CLog: Low Cost Gigabit Full Packet Logging

Chad D. Mano, Jeff Smith, Bill Bordogna, Andrew Matta, Dan Dugovic, Aaron Striegel
Department of Computer Science and Engineering, University of Notre Dame, USA
Email: {chad.mano}@usu.edu, {jsmith30,wbordogn,amatta,dduguovic,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 that end, we developed the *Cheap Logger* (CLog) system which utilizes inexpensive COTS hardware to create 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.**

I. Introduction

An important aspect of various areas of computer network research is the ability to perform analysis in a large-scale network environment. However, most network administrators are 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] and *OpNet* are useful for simulations where basic evaluation is the key issue being addressed, but fall short in the ability to create a highly accurate representation of actual data flow within a large scale network. Although work has been conducted on scaling *ns-2* to larger scales [12], internal data traffic payloads are not available. An accurate representation of traffic is essential where packet payload analysis is needed such as in intrusion detection, virus and worm detection, and other areas [8], [11].

Although software router solutions such as the Click router [13] offer the ability to create a limited network either locally or via Planetlab, the data itself is limited to synthetic or previously traced data, the capture of which is the focus of the paper. 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 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 II briefly presents the motivation and background for the project. Section III details the architecture of the system including communication protocols developed for management of the system. Section IV analyzes the performance capabilities of the system. Finally, Section V summarizes the work.

II. 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. Experiments for group range from similarity analyses of packet payloads for bandwidth conservation to long term monitoring of botnet characteristics. In each case, the packet header and the packet payload are necessary to record in order to construct an accurate assessment of traffic.

The source of the live data is provided through 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 150 Mb/s utilization in each direction with continued patterns of growth.

In the past, a single HP zx2000 Itanium2 high performance workstation was used as the packet logging system. Even this relatively fast desktop system (4 GB memory, 15k SCSI disk) would drop a significant number of packets leaving an incomplete trace file of network traffic. Although the workstation was capable of processing each packet via *libpcap*, the bottleneck, it was discovered, proved to be the disk writes. Despite accommodations for block writes, the workstation could simply not keep up with amount and speed of the traffic coming from the tap. While RAID systems could offer increased throughput, similar bottlenecks would occur on all but the most high performance systems.

This paper is based on “High Speed Packet Logging on a Budget,” by C. Mano, J. Smith, B. Bordogna, and A. Striegel, which appeared in the Proceedings of the IFIP Networking, Coimbra, Portugal, May 2006.
This research was supported in part by the National Science Foundation through the grant CNS03-47392.

© 2006 ACADEMY PUBLISHER

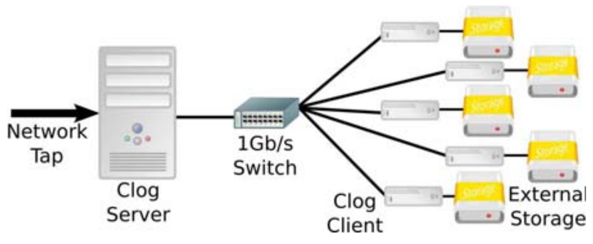


Figure 1. Illustration of the Cheap Logger system.

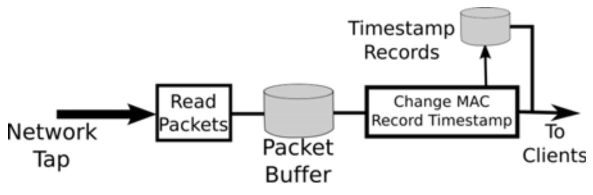


Figure 2. Diagram of the data flow within the Cheap Logger server.

To that end, 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 were 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 and additional storage may be dynamically added to the system as network capacity increases.

III. Architecture

The physical architecture of the CLog system is based on the 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. The server itself need only be capable of processing packets for simple store/forward operations. Clients need simply be capable of access to large amounts of inexpensive storage.

For our internal setup, a Sun dual Opteron 244 workstation with 1GB or RAM was utilized as the server. As space and mobility constraints required all components to fit onto a small cart, the client and storage mechanisms needed to remain small. Therefore, Apple 1.25 GHz Mac mini (PowerPC-based) systems running Darwin Kernel Version 1.9.0 (Mac OS X) 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. With the advent of the new Intel-based Mac mini machines and their inclusion of USB 2.0, USB 2.0 drives would also suffice. Performance results are presented later in the paper for both the PowerPC and Intel-based Mac mini systems.

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.

A. 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 utilized the onboard 1Gb/s Ethernet port for remote management and installed an Intel Pro/1000 MT dual port 1Gb/s NIC for the required input/output ports. In general, we found a separate high performance adapter (not included on the motherboard) was often necessary to scale to higher speeds due to features such as interrupt chaining.

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. As traffic is primarily unidirectional on the Gigabit switch (CLog server to clients), most COTS switches will suffice for distribution to the clients.

The processing of the network tap data at the server is kept minimal to allow the server to 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 ring buffer (default of 10 MB). A separate thread removes packets from the head point of the buffer and overwrites the destination MAC address in the Ethernet header of the packet. The packet is then transmitted on the o-port and is forwarded on to one of the client systems via the switch.

While it is possible to implement a load balancing scheme to enhance the overall effectiveness of the distributed writing system, the complexity of such a scheme offers little benefit. As the client systems are simply logging and not analyzing packets, load across the systems is already well distributed. Thus, 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.

B. Client

The clients perform the “physical labor” of the system of writing the network data to disk. A custom packet logger application, similar to tcpdump, was developed using the Libpcap [4] library. In short, the logger contains only mechanisms to log packets to disk and maintain status communications with the server. The packet logging process is discussed here with the server communication for management purposes to be presented later in this section.

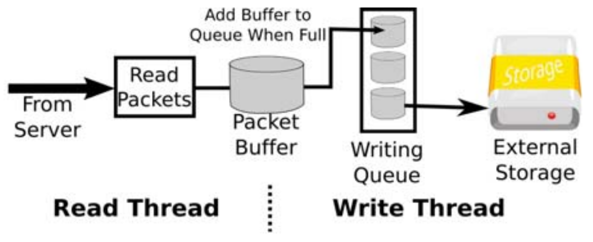


Figure 3. Illustration of the Cheap Logger client system.

Once a client is connected to a server, the client 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 block-wise packet buffer. A parameter setting determines the maximum size of the buffer based on the number of packets contained in the buffer. When the buffer reaches the its limit, an alternate thread is used to take the data in the packet buffer and write 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.

In addition to the other threads, an auxiliary thread is created to simply manage the creation and closing of existing data files. As the file open operation can be a relatively costly operation in the context of a continuous stream of packets, the sole purpose of the auxiliary thread is to maintain a bank of files such that several new file handles are always available to the central writing thread. Thus, it is possible to have relatively small file sizes without a critical loss of performance.

The log files are stored in standard *tcpdump* format. This allows the final trace files to be analyzed using existing applications such as such as *tcpdump* or *ethereal* [2] and others. A critical portion of the logging format is the 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 essential to maintain precise timing data for the entire system. This property is maintained by the timestamp synchronization process.

1) Timestamp Synchronization: Timestamps are created when a new packet arrives at the 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.

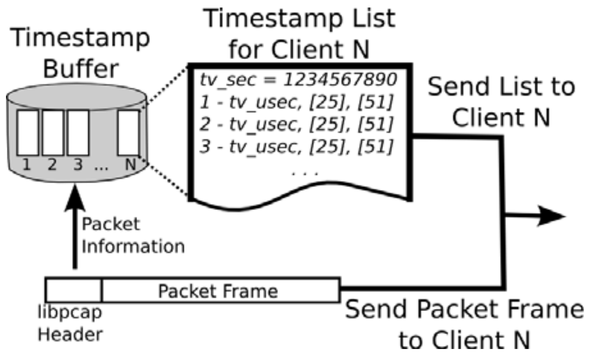


Figure 4. Illustration of the timestamp system.

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. 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*).

Once the timestamp is determined via libpcap, a log entry is created holding the timestamp, a rolling 16 bit ID for the packet for re-assembly and loss identification, and the lower 2 bytes of the source IP. The lower 2 bytes of the source IP are stamped with ID and the packet is forwarded to the client. A buffer of log entries is kept until a fixed number upon which the buffer is dispatched via a special packet to the next round robin client and recorded on the disk of the server. Optionally, server-side recording may be disabled for storage or performance purposes.

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

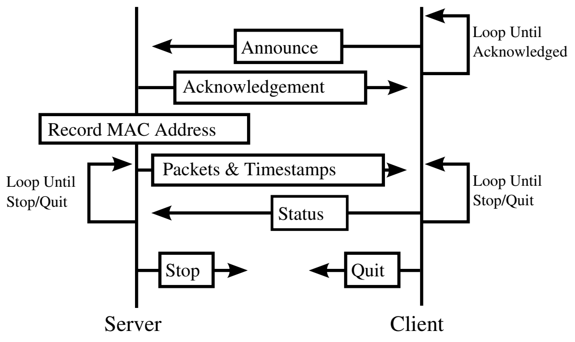


Figure 5. Illustration of the communication protocol between clients and the server.

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.

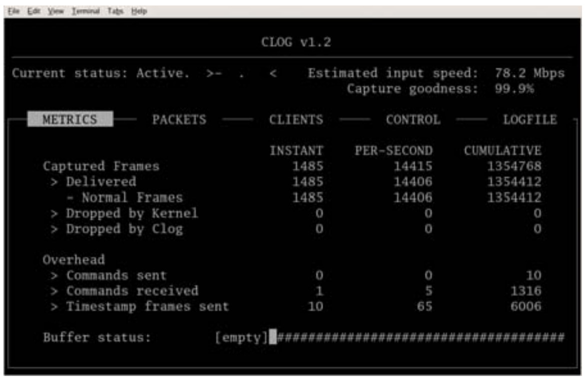
Sorting and re-assembly of packets is provided by a utility program that is executed on the server. The server contacts each client for its repository of data and re-assembles/writes back the new files to the clients. The old ’marked’ files are deleted and replaced with similarly sized log files in the correct order and possessing the correct timestamp. Reassembly is done off-line due to the complexity and bandwidth associated with moving data files between the clients and server. The reassembly utility also possesses mechanisms for extracting specific time periods/packet ranges from the array of data files.

C. 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.

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 ID 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 provided in the next section.



```
                              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]█#####################################
```

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

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.

D. 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 the illustration of a few management screens provides a general idea of the reporting system. Further information is available via the CLog website[1].

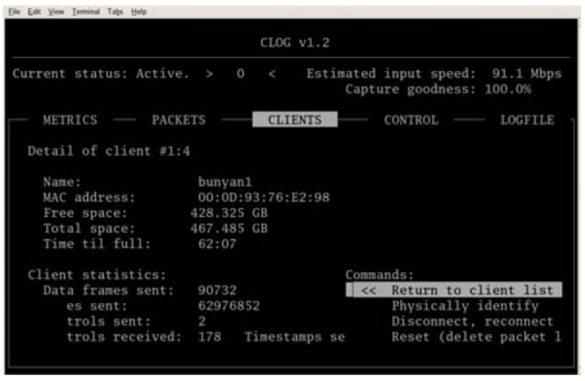
Figure 6 shows the overall statistics of the current capture. The most important feature is the detail of the number of packets dropped, categorized by packets dropped by the system kernel and those dropped by CLog. These statistics enable an administrator to quickly identify the existence of a problem 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. While disks cannot be hot swapped in the sense that a client must be leave/rejoin the server, new disks can easily be added/parsed while the logging system is running.

In addition, a web monitoring tool is available to observe the raw statistics of the CLog system. While the web system does not provide for interactions with the server, it allows the CLog server to output its current status to a separate web server.

As noted earlier, the management utility includes the ability to control data which has been collected on the

[1]http://gipse.cse.nd.edu/CLog



```
                              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
```

Figure 7. Screenshot of the client status report screen.

TABLE I.
LOSS RATES FOR VARIOUS RAW BANDWIDTH RATES (EC = EXCEEDS CAPACITY)

Relative Speed	Power PC Mac Mini	Intel Dual Core Mac Mini	Sun Dual Opteron Workstation
10 Mb/s	0.99%	0.73%	0.00%
20 Mb/s	0.97%	0.63%	0.02%
40 Mb/s	2.29%	0.87%	0.01%
60 Mb/s	6.56%	0.71%	0.05%
80 Mb/s	7.26%	0.74%	0.04%
120 Mb/s	EC	1.16%	0.51%
160 Mb/s	EC	1.34%	0.50%
200 Mb/s	EC	EC	0.82%
240 Mb/s	EC	EC	2.56%
280 Mb/s	EC	EC	3.59%
320 Mb/s	EC	EC	EC

client systems. CLog allows the trace to be reassembled/stored without requiring a massive central repository. As each client may possess 500 GB to multiple terabytes, the use of a central repository with enough simultaneous storage is simply not feasible. CLog uses limited amounts of space in a temporary directory on the server coupled with intelligent network transfers to the clients to provide a simple re-assembly/ordering step. Additional features under development include the ability to allow 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.

IV. 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 evaluations were conducted in two scenarios, a simple scenario with synthetic traffic to test raw tcpdump performance and through scenarios playing back previously captured (albeit slightly lossy) traffic at accelerated rates. The key performance metric was the loss rate recorded for the packets with the target loss rate at less than 1% over the course of the trace.

A. Synthetic Traffic

In the first test environment, UDP packets were sent at the various logging devices to measure overall performance. A custom-built UDP mechanism directed packets at the client that recorded the entire packet and data payload to its local disk. ICMP error messages were disabled to avoid CPU overhead and the test network was isolated to avoid background traffic from the campus network.

Each client had its performance using the most recent version of tcpdump filtering traffic for only inbound UDP traffic (src host 192.168.0.10) along with a snaplen (maximum packet capture size) of 1600 bytes and no limit to the overall file size. Traffic was provided via CBR UDP traffic possessing a data payload of 1400 bytes and a total recorded packet size of 1442 bytes (all headers). Loss measurements were collected using the reporting mechanism of tcpdump and validated by the size of the final packet capture.

Three systems were compared in the tests, the PowerPC-based Mac mini (1.25 GHz, single processor) used as a client in the later tests, a new Intel-based Mac mini (1.66 GHz, dual-core) as well as the workstation operating as the CLog server (Sun Opteron 244 dual CPU workstation). Notable differences between the two versions of the Mac mini include dual-core (Intel) versus single CPU (PowerPC) and the inclusion of a Gigabit adapter (Intel) versus Fast Ethernet (PowerPC). The Itanium2 workstation used for previous logging capacities was not included due to its rarity in the research community.

From Table I, several interesting observations can be drawn. First, the overhead of Mac OS X versus Linux is significant enough to introduce almost 1% loss even at low traffic levels. Despite possessing a second core, the Intel variation still suffers nearly the same losses until the PowerPC variation simply becomes overwhelmed and rapidly increases its loss. As will be shown later, our optimized client logging program significantly reduces this loss. An increase in the priority level of tcpdump via *renice* improves performance slightly but does not reduce the loss to the near zero levels observed on the Opteron workstation.

The EC notation is used to denote when the traffic exceeds the capacity of tcpdump to even record errors. While the PowerPC Mac mini is still able to reliably record losses in plain vanilla tcpdump near the capacity of Fast Ethernet, the Intel-based Mac mini tops out considerably higher. The Opteron workstation possesses

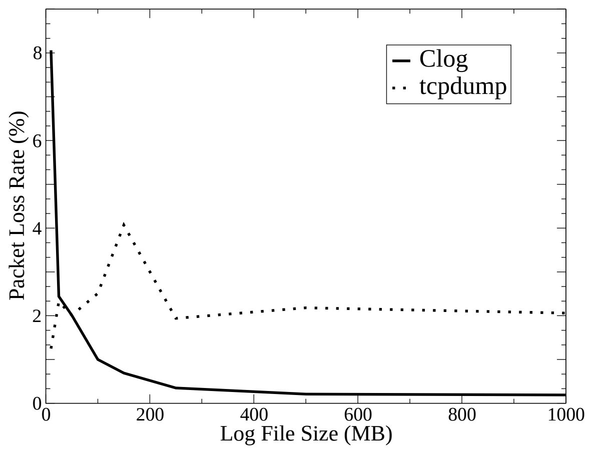


Figure 8. Illustration of packet drop rate of a client system.

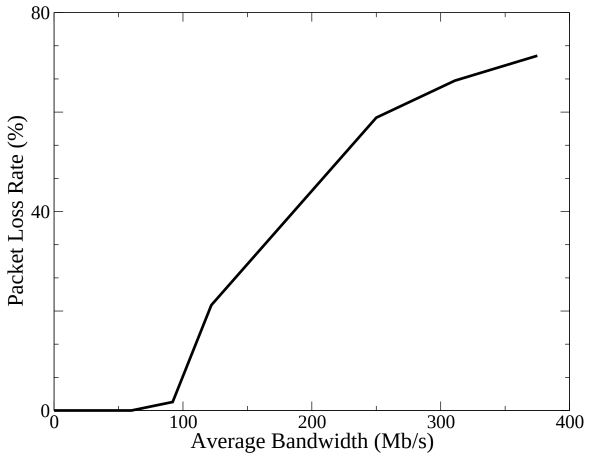


Figure 9. Performance of a single client Cheap Logger implementation.

the fastest raw logging speed due to several factors: its usage of Linux, the incorporation of a server-grade network adapter, and a much faster disk (SCSI). However, the workstation could at best support an extremely limited recording time at an acceptable loss rate (200 Mb/s or less, one direction, 100 Mb/s bi-directional) even if large capacity (750 GB+) drives were utilized.

B. Replayed Trace Traffic

Performance evaluation was also conducted using a large trace file collected from the gateway to the Internet of the University of Notre Dame. The files were replayed using tcpreplay [7] which is able to change the rate of replay to modify bandwidth rates. The 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.

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 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 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

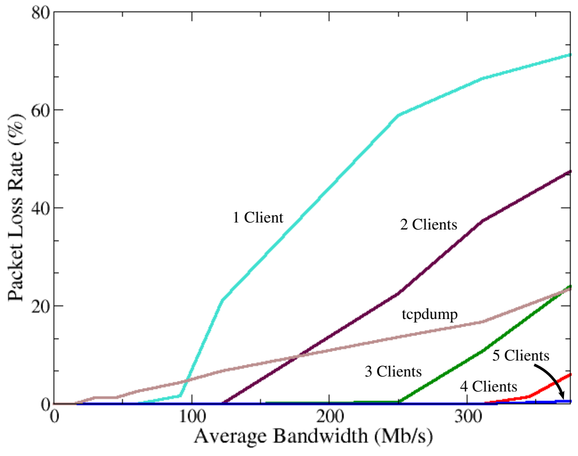


Figure 10. Comparison of the Cheap Logger system with multiple clients to an implementation of tcpdump.

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%.

V. 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 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.

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.
[12] G. F. Riley, M. H. Ammar, R. M. Fujimoto, K. Perumalla, and D. Xu, Distributed Network Simulations using the Dynamic Simulation Backplane *International Conference on Distributed Computing Systems 2001 (ICDCS'01)*
[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek The Click modular router. *ACM Transactions on Computer Systems* 18(3), August 2000, pp. 263-297.

Chad Mano is currently an assistant professor in the Department of Computer Science at Utah State University. He received his Ph.D. in the summer of 2006 in Computer Science at the University of Notre Dame. His research interests include computer security, focusing on rogue wireless detection, network intrusion detection, secure communication protocols, and computer security education.

Jeff Smith, Bill Bordogna, Andrew Matta, and Dan Dugovic are all undergraduate students at the University of Notre Dame in the Department of Computer Science and Engineering. All were participants in the National Science Foundation (NSF) Research Experience for Undergraduate (REU) working on components of the NSF CAREER grant of Dr. Striegel.

Aaron Striegel is currently an assistant professor in the Department of Computer Science and Engineering at the University of Notre Dame. He received his Ph.D. in December 2002 in Computer Engineering at Iowa State University under the direction of Dr. G. Manimaran. His research interests include networking (bandwidth conservation, QoS), computer security, grid computing, and real-time systems. During his tenure as a student at Iowa State, he worked for various companies in research and development that included Sun Microsystems, Architecture Technology Corporation, and Emerson Process. He has received research and equipment funding from NSF, DARPA, Sun Microsystems, Hewlett Packard, Architecture Technology Corporation, and Intel. Dr. Striegel was the recipient of an NSF CAREER award in 2004.