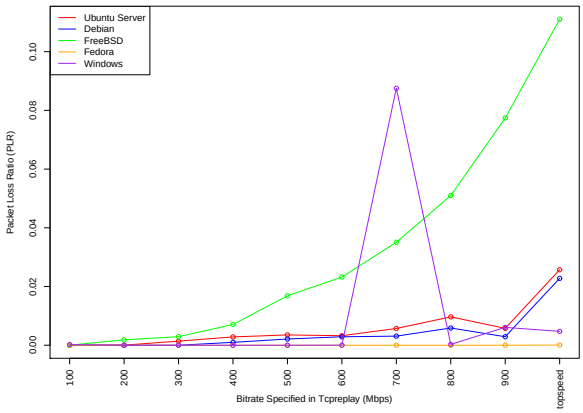

Figure 10: Tcpdump packet loss for Mixed traffic

while Windows usually displays seemingly random spikes of packet loss. In our tests we found that for each of the datasets used, Fedora 14 performed the best, exhibiting a PLR of around 0 for each of the tests.

References

[1] Luca Deri, *Improving Passive Packet Capture: Beyond Device Polling*. Proceedings of SANE 2004, October 2004.

Notes

[1] mergecap is available at http://www.wireshark.org/

[2] Windump was used in-place of tcpdump on Windows. Windump is based on the same library as tcpdump. (http://www.winpcap.org/windump/)

[3] The replayed packets were saved as a pcap file on disk

[4] tcpreplay is an application for replaying pcap files into the ether (http://tcpreplay.synfin.net/)

[5] To do the replaying of pcaps, tcpreplay was run on Fedora 14

[6] We use the `-c` flag in tcpdump so that our testing process can be scripted to avoid human delay

[7] We set the snap-length of the capture to 1518, which is the maximum length of an Ethernet frame