Proceedings of the 10th International Conference on
Computer Communications and Networks (IC3N'01), October, 2001.

MAGNeT: Monitor for Application-Generated Network Traffic

Wu-chun Feng, Jeffrey R. Hay, Mark K. Gardner
{feng, jrhay, mkg}@lanl.gov
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract— **Over the last decade, network practitioners have focused on monitoring, measuring, and characterizing traffic in the network to gain insight into building critical network components (from the protocol stack to routers and switches to network interface cards). Recent research shows that additional insight can be obtained by monitoring traffic at the application level (i.e., before application-sent traffic is modulated by the protocol stack) rather than in the network (i.e., after it is modulated by the protocol stack).**

Consequently, this paper describes a Monitor for Application-Generated Network Traffic (MAGNeT) that captures traffic generated by the application rather than traffic in the network. MAGNeT consists of application programs as well as modifications to the standard Linux kernel. Together, these tools provide the capability of monitoring an application's network behavior and protocol state information in production systems. The use of MAGNeT will enable the research community to construct a library of real traces of application-generated traffic from which researchers can more realistically test network protocol designs and theory. MAGNeT can also be used to verify the correct operation of protocol enhancements and to troubleshoot and tune protocol implementations.

I. Motivation

Although monitoring [1–6] and characterizing [7–10] traffic in the network provides insight into building critical network components (e.g., ideal buffer sizes in routers), recent research [11–13] shows that additional insight can be obtained by monitoring and measuring traffic at the application level (i.e., *before* it is modulated by the protocol stack) rather than in the network (i.e., *after* it is modulated by the protocol stack). For example, knowing application traffic patterns can provide insight into the design of a better protocol stack.

A. Background

Network researchers often use traffic libraries such as `tcplib` [14], network traces such as those at [15,16], or network models such as those found in [9] to drive their network experiments, particularly to test the performance of network-protocol enhancements. However, such libraries, traces, and models are based on measurements made by `tcpdump` [1] (or similar tools like PingER [2], NLANR Network Analysis Infrastructure [4], NIMI [5], CoralReef [6]), meaning that the traffic an application sends on the network is captured only *after* having passed through TCP (or more generally, any protocol stack) and into the network. That is, the tools capture traffic *on the wire* (or *in the network*) rather than at the application level. Thus, the above tools cannot provide any protocol-independent insight into the actual traffic patterns of an application.

This work was supported by the U.S. Dept. of Energy's Laboratory-Directed Research & Development Program and the Los Alamos Computer Science Institute through Los Alamos National Laboratory contract W-7405-ENG-36. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DOE, Los Alamos National Laboratory, or the Los Alamos Computer Science Institute. This paper is LA-UR 01-5063.

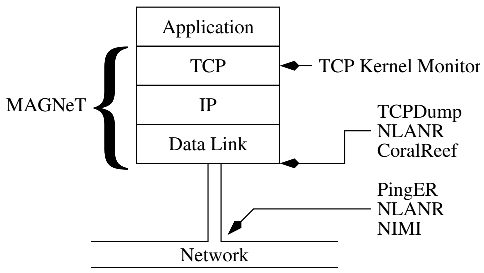

Fig. 1. Monitoring Points of Various Tools

So, researchers have not been testing the performance of network protocols using *real*, application-generated traffic traces; rather, they have used once-modulated (by the protocol stack) traffic traces as input, which are subsequently modulated a second time during the testing of the protocol. If the differences between application-generated traces and network-captured traces are negligible, such a simplification is acceptable. However, as we will show in this paper, the differences in the traces are substantial, indicating that the protocol stack adversely modulates the application-generated traffic patterns. This observation may invalidate the empirical results gathered from the performance evaluation of network protocols over the last decade because many researchers used network-captured traces rather than application-generated traces to drive their experiments.

What network researchers need are traces of real, application-generated traffic. Therefore, this paper describes a Monitor for Application-Generated Network Traffic (MAGNeT) that captures traffic generated by the application rather than in the network.

B. Related Work

As shown in Fig. 1, MAGNeT differs from `tcpdump`-like measurement tools in that it can monitor traffic at the application level, (i.e., *before* it traverses the protocol stack), and throughout the entire protocol stack, as well as traffic entering and leaving the network. The only other measurement tool that makes similar measurements is the TCP kernel monitor from Pittsburgh Supercomputing Center [17].

MAGNeT differs from the TCP kernel monitor in at least three ways. First, MAGNeT, although described in this paper as only a monitor for application-generated traffic (i.e., the interface between the application layer and TCP layer), can be used anywhere in the protocol stack. Second, MAGNeT monitors a superset of the data that the TCP kernel monitor does. Third, MAGNeT is implemented on the Linux operating system whereas PSC's TCP kernel monitor works on NetBSD.

II. Software Architecture

Developing MAGNeT presents two design challenges. First is the issue of accurate time measurement. MAGNeT uses the CPU cycle counter available in modern microprocessors to record timestamps with cycle-level granularity. This functionality relies on code in the architecture-specific Linux code base but should be present on most Linux platforms.

The second, and more difficult, challenge requires that MAGNeT run on machines in a production environment in order to obtain realistic application-traffic traces. Hence, MAGNeT must incur minimal overhead so that the end user is not impacted and so that the application-generated traffic stream is not adversely perturbed. This latter consideration requires that any processing or filtering of the captured data be performed off-line; reducing or filtering the data in real-time not only adds an intolerable performance overhead but also adversely perturbs the application-generated traffic stream.

We address these design issues in our MAGNeT software distribution[1], which consists of several user-application programs as well as modifications to the Linux kernel. The primary functionality of MAGNeT is contained in a patch to the Linux kernel. The patch creates a circular buffer in kernel memory and places calls throughout the networking stack to record appropriate information as data traverses the stack. A user-space program periodically empties this kernel buffer, saving the binary data to the disk. For post-processing the data, a set of data-analysis tools translates the binary data into machine- and human-readable form. Finally, a library of scripts to enable automated data collection from a set of hosts completes the MAGNeT package.

Fig. 2 illustrates the operation and dataflow of MAGNeT at a high level. Unmodified applications (that run on the test system) periodically make `send()` and `recv()` system calls to send and receive network traffic. These calls eventually make use of TCP, IP, or other network protocols in the kernel to transfer data on the network. For systems running MAGNeT, each time a `send()`, `recv()`, or network protocol call is made, an accompanying call is made to `magnet-add()`. This procedure saves relevant data to a circular buffer in kernel space, which is then saved to disk by the user-level application program `magnet-read`. The details of each of these steps is discussed below.

A. MAGNeT in Kernel Space

The MAGNeT kernel patch adds several functions to the Linux 2.4 kernel. The function `magnet_add()` adds a data point to a circular buffer that is guaranteed to always be in kernel memory. This function can be called virtually anywhere in the protocol stack; it is optimized so that each instrumentation call uses as few resources as possible. In addition, a new `/proc` item is added to the file space at `/proc/net/magnet`. This file may be read by any user to determine the current state and parameters of the MAGNeT kernel code.

[1]The MAGNeT software distribution is currently undergoing alpha testing at LANL. A beta prototype will soon be publicly available from `http://www.lanl.gov/radiant`.

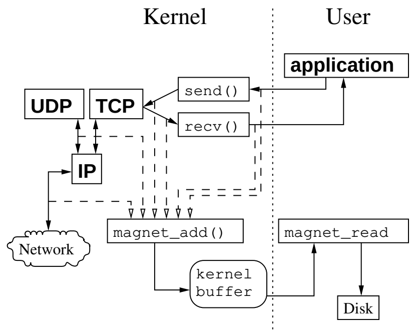

Fig. 2. Overview of MAGNeT Operation

```
struct magnet_data {
  void *sockid;
  unsigned long long timestamp;
  unsigned int event;
  int size;
  union magnet_ext_data data;
}; /* struct magnet data */
```

Fig. 3. The MAGNeT Instrumentation Record

A.1 Instrumentation Record

Fig. 3 shows the MAGNeT instrumentation record, the data structure that `magnet_add()` adds to the kernel buffer at each instrumentation point. `sockid` is a unique identifier for each connection stream, thus giving MAGNeT a way to separate data traces into individual streams while protecting the privacy of the application and user. The `timestamp` field is a CPU cycle counter that is used to synchronize MAGNeT events. Valid values for the `event` field (e.g., `MAGNET_IP_SEND`) indicate the type of event a particular record refers to. `size` contains the number of bytes transferred during a specific event. The `data` field (a optional field selected at kernel compile time) is a union of various structures in which information specific to particular protocols can be stored. This field provides a mechanism for MAGNeT to record protocol state information along with event transitions.

A.2 Instrumented Events

The MAGNeT kernel patch instruments the general socket-handling code, the TCP layer, and the IP layer. Other protocols can be easily instrumented by adding new MAGNeT `event` codes and placing calls to `magnet_add()` at appropriate places in the protocol stack.

There exists a possibility that the space in the fixed-sized, circular kernel buffer may be exhausted before the user-space program is able to read the records it contains. However, MAGNeT will not overwrite any recorded data. This is accomplished by using the `timestamp` field of the instrumentation record as a synchronization flag between MAGNeT user- and kernel- processes. Before writing to a slot in the circular buffer, the MAGNeT kernel code checks the value of the `timestamp` field for that slot. A value other than zero indicates that the slot has not yet been copied to user space and that the kernel buffer is full. In this case, the kernel code increments a count of the number of instrumentation records that could not be written due to the buffer being full. After the user-level application reads a record from the buffer, it writes a zero to the `timestamp` field to signal to the kernel that the slot is available. Once buffer space becomes available, the kernel writes a special instrumentation record with an event type of `MAGNET_LOST` and with the `size` field set to the number of instrumentation records dropped.

Our experience to date indicates that while dropped instrumentation records are possible, they rarely occur during the monitoring of actual users. (See Section III-B.3 for details.)

B. MAGNeT in User Space

The user-level interface to MAGNeT consists of three application programs (`magnet-read`, `magnet-parse`, and `mkmagnet`), a special device file to facilitate kernel-user communication, and a collection of automating scripts. The application program `magnet-read` saves data from the kernel's buffer to a disk file and `magnet-parse` translates the saved data into an understandable format. `mkmagnet` is a small utility program to create the files that `magnet-read` requires to operate. The scripts included with the MAGNeT distribution allow the operation of MAGNeT to be fully automated and transparent to the end user.

B.1 `magnet-read`, `mkmagnet`, `magnet-parse`

To minimize the potential for record loss when MAGNeT reads the instrumentation records from the circular buffer in the kernel and writes them to disk, we map the circular buffer from kernel space into user space via a special "shared memory" device file. With this mapping in place, no additional kernel code is executed; the application program may simply read the shared memory and write it to disk. This transparency is accomplished by the program `magnet-read`, which copies the records by linking the shared memory region to a pre-existing file via the kernel's memory-mapped I/O system. The `mkmagnet` application creates and initializes the aforementioned file prior to being mapped into memory. The last tool, `magnet-parse`, reads data collected by `magnet-read` and dumps a tab-delimited ASCII table of the collected data for further processing.

B.2 Automating Scripts

Two shell scripts in the MAGNeT distribution, `magnet.cron` and `magnet.copy`, provide examples of how to create an automated, transparent, application-monitoring environment on a campus network. `magnet.cron` is the overall MAGNeT management script, which should be executed upon initial boot, and at some interval (e.g., daily) during system operation. When run, it ensures that the proper device file exists and that a file has been created on disk with `mkmagnet`. It then starts `magnet-read`.

C. MAGNeT Timestamps

To ensure the greatest accuracy possible, MAGNeT uses the cycle counter available on contemporary microprocessors as the source of its timestamps. MAGNeT obtains this information via the kernel's `getcyclecounter()` function, which keeps the MAGNeT code hardware-independent. Given the speed of the processor, the difference between two cycle counts can be converted to elapsed time. Thus, the first record created by MAGNeT is of type `MAGNET_SYSINFO` and the size field of this record contains the processor clock speed in kHz.

TABLE I

MAGNeT vs. `tcpdump`

Configuration	Throughput (Kb/s)	Send CPU (%)	Receive CPU (%)
Linux 2.4.3	94.1 ± 0.0	15.2 ± 0.1	33.5 ± 0.1
MAGNeT	94.1 ± 0.1	16.9 ± 0.2	33.5 ± 0.1
`magnet-read`/rcv	90.8 ± 0.8	20.7 ± 0.3	34.4 ± 1.0
`magnet-read`/snd	90.7 ± 0.9	23.7 ± 1.7	32.4 ± 0.4
`tcpdump`/rcv	89.4 ± 1.5	18.0 ± 0.4	59.8 ± 0.9
`tcpdump`/snd	89.4 ± 0.8	45.0 ± 0.6	31.9 ± 0.3

(a) 100Mbs (Fast Ethernet)

Configuration	Throughput (Kb/s)	Send CPU (%)	Receive CPU (%)
Linux 2.4.3	459.5 ± 1.6	61.0 ± 0.3	82.4 ± 0.2
MAGNeT	452.5 ± 1.8	63.0 ± 0.4	82.6 ± 0.3
`magnet-read`/rcv	444.3 ± 1.7	62.4 ± 0.3	82.0 ± 0.3
`magnet-read`/snd	440.2 ± 2.1	63.1 ± 0.5	81.1 ± 0.4
`tcpdump`/rcv	290.7 ± 15.6	36.1 ± 2.0	91.5 ± 0.5
`tcpdump`/snd	343.2 ± 18.7	93.2 ± 0.5	64.1 ± 3.3

(b) 1000Mbs (Gigabit Ethernet)

III. MAGNeT Performance Analysis

In this section, we demonstrate MAGNeT's ability to record application and network-stack events without adversely perturbing the traffic stream or application behavior. We also use MAGNeT-collected data to show significant differences between an application's network demands and the actual traffic delivered to the network.

A. Experimental Method

As an indication of how much MAGNeT perturbs application-generated network traffic, we measure the maximum data rate between a sender and receiver as well as the CPU utilization. In addition, we measure the overhead of running `tcpdump` as a point of comparison. In total, we run six different configurations on 100-Mbps and Gigabit Ethernet networks, as shown in Table I.

We conduct our tests between two identical dual 400MHz Pentium IIs with 100Mbps or 1000Mbps Ethernet card (connected via an Extreme Networks Summit 7i switch) and configure MAGNeT to record application `send()` and `recv()` socket calls as well as TCP and IP events. MAGNeT uses the default 256KB kernel buffer to store event records.

To generate the workload, we run `netperf` [18] on the sender,[2] transmitting data as fast as possible. We minimize the amount of interference in our measurements by eliminating all other network traffic and minimizing the number of processes running on the test machines to `netperf` and a few essential Linux services.

[2]The command used was "`netperf -P 0 -c {local CPU index} -C {remote CPU index} -H {hostname}`"

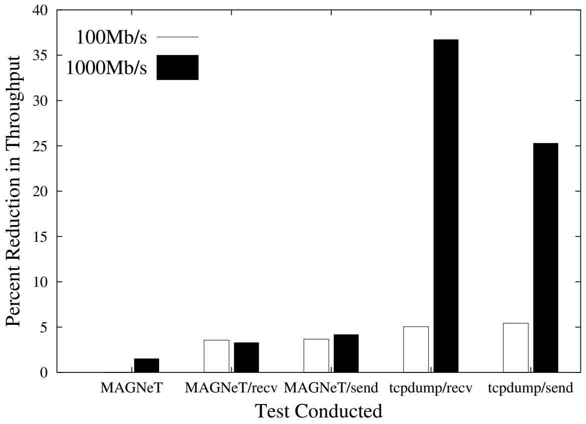

Fig. 4. Percent Reduction in Network Throughput

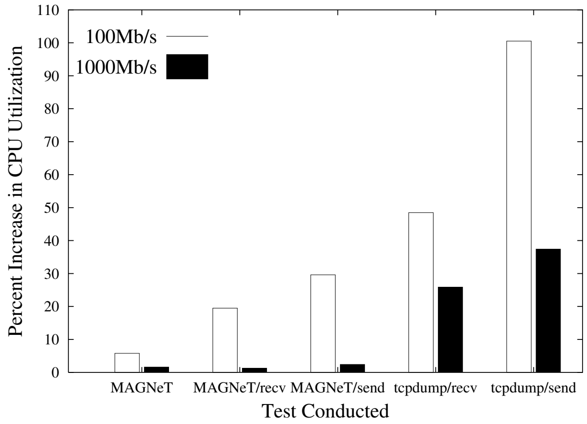

Fig. 5. Average Percent Increase in CPU Utilization

B. Performance

While no monitoring system can be completely transparent to the workload being monitored, MAGNeT is designed to have minimal impact on overall network throughput. Table I shows how MAGNeT performs in comparison to `tcpdump`. Along with the mean, the width of the 95% confidence interval is given. Figures 4 and 5 present this data graphically.

By default (and as used in our experiments), `tcpdump` stores the first 68 bytes of every packet. Our configuration of MAGNeT, on the other hand, stores 96 bytes for each packet.[3]

B.1 Network Throughput

The kernel-resident portion of MAGNeT is always executing, regardless of whether information is being saved to disk or not. The first pair of bars in Fig. 4, labeled "MAGNeT," shows only a small penalty when no data is being saved to disk. The next two pairs of bars show that MAGNeT incurs less than a 5% reduction in network throughput when `magnet-read` runs on either the receiver or sender. Furthermore, the penalty is nearly constant regardless of network speed. In contrast, while `tcpdump` incurs roughly the same penalty as MAGNeT over 100Mbps networks, the penalty increases to 25%-35% of total throughput at 1000Mbps. Thus, MAGNeT scales better with increasing link speeds than `tcpdump`.

B.2 CPU Utilization

Next, we compare MAGNeT's and `tcpdump`'s CPU utilization as reported by `netperf`.[4] Each bar in Fig. 5 reflects the percentage increase in CPU utilization averaged over both the sender and the receiver. In this case, MAGNeT also performs better than `tcpdump`, which is not surprising since `tcpdump` makes system calls from user space (thus incurring a context switch) for every packet while MAGNeT executes primarily within the kernel. What may be surprising is that both MAGNeT's and `tcpdump`'s overhead appears to decrease when run on the faster network. This is due to interrupt coalescing, i.e., the network interface cards accumulate several incoming packets before interrupting the CPU. Thus, the average overhead of servicing interrupts is greatly reduced. Had interrupt coalescing been disabled, the average CPU utilization for both MAGNeT and `tcpdump` would have increased.

B.3 Event Loss

During our testing, analysis of the MAGNeT-collected data revealed that MAGNeT occasionally loses events at high network utilization. For the 100Mbps trials, MAGNeT lost less than than 3.2% of the total events; for the 1000Mbps tests, the loss rate approached 15%. These losses are due to the 256KB circular buffer in the kernel filling before `magnet-read` is able to drain it.

Two methods exist for reducing the loss rate: (1) increasing the kernel buffer size and/or (2) reducing the time `magnet-read` waits before draining the kernel buffer. Fig. 6 shows how changing the kernel buffer size or `magnet-read` delay affects event-loss rate on a 100Mbps network. (Note: All other tests discussed in this paper used MAGNeT's default values for these two parameters – 256KB kernel buffer with `magnet-read`'s automatically calculated delay time.)

Increasing the size of the kernel buffer dramatically reduces MAGNeT's potential for event loss, down to virtually no lost events under any network load with a 1MB buffer. However, because this buffer is pinned in memory, a large buffer also reduces the amount of physical memory available to the kernel and applications. We chose 256KB to be the default buffer size to reduce both CPU utilization and physical memory consumed (i.e., to reduce the invasiveness of the monitor).

Another method for reducing event loss entails adjusting the amount of time `magnet-read` sleeps before draining the kernel buffer. However, shorter sleep times create more work (in terms of CPU usage, and possibly, disk write activity), and thus may interfere with the system's normal use in a production environment.

The default sleep-time is computed as the average amount of time it takes to fill the kernel buffer on a 100Mbps network. This heuristic was chosen because it provides relatively low event-loss rates without significantly impacting the user.

[3]Although MAGNeT's record size is only 24 bytes *per event*, our configuration of MAGNeT instruments the events at each protocol layer in the stack, resulting in four events per packet (or $24 \times 4 = 96$ bytes per packet).

[4]This version of `netperf` measures CPU utilization via an "idle" loop in a very low-priority process (which theoretically should only run when the system is otherwise idle).

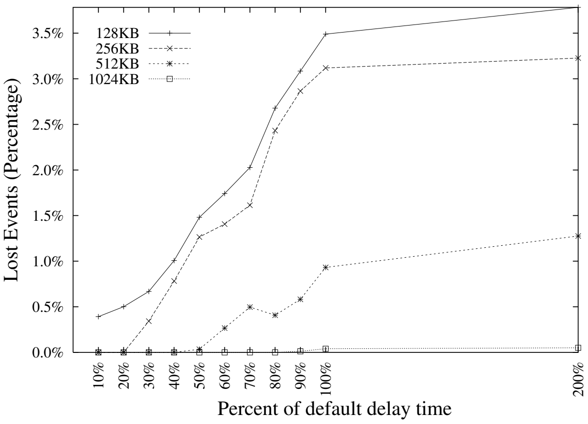

Fig. 6. MAGNeT's Event-Loss Rate, 100Mbs Ethernet

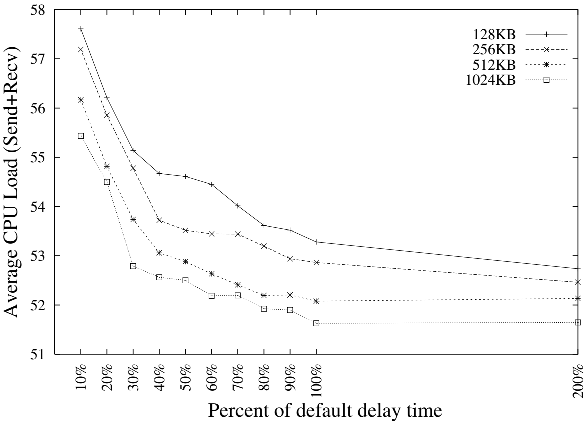

Fig. 7. MAGNeT's Average CPU Utilization, 100Mbs Ethernet

Command-line options are provided in `magnet-read` to adjust the delay. Fig. 7 shows the increase in average CPU utilization for different sleep times and buffer sizes with MAGNeT running on the sending machine. While these differences appear minor, even a small decrease in available CPU cycles can have a dramatic effect on application run-time and communication patterns (and thus, increased CPU utilization may result in large perturbations of the monitored traffic).

By comparison, `tcpdump` also loses information, approximately 15% on a 100Mbps network (for the extreme load conditions of our tests). This loss rate increases rapidly with higher network speeds because `tcpdump` does no buffering.

In summary, if MAGNeT's loss rate is too high, it can be adjusted to an acceptable level via the mechanisms discussed above; `tcpdump` lacks such adjustability. For example, to drop MAGNeT's loss rate to 0.5% while using the default 256K buffer, we adjust the `magnet-read` delay time to 35% of its original value (Fig. 6). According to Fig. 7, this would increase the overall CPU utilization from 53% to 55%. This is still less than the 60% CPU utilization seen with `tcpdump`.

C. Network Perturbation

From the measurements presented in this section, MAGNeT clearly performs as efficiently as `tcpdump` on contemporary networks and more readily scales to higher-speed networks. In addition, by adding CPU cycle-counter code around `magnet-add()` and relevant areas of `magnet-read`, we are able to compute the number of cycles, on average, that MAGNeT consumes while recording a single event. This value is of interest in comparison to the minimum interarrival time for packets on the physical network. For instance, on a 100Mbps Ethernet, a 40-byte Ethernet packet — the size of an "empty" TCP packet — will arrive no faster than $(40$ bytes $*$ 8 bits/byte$)/100$ Megabits/second $= 3.2$ μsec. Our conservative tests indicate that `magnet-add()` requires 556 cycles, on average, per recorded event while `magnet-read` requires 425 cycles. Thus, on our 400-MHz machines, MAGNeT takes $(556 + 425)$ cycles$/(400000000$ cycles/second$) = 2.4$ μsec to record a single event. Since this is less time than it takes a minimal TCP packet to arrive or to be sent, the MAGNeT-induced disturbances into the traffic stream should be quite small.

D. Design of `tcpdump` vs. MAGNeT

The performance numbers in this paper reflect a difference in design philosophies between `tcpdump` and MAGNeT. `tcpdump` is a purely user-level, network-inspection tool, which can be (and has been) relatively easily ported to a large number of different operating systems. Thus, many of the design tradeoffs in `tcpdump` favor portability over pure performance. In particular, `libpcap`, the packet capture library created for `tcpdump` and used by other network monitors, such as CoralReef, reflects this approach. The exact method used to intercept network packets varies depending on the features available in the root operating system, but it always involves a system call or other facility to cause a switch into kernel mode and a copy of memory from the kernel to the user-level program. This call-and-copy is repeated for every packet traveling across the interface being monitored. At high network speeds, the overhead of copying each individual packet between kernel and user space becomes a significant burden. MAGNeT benefits from having code embedded in the kernel to aggregate multiple network packets into a single space which then is copied in bulk, thus amortizing the cost of the copy over multiple packets. This approach incurs less overhead but is not as portable as `libpcap`'s method.

IV. MAGNeT in the Real World

We initially developed MAGNeT to investigate differences between traffic generated by an application and that same traffic after modulation by the protocol stack, i.e., when the traffic hits the network. Thus far, we have observed many situations where significant differences exist. For instance, Fig. 8 shows a one-second trace of an FTP session that was taken one minute into sending a Linux 2.2.18 kernel bzipped tar file from our facilities in Los Alamos, NM to a location in Dallas, TX — a trip of 30 hops across the Sprintlink backbone fabric. This graph shows a dramatic difference (in both packet size and inter-packet spacing) between application-generated traffic patterns and the traffic actually delivered to the network.

Network researchers routinely test new network and protocol designs using traces of traffic seen in current networks rather than the real traffic demands of applications. That

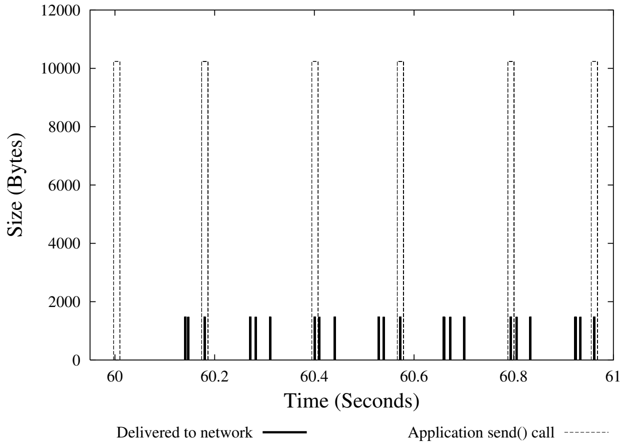

Fig. 8. MAGNeT FTP Trace

TABLE II
EFFECT OF MULTIPLE TCP-STACK TRAVERSALS

Trial	Data Size	Interpacket Spacing (sec)
Application	3284	0.124
1st TCP stack	1016	0.045
2nd TCP stack	919	0.037
3rd TCP stack	761	0.079
4th TCP stack	723	0.122

is, new designs are tested with network-delivered traffic, not application-generated traffic. Thus, differences such as those shown in Fig. 8 may have significant impact on the accuracy of such tests. For example, if the network-delivered traffic pattern from the above example is taken as the input to test the performance of an enhanced TCP stack (as would be the case if a `tcpdump` trace of the FTP session was used as input to a network simulation), the packet sizes and inter-packet spacing are once again modulated, as illustrated in Table II (in the row labeled "1st TCP stack"). The remainder of the table shows the effect of repeatedly using TCP output streams as inputs to subsequent TCP stacks.[5]

V. CONCLUSION

Current traffic libraries, network traces, and network models are based on measurements made by `tcpdump` (or similar tools such as PingER, NLANR Network Analysis Infrastructure, NIMI, and CoralReef). These tools do *not* capture an application's true traffic demands; instead they capture an application's demands *after* having been modulated by the protocol stack. Therefore, existing traffic libraries, network traces, and network models cannot provide any protocol-independent insight into the actual traffic patterns of an application.

MAGNeT fills the above void by providing a flexible and low-overhead infrastructure to monitor network traffic virtually anywhere in the protocol stack. Using MAGNeT, we have shown that the traffic demands of applications are *not* accurately reflected by the traffic on the network wire because the protocol stack (i.e., TCP) adversely modulates the application-generated traffic before it gets to the network wire. Hence, while current network models may accurately reflect current network-*wire* traffic, they are not useful in providing insight into application-generated traffic patterns nor in optimizing *application* communications.

The potential applications of MAGNeT include, but are not necessarily limited to, (1) constructing a library of traces of application-generated traffic, (2) verifying the correct operation of protocol enhancements to TCP, IP (also IPv6 and IPsec), or other protocols, (3) troubleshooting and tuning protocol implementations, and (4) security scanning. In this paper, we focused on the first application for MAGNeT — by providing a library of application-generated network traces, the network research community can drive their experiments (whether simulative and live) with real, application-generated traffic.

[5] Such repeated use is possible when communicating between end hosts in different domains. For example, when sending data via FTP between hosts at Los Alamos National Laboratory (LANL) and Sandia National Laboratory (SNL), the FTP connection is actually broken up into three consecutive connections, i.e., (1) LANL end host to LANL firewall, (2) LANL firewall to SNL firewall, and (3) SNL firewall to SNL end host. Thus, the initial application-generated traffic pattern that originated at the LANL end host goes through the protocol stack a total of six times (three sends and three receives).

REFERENCES

[1] "tcpdump," http://www.tcpdump.org.
[2] W. Matthews and L. Cottrell, "The PingER Project: Active Internet Performance Monitoring for the HENP Community," *IEEE Communications*, May 2000.
[3] S. Kalidindi and M. Zekauskas, "Surveyor: An Infrastructure for Internet Performance Measurements," in *INET'99 Proceedings*.
[4] A.J. McGregor, H-W Braun, and J.A. Brown, "The NLANR Network Analysis Infrastructure," *IEEE Communications*, May 2000.
[5] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, "An architecture for large-scale internet measurement," *IEEE Communications*, 1998.
[6] CAIDA, "CoralReef Software Suite," http://www.caida.org/tools/measurement/coralreef.
[7] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, February 1994.
[8] K. Park, G. Kim, and M. Crovella, "On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic," in *Proc. of the Int'l Conf. on Network Protocols*, October 1996.
[9] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, June 1995.
[10] V. J. Ribeiro, R. H. Riedi, M. S. Crouse, and R. G. Baraniuk, "Multiscale Queuing Analysis of Long-Range Dependent Network Traffic," in *Proc. of IEEE INFOCOM'00*, March 2000.
[11] P. Tinnakornsrisuphap, W. Feng, and I. Philp, "On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System," in *Proc. of the Int'l Conf. on Dist. Comp. Sys.*, April 2000.
[12] W. Feng and P. Tinnakornsrisuphap, "The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems," in *Proc. of the Int'l Conf. on Parallel Processing*, August 2000.
[13] W. Feng and P. Tinnakornsrisuphap, "The Failure of TCP in High-Performance Computational Grids," in *Proc. of SC 2000: High-Performance Networking and Computing Conf.*, November 2000.
[14] P. Danzig and S. Jamin, "tcplib: A Library of TCP Internetwork Traffic Characteristics," http://irl.eecs.umich.edu/jamin/papers/tcplib/tcplibtr.ps.Z, 1991.
[15] "The Internet Traffic Archive," http://ita.ee.lbl.gov/html/traces.html.
[16] A. Kato, J. Murai, and S. Katsuno, "An Internet Traffic Data Repository: The Architecture and the Design Policy," in *INET'99 Proceedings*.
[17] J. Semke, "PSC TCP Kernel Monitor," Tech. Rep. CMU-PSC-TR-2000-0001, PSC/CMU, May 2000.
[18] "Netperf," http://www.netperf.org.