High Speed Packet Logging on a Budget

Chad D. Mano, Jeff Smith, Bill Bordogna, and Aaron Striegel*

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{cmano, jsmith30, wbordogn, striegel}@nd.edu

Abstract. Creating high quality network trace files is a difficult task to accomplish on a limited budget. High network speeds may overburden an individual system running packet logging software such as tcpdump, resulting in trace files with missing information and making analysis difficult or incomplete. High end specialized systems may perform the job well, but may be out of reach due to financial constraints. To the end we developed the *Cheap Logger* (CLog) system which utilizes inexpensive COTS hardware to create a high quality, complete network trace files. A scalable distributed storage system enables the CLog system to expand and continue to create high quality, complete network data trace files even at extremely high data rates.

1 Introduction

An important aspect of some areas of computer network research is the ability to perform analysis in a large-scale network environment. However, most network administrators would be reluctant to allow an experimental device to be incorporated into a live enterprise network, particularly in critical areas such as the gateway to the Internet. This leaves laboratory simulation as the only remaining option to perform large-scale network analysis.

Software packages such as *ns-2* [10] are useful for simulations where performance is the key issue being addressed, but falls short in the ability to create a highly accurate representation of actual data flow within a large scale network. An accurate representation of traffic is essential where packet payload analysis is needed such as in virus and worm detection research [8, 11].

A solution to the need for an accurate representation of network traffic is to capture and store live network data. On a small scale this can be easily accomplished with a standard desktop computer and an application such as tcpdump [6]. However, as network capacity increases it becomes difficult to keep up with line speeds resulting in an incomplete network trace. Powerful systems designed for high speed packet logging are available, but may break the budget of a research group [1, 5].

* This research was supported in part by the National Science Foundation through the grant CNS03-47392.

This financial hurdle led to our development of a high speed network packet logger which can be built for a fraction of the cost of a commercial system. The *Cheap Logger* (CLog) system is built from inexpensive hardware and can easily be scaled to meet increased demands for logging speed. This paper describes the CLog system architecture, associated utilities, and presents a performance analysis of the system.

The remainder of the paper is organized as follows. Section 2 briefly presents the motivation and background for the project. Section 3 details the architecture of the system including communication protocols developed for management of the system. Section 4 analyzes the performance capabilities of the system. Finally, Section 5 summarizes the work.

2 Motivation and Background

For various endeavors in our research group, a trace of live network traffic is needed to measure the performance and scalability of systems which have been developed. Live data is accessed via a tap of the University of Notre Dame Internet gateway. The network tap comes from a fiber gigabit link at the edge of the Notre Dame network which feeds an OC-12 line to the University's ISP via multiple 100Mb/s connections. The tap point averages just over 130Mb/s utilization.

In the past, an HP zx2000 Itanium2 workstation was used as the packet logging system. Even this relatively fast desktop system would drop a significant number of packets leaving an incomplete trace file of network traffic. The workstation was assumed to be powerful enough to keep pace with the speed of the Internet tap and through simple experimentation and analysis this proved to be the case. The bottleneck, it was discovered, proved to be the hard disk (15k rpm SCSI drive) which could not keep up with amount and speed of the traffic coming from the tap.

To ease the disk write speed problem, we developed a solution that distributes data writing tasks over multiple systems, thus relieving the bottleneck created by a single system logger. Two important requirements for the system is that it should be inexpensive and scalable. The components of the system are typical *commercial-off-the-shelf* (COTS) hardware, making the creation of the system affordable. The client/server architecture makes the system scalable as additional inexpensive clients may be dynamically added to the system as network capacity increases.

3 Architecture

The physical architecture of the CLog system is a standard client/server model. The server acts as the gateway for the network tap and forwards data to be logged to each individual client system as illustrated in Figure 1. A Sun dual Opteron 244 workstation with 1GB or RAM was utilized as the server for CLog. It was desireable for the clients to be physically small as there would be multiple client machines and the entire CLog system needed to fit on a cart that could

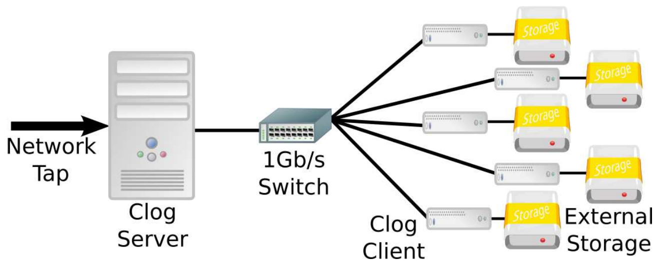


Fig. 1. Illustration of the Cheap Logger system.

be moved from place to place. Therefore, Apple 1.25 GHz Mac mini systems running Darwin Kernel Version 1.9.0 were utilized as the client machines for the system. An external LaCie Firewire 7200 RPM 500GB hard drive was attached to each Mac mini for data storage.

The remainder of this section details the individual components of the CLog system and the communication protocols designed to maintain data consistency and incorporate management capabilities.

3.1 Server

The server provides the central control of the system and is the gateway for the network tap data. The server must have at least two Ethernet ports and a third if remote access is required to operate the system. Remote management is simply performed by establishing an SSH session from a different system. We utilize the on-board 1Gb/s Ethernet port for remote management and installed an Intel Pro/1000 MT dual port 1Gb/s NIC for the required ports.

The data flow of the network tap traffic is one-way as the CLog system does not inject data back into the network. Therefore, the port which is connected to the network tap is designed as the *inbound port* (i-port) and the remaining port is designated the *outbound port* (o-port). The o-port is connected to a 1Gb/s switch which, in turn, is connected to each client system.

The processing of the network tap data is minimal as the server must keep up with the speed of the incoming packets. The data flow of within the server is illustrated in Figure 2. When a packet is captured on the i-port the server immediately writes the packet to tail point of a cyclical buffer. A separate thread removes packets from the head point of the buffer and overwrites the destination MAC address in the Data Link header of the packet. The packet is then transmitted on the o-port and is forwarded on to one of the client systems.

It is possible to implement a load balancing scheme to enhance the overall effectiveness of the distributed writing system. However, in an effort to minimize the processing requirements of the server a simple round-robin system is utilized to determine the destination client of each packet. The round-robin method is essentially a *least recently used* queue of all client systems.

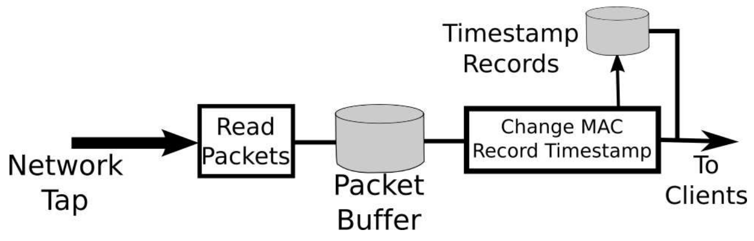


Fig. 2. Diagram of the data flow within the Cheap Logger server.

In cases where analysis is only concerned with capturing complete packet traces this process is sufficient. However, in most cases exact ordering of packets and accurate timing records are essential. This is accomplished by the server taking additional processing steps. This process will be discussed following the introduction of the client system.

3.2 Client

The clients perform the "physical labor" of the system by writing the network data to disk. A custom packet logger application, similar to tcpdump, was developed using the Libpcap [4] C library. The packet logging process is discussed here with the server communication for management purposes to be presented later in this section.

After a client is connected to a server it is able to begin logging packets. In order to do this efficiently a packet buffer and a multi-threaded disk writing process is utilized as illustrated in Figure 3. Each incoming packet is stored in a single packet buffer. A parameter setting determines the maximum size of the buffer based on the number of packet contained in the buffer. When the buffer reaches the packet number limit a new thread is created which takes the data in the packet buffer and writes it to disk.

The main thread is able to continue to capture new packets while the disk writing takes place. The new packet writing thread is placed in a queue of all packet writing threads. A new thread is created each time the packet buffer is filled to prevent packet loss which may occur if a single thread was expected to perform all writing responsibilities. In such a case the thread may be busy writing and unable to collect data from the packet buffer at the instant it is needed.

The log files are stored in tcpdump format. This allows the final trace files to be analyzed using existing applications such as such as tcpdump or ethereal [2]. As part of this logging format, a timestamp header is stored with each packet. The timestamp is important for any timing related analysis or to simulate the data flow at a later time using an application such as *tcpreplay* [7]. It is critical, therefore, to maintain precise timing data for the entire system. This property is maintained by the timestamp synchronization process.

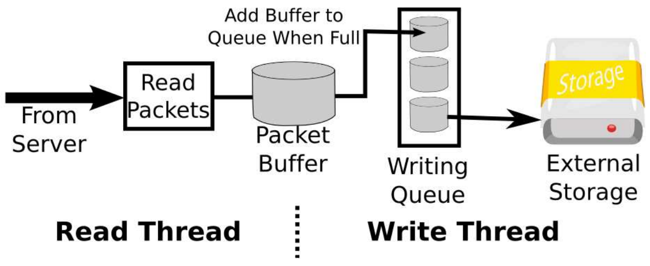


Fig. 3. Illustration of the Cheap Logger client system.

Timestamp Synchronization Timestamps are created when a new packet arrives on a packet logging system using the libpcap library. In the Clog system this results in two timestamps being associated with each packet. First, when a packet is received by the server a timestamp is created. This timestamp is the most precise in relation to other packet timestamps as all timestamps at this point are based on a single clock and are recorded prior to processing within the CLog system. The second timestamp is recorded on the client systems, each using the local clock to determine the timestamp value.

This second timestamp is not useful as clocks most certainly vary slightly, even if attempts are made to synchronize the clocks with a common time server [9]. In addition, the timestamps generated on the clients are created after the packets have spent a non-trivial amount of time due to processing and transmission in the server system. The time synchronization solution requires the original timestamps to ultimately replace the timestamps recorded on the client systems.

Libpcap headers (including timestamps) are prepended to the packets and are not actually part of the raw packet. Therefore, timestamps cannot simply be added to each existing packet header prior to forwarding the packet to the client. In addition, appending timestamps as a footer to the payload of each packet is not possible due to MTU restrictions. Even without the MTU restriction, the overhead of updating existing packet header information (size information and checksums) may prove to create a new bottleneck for the system.

Our solution was to log timestamps as generated on the server and periodically forward the log to the associated client. Figure 4 illustrates this process. When a packet is received by the server, a timestamp is immediately generated. Once the destination client has been determined for the packet the timestamp is recorded in a buffer. An individual timestamp consists of two parts, the number of seconds (*tv_sec*) since the Epoch (00:00:00 UTC, January 1, 1970) and the number of additional microseconds (*tv_usec*). When the first timestamp is recorded in the log, four items are recorded. The value *tv_sec* is recorded as a global value for the log meaning this value is stored only once for all packets which will be part of the log. Next the value *tv_usec* is recorded along with two additional bytes from the header of the packet. These last three items are logged for each packet added to the log.

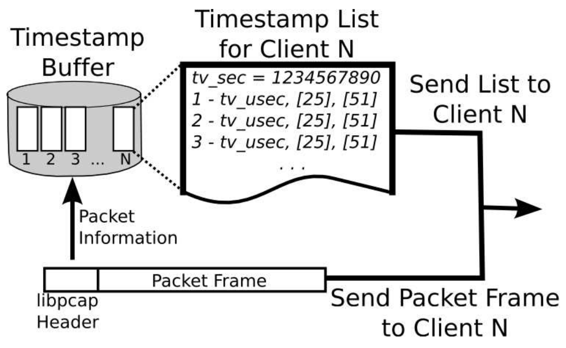


Fig. 4. Illustration of the timestamp system.

The two bytes are used as a simplistic method to match the timestamp with the appropriate packet stored on the client. The bytes are designated as an offset from the beginning of the packet frame and through empirical analysis values of 25 and 51 were found to be effective. An analysis of the network trace shows that in 99.92% of the time the combination of these two bytes are unique from the neighboring ten packet frames. The packets and timestamps are stored sequentially, therefore, any ambiguity between packet frames in which these two bytes match is easily removed by comparing neighboring values.

A single timestamp log is forwarded on to a client if either the log becomes full or if a timeout period expires. The timeout period is essential as the log stores a global *tv_sec* value for all packets in the log. The timeout period is set to be less than one second which restricts a log from containing anything else besides packets with an identical *tv_sec* value or a value one second greater than the first packet stored in the log. This allows a post-processing step to determine, without ambiguity, the exact timestamp of the packet.

The Libnet [3] C library is used to create the timestamp log packet to send to the client. The *Ethertype* field of the Ethernet header is modified to a custom value enabling the post-processing step to identify timestamp log packets. The post-processing procedure can be implemented in one of two ways. First, the timestamps can be corrected on the trace files from a single client. A packet sorter can then be applied later to merge the files from all clients. The other option is to process trace files from all clients simultaneously which results in new trace files containing the merged data from all files.

3.3 Communication

Communication between the server and clients is required for two purposes: for the discovery of clients in the system, and for statistical updates throughout the capture. This communication protocol is efficiently implemented using the libnet library and handles all communication via Ethernet addressing and therefore does not require IP layer processing. The basic communication structure is illustrated in Figure 5.

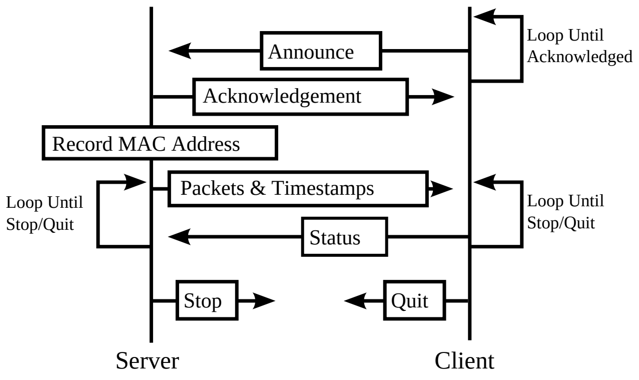


Fig. 5. Illustration of the communication protocol between clients and the server.

A client, when ready to begin logging, broadcasts an announce (ANN) message and continues to do so until a response is received. The server listens on the o-port for ANN messages and responds with an acknowledgment (ANN_ACK). The server then adds the MAC address of the client to the LRU queue for packet delivery. For management convenience, the client includes a name with the ANN message enabling the server to display a name, rather than simply and I.D. number or MAC address on the management interface.

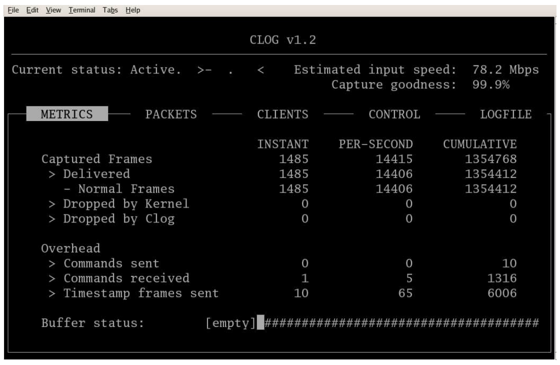
Upon receiving the ANN_ACK message the client terminates the broadcast ANN message, but continues to send statistical updates periodically via STATUS messages. These status messages enable the system administrator to view logging statistics for each client individually as well as system statistics such as disk usage. Details on the statistical reports and management interface will be detailed shortly.

Client systems must quickly check the *Ethertype* field of each Ethernet header in order to effectively process management communication as network tap data arrives on the same data port. When the capture is complete the server can send a STOP message to the client, notifying the client that additional packets will not be delivered. In addition, a client may also send a QUIT message if it must be removed from the system for some reason.

3.4 Management Utility

The management utility is designed to give an administrator the ability to view statistics regarding the data capture and to perform basic functionality such as starting and stopping the capture. A detailed description of each feature of the utility is unnecessary here, but illustration of a few management screens will give a general idea of the reporting system.

Figure 6 shows the overall statistics of the current capture. The most important feature is the detail of the number of packets dropped, categorized by packets dropped by the system kernel and those dropped by CLog. These stat-

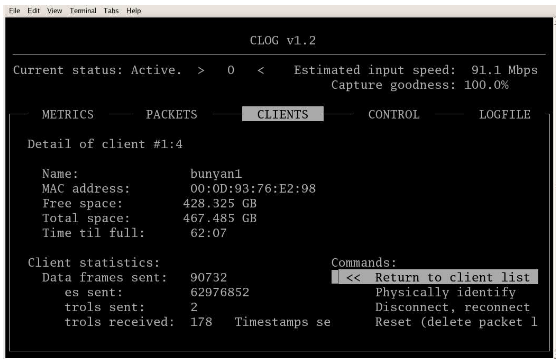


```
                              CLOG v1.2
Current status: Active.  >-   .   <    Estimated input speed:  78.2 Mbps
                                            Capture goodness:  99.9%

    METRICS   PACKETS    CLIENTS    CONTROL    LOGFILE

                                  INSTANT    PER-SECOND    CUMULATIVE
    Captured Frames                  1485         14415       1354768
     > Delivered                     1485         14406       1354412
       - Normal Frames               1485         14406       1354412
     > Dropped by Kernel                0             0             0
     > Dropped by Clog                  0             0             0

    Overhead
     > Commands sent                    0             0            10
     > Commands received                1             5          1316
     > Timestamp frames sent           10            65          6006

    Buffer status:         [empty] #####################################
```

Fig. 6. Screenshot of the overall system status report screen.

ics enable an administrator to quickly identify the existence of a problem if the percentage of dropped packets increases to unexpected levels.

Figure 7 details the statistics of individual systems including total disk usage. This enables an administrator to predict with better clarity when a client may fill up and identify the cause of any problems related to the performance of the data capture.



```
                              CLOG v1.2
Current status: Active.  >   0   <    Estimated input speed:  91.1 Mbps
                                           Capture goodness: 100.0%

    METRICS   PACKETS    CLIENTS    CONTROL    LOGFILE

  Detail of client #1:4

    Name:                bunyan1
    MAC address:         00:0D:93:76:E2:98
    Free space:         428.325 GB
    Total space:        467.485 GB
    Time til full:       62:07

  Client statistics:                        Commands:
    Data frames sent:    90732              << Return to client list
       es sent:          62976852                 Physically identify
       trols sent:       2                        Disconnect, reconnect
       trols received:   178   Timestamps se      Reset (delete packet l
```

Fig. 7. Screenshot of the client status report screen.

3.5 Future Enhancements

The CLog system is fully functional and has been used to create high quality network trace files for various research projects. Future enhancements of the system address usability issues of a completed network trace rather than the functionality of the core packet capture process.

Improvements to the management utility include adding the ability to control data which has been collected on the client systems. The current state of the system requires data to be removed manually to other storage areas for experimental use. In addition, in order to create a completely merged trace file a significant amount of storage space is required to consolidate the files. This limits the utilization of the client harddisks if they are required to keep free space for the purpose of creating a merged file.

Currently under development is a feature which allows CLog to control the entire trace without requiring trace files to be merged from each client system. Following the update of the timestamps for a single client, an index will be created for the trace files on the client. The entire trace for a single client consists of numerous individual files which sequentially numbered with a pre-determined size such as 100MB. The indexing system would enable CLog to quickly locate a position in the entire trace. The goal is to be able to request CLog to replay the network flow based on some parameter such as time-of-day or average bandwidth consumption. An experimental system could then be fed the output from CLog, replaying the feed from a live network.

4 Performance

The performance metrics we address here are directed at the ability of the CLog system to record network traces in terms of the percentage of dropped packets. Processor and memory usage cannot be ignored, but in this case extensive analysis is not necessary due to the minimal impact on the components. CPU usage ranged from 5% to 10% even when processing high bandwidth rates. Memory usage was limited as well and is only an issue when the line speed is greater than the CLog system can process. However, in such a case more memory may not necessarily solve the problem as a larger buffer does not resolve the issue of the buffer being filled faster than it can be emptied.

Performance evaluation was conducted using a large trace file collected from the gateway to the Internet of the University of Notre Dame. The file was replayed using tcpreplay [7] which is able to change the rate of replay to modify bandwidth rates. An actual network trace file is advantageous as the performance of the CLog system is not only affected by the bandwidth of the data, but also by the density in terms of packets per second. Thus, authentic network traffic may give more accurate results than is possible using an artificially generated trace file.

There are two areas of the system where packets may be dropped, in the server or in a client. If there are more packets available than the clients can record then the server is unable to empty its storage buffer quickly enough and packets are lost within the server. Because packets are dropped in this way to handle data overload, client systems are not actually affected by a dramatic increase in bandwidth consumption on the network that is being monitored. However, packet loss in the clients still does occur as show in Figure 8.

The packet loss rate in clients is a function of the number of new log files created. A single trace file was used as input for the experiments which are

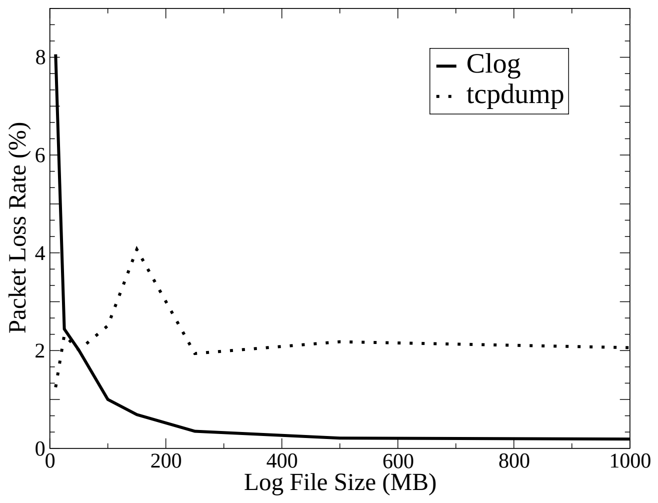


Fig. 8. Illustration of packet drop rate of a client system.

represented by this graph, meaning the size parameter of the log files was the determining factor on the number of files created. Identical experiments show that tcpdump is not affected in the same manner. Tcpdump was running on the same workstation used as the server for the CLog system utilizing an internal SCSI hard drive for storage. The fact that tcpdump file sizes did not influence the packet loss rate of the capture indicates the CLog clients may be optimized to reduce or eliminate this drawback. This issue will be addressed in future development of the system. For the remainder of the performance measurements log files of size 250MB were used

The overall performance of the system can be determined by measuring the packet loss on the server while varying the number of clients and the speed of the input data. Figure 9 shows the packet loss rate of the system with only a single client logging packets. The input speed is the average speed over the trace file replay. Peak bandwidth during the replay is approximately 50% higher than the average speed.

A single client is able to avoid packet loss at approximately an average bandwidth speed of 85Mb/s. At higher rates the storage buffer of the server reaches maximum capacity and packets are lost. As the average bandwidth rate increases, the system reaches a threshold where the buffer loses all effectiveness and extreme packet loss occurs. This can be seen in each case where a dramatic increase in packet loss occurs.

Figure 10 illustrates the same data associated with a varying number of clients as well as with the tcpdump packet logger. The original problem the CLog system was designed to overcome was the inability of a hard disk to keep up with the data served by a network monitoring feed. This figure clearly shows the effectiveness of the solution as the introduction of additional client systems

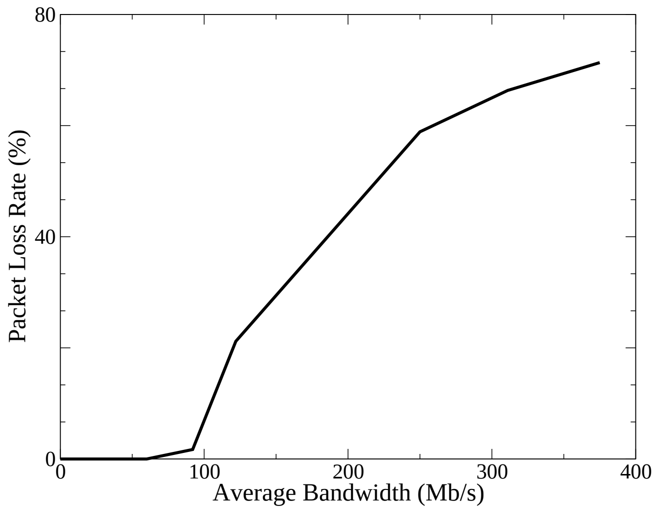


Fig. 9. Performance of a single client Cheap Logger implementation.

greatly improves over the single tcpdump system. It is important to implement a sufficient number of clients for the rate of the data to be recorded as the CLog system reaches a saturation point where packet loss increases dramatically and, in fact, performs much worse than tcpdump. When a sufficient number of clients are added, however, it is possible to record a much more complete trace of network data flow.

At an average data rate of approximately 375Mb/s the number of dropped packets for the five client system is non-zero, although somewhat negligible (0.6%). We note this because the packet loss is reported not as a loss within the CLog system, but as a result of the kernel dropping packets. We have not determined the exact cause of the packet loss, and therefore do not know if this is a result of hardware system capabilities or the CLog system. At an average data rate of approximately 470Mb/s tcpreplay seems to level off and is not able to replay our trace file at greater speeds. At this speed the CLog system still does not report any packet loss, but loss due to kernel packet drops was measured at 0.9%.

5 Summary

Capturing complete network trace files can be very difficult where speeds are high and budgets are low. The *Cheap Logger* (CLog) system is designed to eliminate this tradeoff by providing high quality network data capture without breaking the bank. The system utilizes COTS hardware and is easily scaled with the addition of individual client systems. Once a data capture is complete simple post processing steps reconstruct the original flow by merging files from

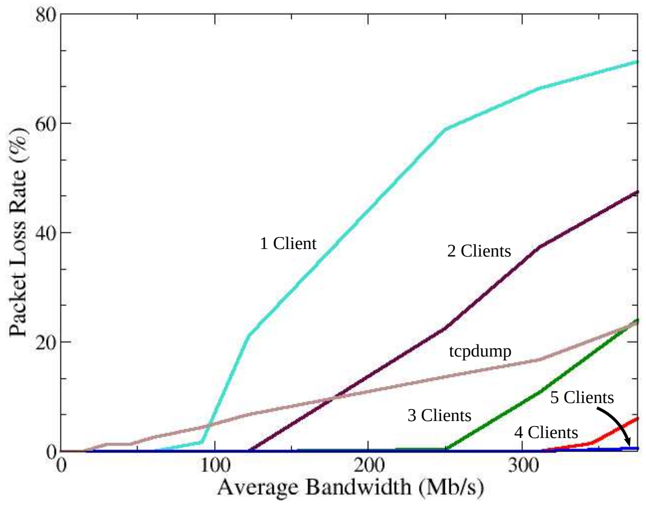


Fig. 10. Comparison of the Cheap Logger system with multiple clients to an implementation of tcpdump.

the distributed logging system. Timestamp synchronization is handled via an efficient timing reporting mechanism and postprocessing of the logged files.

The system significantly outperforms a single system running tcpdump, a very commonly used packet logging application. The current version of the CLog system can be obtained at http://gipse.cse.nd.edu/CLog. Future enhancements to the file creation process of the client application and improvements of data management capabilities are planned for the system.

References

1. Conduant corporation. http://www.conduant.com/.
2. Ethereal. http://www.ethereal.com.
3. Libnet c library. http://www.packetfactory.net/libnet/.
4. Libpcap c library. http://www.tcpdump.org.
5. Network flight recorder. http://www.nfr.net/.
6. Tcpdump. http://www.tcpdump.org.
7. Tcpreplay project. http://tcpreplay.sourceforge.net/.
8. H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of USENIX Security Symposium*, pages 271–286, San Diego, CA, August 2004.
9. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
10. S. McCanne and S. Floyd. ns Network Simulator. http://www.isi.edu/nsnam/ns/.
11. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of USENIX OSDI*, San Francisco, CA, December 2004.