Appeared in proceedings of 2nd Los Alamos Computer Science Institute Symposium (LACSI 2001), Santa Fe NM, October 15-18 2001.

The MAGNeT Toolkit: Design, Implementation, and Evaluation

Jeffrey Hay[†], Wu-chun Feng[†‡], Mark K. Gardner[†]
{jrhay, feng, mkg}@lanl.gov

[†] Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

[‡] Department of Computer & Information Science
Ohio State University
Columbus, OH 43210

Abstract—

The current trend in constructing high-performance computing systems is to connect a large number of machines via a fast interconnect or a large-scale network such as the Internet. This approach relies on the performance of the interconnect (or Internet) to enable fast, large-scale distributed computing. A detailed understanding of the communication traffic is required in order to optimize the operation of entire system.

Network researchers traditionally monitor traffic in the network to gain the insight necessary to optimize network operations. Recent work suggests additional insight can be obtained by also monitoring traffic at the application level.

The Monitor for Application-Generated Network Traffic toolkit (MAGNeT) we describe here monitors application traffic patterns in production systems, thus enabling more highly optimized networks and interconnects for the next generation of high performance computing systems.

Keywords—monitor, measurement, network protocol, traffic characterization, TCP, MAGNeT, traces, application-generated traffic, virtual supercomputing, network-aware applications, computational grids, high-performance computing.

I. Background

Modern high-performance computing environments, such as *Beowulf*-type clusters [1] and the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) [2], seek to achieve supercomputer performance by connecting many commodity computing nodes via a high-speed network interconnect. Additionally, recent supercomputing research focuses on building computational grids [3, 4] which form a virtual supercomputer from computing facilities at diverse sites, operating as a single system image by communicating on the Internet. In both types of systems, the network is a critical system component and a potential bottleneck.

Network performance is determined by a combination of the physical speed of the networking media, the protocols used to communicate information over that media, and the traffic patterns generated by the applications which use the network. The physical speed of the network is the upper limit on network performance, as traffic is unable to travel faster than media limits. However, the operation of network protocols such as TCP has been shown to place artificial limits on achievable network bandwidth. [5] These limits are caused by the interplay between the operation of the protocol and the network traffic required by the application. Thus, in order to improve network performance, network researchers must have a detailed understanding not only of the operation of current protocols, but of the network requirements of the applications themselves.

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 Los Alamos Unclassified Report (LA-UR) 01-4843.

Traffic monitors such as `tcpdump` [6], the CoralReef Software Suite [7], and Remote MONitoring (RMON) systems are valuable tools for obtaining information about active networks. Information gathered by traffic monitors can be used to verify the operation of network protocols, or can be combined into archives, such as the Internet Traffic Archive [8] and the Internet Traffic Data Repository [9], and used to generate models of global network traffic patterns.

Recent work suggests, however, that traditional traffic monitors miss a valuable part of the available information. [10–12] Specifically, the tools capture traffic *on the wire* (or *in the network*) rather than at the application level. Thus, the traffic an application sends to the network is captured only after having passed through a protocol stack (e.g., TCP/IP) and into the network. Consequently, these tools cannot provide protocol-independent insight into the traffic patterns of an application.

To determine application traffic patterns before being modulated by a protocol stack, as well as to determine the modulation caused by each layer of a protocol stack, we present the Monitor for Application-Generated Network Traffic (MAGNeT). The MAGNeT toolkit captures traffic (1) generated by applications, (2) passing through each layer of the protocol stack (e.g., from TCP to IP), and (3) entering and leaving the network. MAGNeT differs from existing tools in that traffic is monitored not only upon entering and leaving the network, but also throughout the entire network protocol stack, including at the application layer. Hence, MAGNeT provides network developers with information necessary to improve the performance of future supercomputers.

In this paper, we present the overall design of MAGNeT and discuss implementation details of our MAGNeT toolkit. We also present an evaluation of the performance of the MAGNeT toolkit, and conclude with some example uses for such a tool.

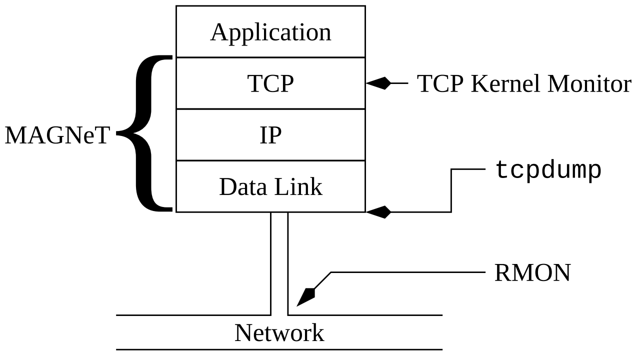

Fig. 1. Monitoring Points of Various Tools

A. Related Work

MAGNeT is a software-only solution to the problem of monitoring application-level network traffic. As Figure 1 shows, MAGNeT differs from `tcpdump`-like and RMON tools in that it makes fine-grained measurements throughout the entire protocol stack, not just at the network wire level. We are aware of two tools similar in nature to the MAGNeT toolkit.

One alternative is Pittsburgh Supercomputing Center's TCP kernel monitor [13]. MAGNeT differs from the TCP kernel monitor in three ways. First, MAGNeT can be used anywhere in the protocol stack, not just for monitoring TCP. Second, MAGNeT monitors a superset of the data that the TCP kernel monitor does. And third, MAGNeT runs on Linux whereas PSC's TCP kernel monitor works on NetBSD.

Bolliger and Gross describe a method of extracting network bandwidth information per TCP connection under BSD. [14] While their research tool appears to have a similar architecture to MAGNeT, their application is limited in scope, as it only records the specific information needed to compute estimated bandwidth for TCP connections. In addition, their tool does not appear to be publicly available, whereas we intend for MAGNeT to be included in the toolbox of every network researcher.

II. MAGNeT Design

The fundamental vision of the MAGNeT toolkit is to capture application-level network traffic patterns. To be of maximum benefit, MAGNeT traces must capture the behavior of real-world applications (i.e., monitoring simulated applications is of little benefit). In addition, no application modification or special user actions should be required.

To obtain this level of transparency, the MAGNeT toolkit must either perform its work within the communication library linked against the application or within the operating system (OS) kernel. Working in the communication library requires each monitored application to be re-compiled or re-linked against a MAGNeT-ized library. On the other hand, working in the OS kernel allows any existing application to be monitored. Placing MAGNeT in the OS kernel also allows it to record protocol level transitions (e.g., when a data packet is passed from TCP to IP), as well as network protocol state variables. Due to these factors, the MAGNeT toolkit is designed as a series of modifications to the OS kernel's networking stack.

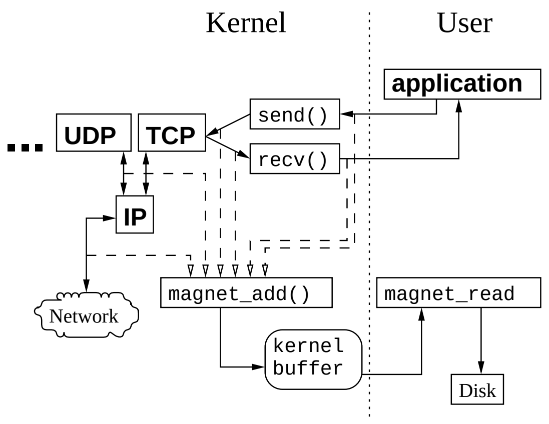

Fig. 2. Overview of MAGNeT Operation

The modified OS provides hooks throughout the protocol stack to call a MAGNeT data recording procedure. The hooks provide access to each layer of the stack, from the `send()` and `recv()` calls to the network device. Thus, the kernel hooks allow applications to run completely unmodified (since all changes are in the OS) and yet capture data throughout the protocol stack, including true application traffic patterns.

Also desirable is the ability to export collected data to user space in real time. Having run-time data available to applications allows for the development of network-aware applications. Recent work, including that performed by Bolliger and Gross [14], suggest that if applications know the run-time state of the network, they may better tune their network use to achieve maximum performance. MAGNeT records the type of information that is of interest to network-aware applications, and hence facilitates the development of these applications.

The flow of data in MAGNeT is shown in Figure 2. Applications run in the normal manner. These applications make `send()` and `recv()` system calls during the course of their execution. These calls eventually make use of TCP, IP, or other protocols to transfer data to and from the network. For systems running MAGNeT, each time a network protocol event occurs, the kernel makes a call to the MAGNeT recording procedure (which in our implementation is called `magnet_add()`). This procedure saves data to a circular buffer in kernel space, which is then saved to disk by a MAGNeT user-level application program (in our implementation this program is called `magnet-read`).

A. MAGNeT Timestamps

To accurately gauge the amount of time spent in protocol stack layers, MAGNeT requires high-fidelity timing. To this end, events are timestamped by MAGNeT using the highest-resolution time source available. On most systems, the source with the highest resolution is the CPU cycle counter, which increments on each CPU clock tick. If the speed of the CPU clock is known, then the difference between two cycle counts can be converted to elapsed time.

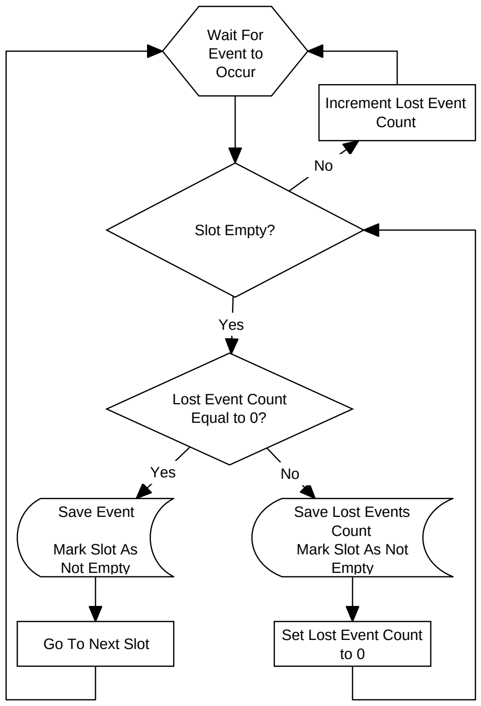

Fig. 3. MAGNeT Kernel Operation

B. User/Kernel Interface and Synchronization

The MAGNeT additions to the kernel export a circular buffer to user-space via shared memory. Because the kernel and user processes access the same area of physical memory, MAGNeT provides a means of synchronization between the two processes. This is accomplished by using a field of the instrumentation record as a synchronization flag between the MAGNeT user and kernel processes, as shown in Figures 3 and 4.

Before writing to a slot in the circular buffer, MAGNeT kernel code checks the synchronization field for that slot. If the field indicates that the slot has not yet been copied to user space, the kernel buffer is full. In this case, the kernel code increments a count of the number of instrumentation records that could not be saved due to the buffer being full. Otherwise, the kernel code writes a new instrumentation record and advances to the next slot in the circular buffer.

The user application accesses the same circular buffer via kernel-user shared memory. When the synchronization field at the current slot indicates the slot is ready to be copied to user space, the application reads the entire record and resets the synchronization field to signal the kernel that the slot is once again available. The application then advances to the next slot in the circular buffer.

When the kernel has a non-zero count of unsaved events and buffer space becomes available, the kernel writes a special instrumentation record to report the number of instrumentation records that were not recorded. Thus, during post-processing of the data, the fact that events were lost is detected at the appropriate chronological place within the data stream.

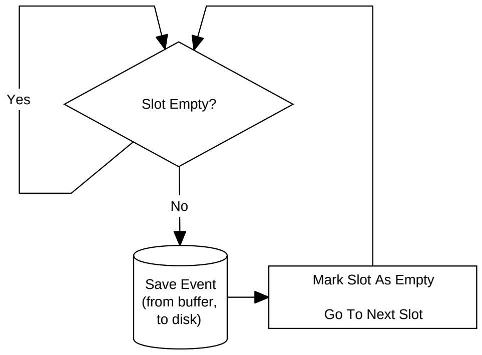

Fig. 4. MAGNeT User Operation

III. MAGNeT Implementation

We now discuss our implementation of the design principles outlined in section II. Due to our desire to monitor a large subset of our computing environment, we base our implementation on the Linux 2.4-series kernel. Our MAGNeT toolkit software distribution consists of a patchfile for the Linux kernel, three user-interface application programs, and a pair of scripts to automate distributed data collection.

A. MAGNeT-izing the Kernel

The primary changes to the Linux kernel to implement MAGNeT involve pinning a large area of physical memory for the circular buffer, exporting this buffer to user-space via shared memory, and writing instrumentation records to the buffer as packets progress through the kernel's protocol stack. Code to perform all these steps is located in the new kernel source file `net/magnet/magnet.c`.

The pinned circular buffer can be configured to any desired size. The default buffer size is 256KB. Increasing the size of the buffer uses more physical memory but reduces the potential for lost events. A full analysis of the effects of buffer size is conducted in Section IV-B.3.

`magnet_add()` is the MAGNeT data recording procedure, which adds a record to the circular buffer. `magnet_add()` is written to be very lightweight so that it can be called at multiple points in the protocol stack without inducing a significant amount of overhead in the protocol processing. (See Section IV-D for an analysis of the cost of `magnet_add()`.) The current distribution instruments the general socket-handling code, the TCP layer, and the IP layer. Other protocols and layers may be instrumented by placing calls to `magnet_add()` at appropriate locations in the code for that protocol or layer.

Linux exports kernel/user shared memory via device files. That is, the kernel denotes a shared memory segments by a device file which user-space programs may open. Opening this file causes Linux to create a shared memory region which can then be mapped into the application's address space with the `mmap()` system call. With this mapping in place, no additional kernel code is executed; the application program simply reads the shared memory and writes it to disk.

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

Fig. 5. The MAGNeT Instrumentation Record

A.1 Instrumentation Records

Instrumentation records contain the data which `magnet_add()` stores in the buffer for each instrumented event. Our implementation uses fixed-sized instrumentation records to minimize the time spent recording individual events. The file `include/linux/magnet.h` contains the definition of our instrumentation record, which is reproduced in Figure 5.

`sockid` is a unique identifier for each connection stream.[1] This allows data traces to be separated into individual streams during post processing, while protecting the privacy of the application and user. The `timestamp` field contains a CPU cycle count which serves not only to provide time measurements for MAGNeT traces, but also acts as the synchronization flag between the user and kernel processes, as described in Section II-B. (A `timestamp` of zero indicates the record has been copied to user-space; a non-zero value indicates the record has not yet been saved.) The `event` field indicates to what class of events a particular record belongs. Valid values for the `event` field (e.g., `MAGNET_IP_SEND`) are given by an `enum` declaration at the beginning of `magnet.h`. The `size` field contains the number of bytes transferred during a specific event.[2] The `data` field (an optional field selected at kernel compilation time) is a union of structures in which information specific to a particular protocol can be stored, thus providing a mechanism for MAGNeT to record protocol state information along with event transitions.

A.2 Instrumented Events

In its default configuration (i.e., without the optionally-compiled `data` field), our toolkit records only the timestamp and associated data size for each transition between network stack layers. That is, MAGNeT will record an event indicating when the socket handling code receives data from an application (and how much data was received), when the TCP layer receives data from the socket handling code, when the IP layer receives data from TCP, and, finally, when IP hands the data off to the network device driver. (A similar set of events is recorded for the receive pathway.)

With the `data` field compiled in, MAGNeT records extensive data about the instantaneous state of the protocol being monitored. This data consists of all protocol header information as well as run-time protocol state variables which are not usually available. As an example of the kind of information stored within the `data` field, Figure 6 shows the union members for TCP and IP events.

[1] The `sockid` field contains the physical memory address of the kernel's status information for the connection.

[2] A negative value in the `size` field reflects the error code returned by the function causing the event.

A.3 MAGNeT System Information

Our MAGNeT implementation uses the Linux kernel's `get_cycles()` function to generate CPU cycle-counter timestamps. To provide a means of calculating wall-clock time from the timestamps, the first record stored by our toolkit in the circular buffer is a record of type `MAGNET_SYSINFO`, whose `size` field contains the processor clock speed in KHz.

In addition to providing CPU clock rate information, the `MAGNET_SYSINFO` instrumentation record also allows MAGNeT to be endian-aware. Since this record is guaranteed to be the first record in the circular buffer every time the MAGNeT device file is opened, it will be the first record that the user-application reads and saves to disk. Post processing software can use the value of the `event` field of this record to determine if the trace was saved on a big-, little-, or mixed-endian machine. Specifically, the `MAGNET_SYSINFO` event type is defined as the hexadecimal value 01234567. If the first record read by the data processor has an `event` field of this value, no endian translation is necessary. On the other hand, if the `event` field contains a different value (e.g., a value of hexadecimal 67452301), the file was saved on a machine with a different endian orientation than the processing machine, so endian translation is necessary.

A.4 `/proc/net/magnet`

The `/proc/net/magnet` file allows user applications to determine the state of the MAGNeT kernel process. The existence of this file is proof that the kernel has been MAGNeT-ized (that is, the MAGNeT code is active in the kernel). Displaying this file yields information such as the major and minor numbers for the MAGNeT shared memory device file, the size of the circular buffer, and other information that may be useful to user-level applications.

A.5 MAGNeT User-level Interface

In our current distribution of MAGNeT, the user-level interface consists of three programs, `magnet-read`, `mkmagnet`, and `magnet-parse`, along with a pair of scripts to ease automated traffic trace generation and collection.

`magnet-read` is the primary means of obtaining MAGNeT traffic traces; its function is to read the data from the kernel's circular buffer. Our implementation makes use of the memory-mapped I/O features of the Linux kernel. Linux memory-mapped I/O requires a file to exist before it is mapped into memory. We refer to this file as a "binary trace file." Once a binary trace file exists, `magnet-read` maps this file into its memory space, and then saves data to the file by simply performing a memory copy between the kernel-user shared mem-

```
struct magnet_tcp {
  /* data from "struct tcp_opt" in
     include/net/sock.h */

  unsigned short  source;
  /* TCP source port */
  unsigned short  dest;
  /* TCP destination port */

  unsigned long snd_wnd;
  /* Expected receiver window */

  unsigned long srtt;
  /* smothed round trip time << 3 */
  unsigned long rto;
  /* retransmit timeout */

  unsigned long packets_out;
  /* Packets which are "in flight" */
  unsigned long retrans_out;
  /* Retransmitted packets out */

  unsigned long snd_ssthresh;
  /* Slow start size threshold */
  unsigned long snd_cwnd;
  /* Sending congestion window */

  unsigned long rcv_wnd;
  /* Current receiver window */
  unsigned long write_seq;
  /* Tail+1 of data in send buffer */
  unsigned long copied_seq;
  /* Head of yet unread data */

  /* TCP flags*/
  unsigned short  fin:1,syn:1,rst:1,
      psh:1,ack:1,urg:1,ece:1,cwr:1;
}; /* struct magnet_tcp */

struct magnet_ip {
  unsigned char  version;
  unsigned char  tos;
  unsigned short id;
  unsigned short frag_off;
  unsigned char  ttl;
  unsigned char  protocol;
}; /* struct magnet_ip */
```

Fig. 6. MAGNeT Extended Data for TCP and IP

ory and the memory region mapped to the file. This approach allows MAGNeT to record data on high-speed networks with minimal chance of record loss. The `mkmagnet` application creates and initializes the binary trace file prior to it being mapped into memory. The program `magnet-parse` reads data collected by `magnet-read` and dumps a tab-delimited ASCII trace of the collected data for further processing, performing endian translation as necessary.

The current MAGNeT distribution also includes the two shell scripts `magnet.cron` and `magnet.copy`. These two scripts allow network administrators to create an automated application-monitoring environment. `magnet.cron`, the overall MAGNeT management script, ensures that the MAGNeT device file exists and that a binary trace file has been created by `mkmagnet`. If invoked while `magnet-read` is running, the script terminates the current MAGNeT data collection session and calls `magnet.copy` to transfer the data to a remote archive.[3] After all these tasks are completed, `magnet.cron` restarts `magnet-read` to save network events to disk. By design, the management script may be run periodically (e.g., every midnight) to collect data on a diverse set of machines without requiring special action by the users.

IV. MAGNeT Performance Analysis

In this section, we determine the effect of running our MAGNeT implementation via a variety of tests. We compare attainable bandwidth and the resultant CPU utilization on a system running MAGNeT to the same system running `tcpdump`, as well as the same system running no monitoring software. We use this comparison to verify that our implementation of the MAGNeT toolkit performs application-level monitoring without causing significant variation in the traffic pattern of live applications, and without appreciably affecting system usage.

A. Experimental Method

To determine the overhead of running MAGNeT, we measure the maximum data rate and the CPU utilization between a sender and receiver with and without MAGNeT. For comparison, we also measure the overhead of running `tcpdump`. In total, the six configurations shown in Table I are compared.

Our baseline configuration runs between two machines with stock Linux 2.4.3 kernels. The second configuration uses the same machines but with the MAGNeT patches installed. Although present in memory, MAGNeT records are not saved to disk. The third configuration is the same as the second except `magnet-read` runs on the receiver to drain the MAGNeT buffer. The fourth configuration is also the same as the second, but with `magnet-read` on the sender. For the fifth and sixth configurations, `tcpdump` is run on stock Linux 2.4.3 kernels.

[3]Since `magnet.copy` is not called while `magnet-read` is running, the extra traffic produced by the transfer will not be captured by MAGNeT. This behavior can easily be changed by re-ordering the commands in `magnet.cron`.

Bandwidth	Configuration	Throughput (Kb/s)		Send CPU (%)		Receive CPU (%)	
100 Mbps	Linux 2.4.3	94.14	± 0.00	15.19	± 0.12	33.49	± 0.06
	Linux 2.4.3 w/MAGNeT	94.13	± 0.01	16.93	± 0.21	33.45	± 0.06
	`magnet-read` on receiver	90.79	± 0.82	20.73	± 0.29	34.35	± 1.02
	`magnet-read` on sender	90.69	± 0.88	23.69	± 1.73	32.39	± 0.35
	`tcpdump` on receiver	89.39	± 1.48	18.00	± 0.37	59.78	± 0.88
	`tcpdump` on sender	89.04	± 0.84	45.00	± 0.63	31.86	± 0.31
1000 Mbps	Linux 2.4.3	459.48	± 1.63	61.03	± 0.30	82.43	± 0.23
	Linux 2.4.3 w/MAGNeT	452.46	± 1.82	62.99	± 0.35	82.58	± 0.29
	`magnet-read` on receiver	444.31	± 1.66	62.36	± 0.31	81.96	± 0.28
	`magnet-read` on sender	440.24	± 2.11	63.08	± 0.52	81.10	± 0.37
	`tcpdump` on receiver	290.68	± 15.64	36.05	± 1.98	91.50	± 0.50
	`tcpdump` on sender	343.22	± 18.71	93.17	± 0.45	64.07	± 3.32

TABLE I
MAGNeT vs. `tcpdump` Performance

The fifth configuration runs `tcpdump` on the receiver, while the sixth runs `tcpdump` on the sender. All configurations are tested on both 100Mbps and 1000Mbps Ethernet networks.

We conduct our tests between two identical dual 400MHz Pentium IIs with NetGear 100Mbps and Alteon 1000Mbps Ethernet cards. MAGNeT is configured to record application socket calls as well as TCP and IP events, using the default 256KB kernel buffer to store event records.

As a workload, we use `netperf` [15] on the sender to saturate the network.[4] 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 services.

B. Performance

Although MAGNeT records a different set of information than `tcpdump` (i.e., MAGNeT records application and protocol stack-level traffic while `tcpdump` only records network-wire traffic), we compare performance with `tcpdump` as the closest commonly-available tool. Table I lists the performance of MAGNeT and `tcpdump` for our tests. Along with the mean, the width of the 95% confidence interval is given. Figures 7 and 8 present this data graphically.

By default (and as used in our experiments), `tcpdump` stores the first 68 bytes of every packet. As configured, MAGNeT stores 96 bytes for each packet.[5]

B.1 Network Throughput

The kernel-resident portion of MAGNeT executes whether information is being saved to disk or not. The first data point in Figure 7, labeled "MAGNeT," shows virtually no penalty when data is not being saved to disk. The next two data points

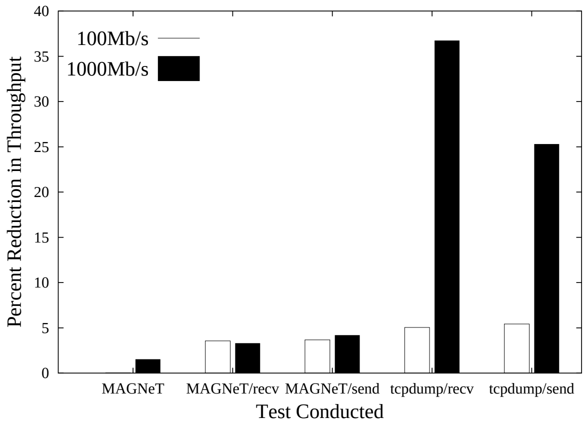

Fig. 7. Percent Reduction in Network Throughput

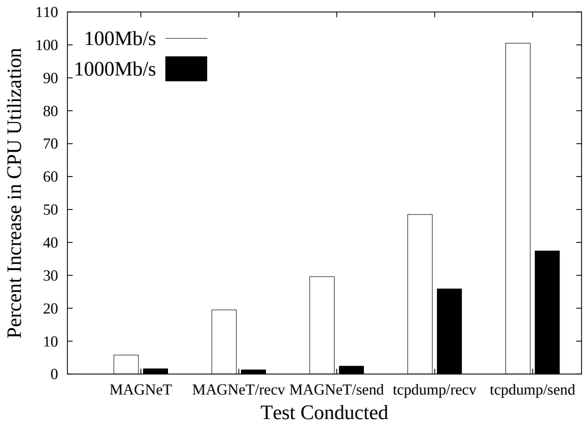

Fig. 8. Average Percent Increase in CPU Utilization

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

[5]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).

show MAGNeT incurs less than a 5% reduction in network throughput when `magnet-read` runs on either the receiver or sender (even though the network is saturated). 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 than `tcpdump` with increasing link speeds.

B.2 CPU Utilization

We next compare the CPU utilization reported by `netperf` under both MAGNeT and `tcpdump`. In Linux, `netperf` estimates CPU load by creating a low-priority process which increments a counter. Since this process is the lowest priority task in the system, the counter is only incremented when the CPU would otherwise be idle. Thus, a low counter value implies a high CPU utilization, and a high counter value implies low CPU utilization. Using this feature in the above set of tests, we estimate the additional CPU load incurred by both MAGNeT and `tcpdump`. The increase in CPU load for the above tests is shown in Figure 8.

MAGNeT requires less CPU (averaged over both sender and receiver) 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. Note that the overhead of both MAGNeT and `tcpdump` appears to decrease when run on the faster network. This is due to interrupt coalescing — the network interface cards used for the 1000Mbps experiments accumulate several incoming packets before interrupting the CPU. Thus, the average overhead of servicing interrupts is greatly reduced. If interrupt coalescing were disabled, the average CPU utilization for both MAGNeT and `tcpdump` would increase.

Finally, we mention that unlike `tcpdump`, the overhead of MAGNeT can be tuned. We discuss tuning MAGNeT for specifi c environments in the next section.

B.3 Event Loss

Analysis of the MAGNeT-collected data of our tests reveals that MAGNeT occasionally fails to record events at high network utilization. On a saturated network, MAGNeT did not record approximately 3% of the total events for the 100Mbps trials, while for the 1000Mbps tests the loss rate approached 15%. These losses are due to the 256KB buffer in the kernel fi lling before `magnet-read` is able to drain it.

Our implementation provides two methods for reducing the event loss rate: (1) increasing the kernel buffer size and/or (2) reducing the time `magnet-read` waits before draining the kernel buffer. Figure 9 shows the effect of these parameters on event loss rate for the 100Mbps saturated network tests. [6]

Increasing the kernel buffer size dramatically reduces MAGNeT's event loss rate, down to virtually no lost events under

[6]All other tests discussed in this paper are conducted with MAGNeT's default values for these two parameters.

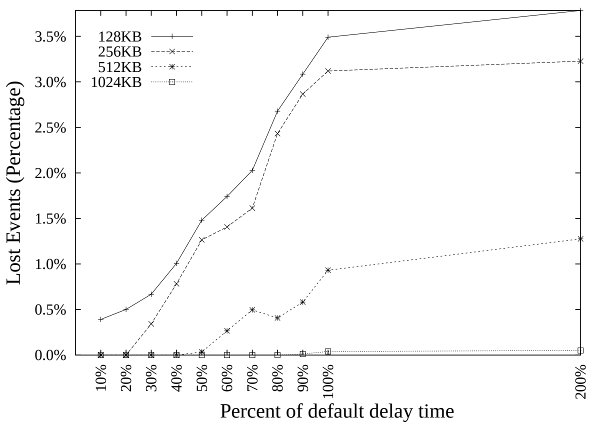

Fig. 9. MAGNeT's Event-Loss Rate, 100Mbs Ethernet

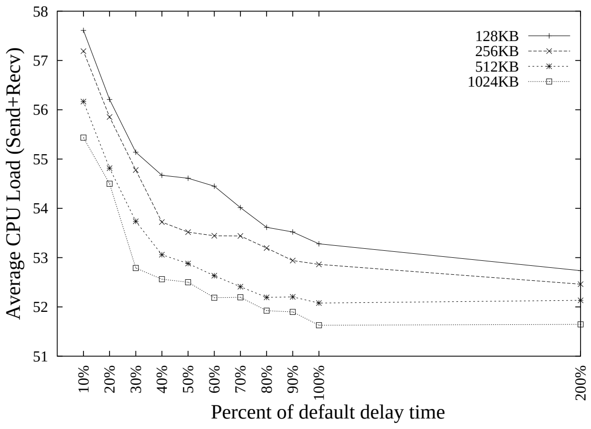

Fig. 10. MAGNeT's Average CPU Utilization, 100Mbs Ethernet

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. The default 256KB buffer size is a compromise between CPU utilization and physical memory consumed.

Another method for reducing event loss entails adjusting the amount of time `magnet-read` sleeps before draining the kernel buffer. Shorter sleep times cause the buffer to be drained more frequently, thus reducing the chance of lost events. 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.

The default sleep-time is computed as the amount of time taken to fi ll the kernel buffer on a saturated 100Mbps network. This heuristic was chosen because it provides relatively low event-loss rates without signifi cantly impacting the user. Command-line options to `magnet-read` allow adjustment of the delay.

Figure 10 shows the increase in average CPU utilization for different sleep times and buffer sizes with MAGNeT running on the sending machine. The high CPU utilization reported in this graph are due to our test procedure of flooding the network with `netperf`, which places an unusually high load on the system CPU. While the utilization differences appear minor, even a small decrease in available CPU cycles can have a dramatic effect on application run-time and communication patterns. Conversely, increased CPU utilizations result in larger perturbations of the monitored traffic.

In all cases, the average CPU utilization of MAGNeT is less than that of `tcpdump`.

By comparison, loss rates for `tcpdump` are significantly higher. Under Linux, `tcpdump` always reports a dropped packet count of zero, so we must estimate the true loss rate. We use UDP traffic to arrive at our estimation. `tcpdump` can record UDP packets as well as TCP, and `netperf` can send UDP if requested. In addition, `netperf` reports the number of packets sent, and `tcpdump` does accurately record the number of packets received. Thus, using UDP it is easy to compute an average loss rate for `tcpdump` – the difference between the number of packets sent by `netperf` and the number received by `tcpdump` is the number of packets not processed (dropped) by `tcpdump`. Using this estimation methodology, we obtain average packet loss rates for `tcpdump` of approximately 15% on 100Mbps networks for the conditions of our tests.

Because `tcpdump` does no buffering, loss rates will increase as network speeds increase. In contrast, 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 (Figure 9). According to Figure 10, this increases the overall CPU utilization from 53% to 55%. A CPU utilization rate of 55% is still less than the 60% seen with `tcpdump`.

C. Streaming MAGNeT

To assess the effect of MAGNeT on streaming media, we set up a web server on one of our test machines to stream an 8-minute, 51-second MPEG clip of *Crocodile Dundee* via HTTP. We determine the performance running in a stock configuration, running with MAGNeT executing only on the server, and running with MAGNeT executing only on the client. The results are shown in Figure 11. Since streaming MPEG clients usually buffer data to maintain an even framerate, counting frames per second is not a valid measure of streaming performance. Therefore, the metric we use is total time taken for the entire clip to be sent to the client.

MAGNeT has minimal effect on MPEG streaming. Over 100 trials, the average time to stream the movie clip on a 100Mbps network without MAGNeT is 46.02 seconds, with a 95% confidence interval of ±0.07 seconds. To transfer the movie clip with MAGNeT running on the server took an average of 46.07 seconds, and with MAGNeT on the client, 46.05 seconds. (Both MAGNeT cases have a 95% confidence interval of ±0.06 seconds.)

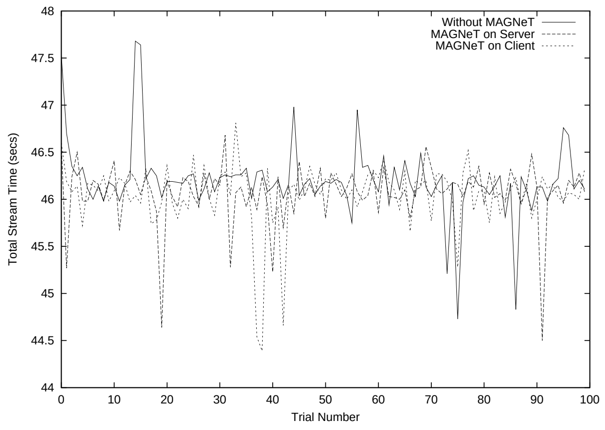

Fig. 11. Streaming MAGNeT Effects

D. Network Perturbation

From the previous measurements, MAGNeT performs at least as efficiently as `tcpdump` on contemporary networks and scales more readily to higher-speed networks. Another critical metric of application traffic monitors is the extent to which operation of the monitors disturbs the traffic patterns being monitored.

By adding CPU cycle-counter code around `magnet_add()` and relevant areas of `magnet-read`, we can estimate the number of cycles, on average, that MAGNeT consumes while recording data. This value can then be compared to the minimum interarrival time for packets on the physical network.

On a 100Mbps Ethernet, a 40-byte Ethernet packet — the size of an "empty" TCP packet — will arrive no faster than $(40 \text{ bytes} * 8 \text{ bits/byte})/100 \text{ Megabits/second} = 3.2\ \mu\text{sec}$. Our conservative tests indicate that `magnet_add()` consumes 556 cycles, on average, while recording a packet, while `magnet-read` requires 425 cycles. Thus, on our 400-MHz machines, MAGNeT takes $(556 + 425) \text{ cycles}/(400000000 \text{ cycles/second}) = 2.4\ \mu\text{sec}$ to monitor a single network packet. Since this is less time than a minimal TCP packet takes to arrive or to be sent, the MAGNeT-induced disturbances into the traffic stream should be small.

V. Implications and Applications

We have shown that the MAGNeT toolkit provides a transparent method of gathering application traffic data on individual machines. MAGNeT meets its goal of generating application traffic pattern traces and OS protocol stack operation traces while causing minimal interference of the patterns being monitored. In this section, we provide examples of how MAGNeT-collected information can be utilized when designing next-generation high performance computing environments.

A. Traffic Pattern Analysis

One use of the MAGNeT toolkit is to investigate differences between the traffic generated by an application and that same traffic as it appears on the network (i.e., after modulation by a

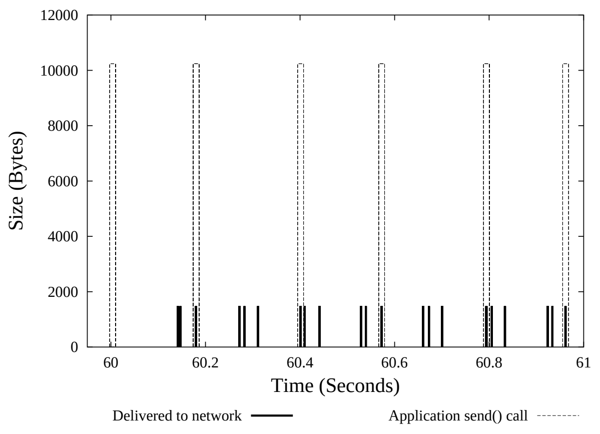

Fig. 12. MAGNeT FTP trace

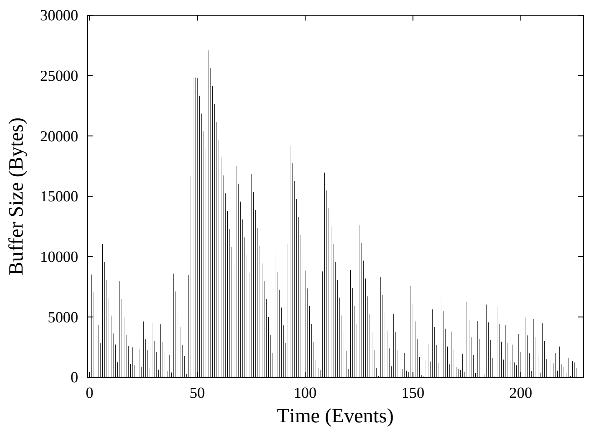

Fig. 13. Size of Sending TCP Buffer

protocol stack). As a simple example of the kind of modulation possible, we consider a trace of a FTP session from our facility in Los Alamos, NM to a location in Dallas, TX. A one-second MAGNeT trace, taken one minute into the transfer, is shown in Figure 12.

As can be seen by examining the graph, the FTP application attempts to send 10KB segments of data every 20 milliseconds, but the protocol stack (TCP and IP in this case) modulates the traffic into approximately 1500 byte packets at intervals of varying duration. Since the maximum data size on an Ethernet network is 1500 bytes, the protocol stack segments the data to this size. The variable spacing of the traffic intervals is caused by TCP waiting for positive acknowledgements before sending more traffic.

If we send the traffic stream *as it was delivered to the network* through another TCP stack (which is done, for instance, when conducting network simulations with `tcpdump` data), we again see modulation. Each subsequent run of network-delivered traffic through TCP further modulates the traffic, as shown in Table II. Thus, we see that TCP significantly perturbs traffic patterns, even when the initial traffic pattern has previously been shaped by TCP. This conclusion implies that wire-level traffic traces inaccurately represent true application networking requirements.

TABLE II
EFFECT OF MULTIPLE TCP STACKS

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

B. Next-Generation Interconnect Design

MAGNeT gives network designers the knowledge of what applications expect from the network. This knowledge is beneficial when optimizing application communication across networks and supercomputer interconnects. For example, conventional traffic monitors (which are only capable of capturing network-delivered traffic) would report the traffic of the FTP session in Figure 12 to be composed of short bursts of data at erratically spaced intervals. Communication fabrics designed to optimize performance using that traffic pattern may not adequately handle the traffic pattern intended by the application (longer bursts of data at consistently spaced intervals). Thus, fabrics designed by referring to traces generated by traditional network monitors fail to meet the true needs of the application.

Traces generated by MAGNeT provide a realistic picture of the protocol-independent traffic demands of applications running on today's networks. Thus, MAGNeT provides network developers with a better understanding of the requirements of future networks and high-performance cluster interconnects.

C. Resource Management

With the optional `data` filed compiled in, MAGNeT can return snapshots of the complete protocol state, information previously only available under simulation environments, during execution of real applications on live networks. This kind of data is invaluable when planning proper resource allocation on large computing systems.

For example, Figure 13 shows the size of the sending TCP buffer during the MAGNeT streaming trials discussed in Section IV-C. This buffer contains data recorded as "sent" by the application, but not yet actually delivered to the network by TCP. The buffer reaches a maximum size of around 30KB, but averages 6.5KB for the life of the connection. With this kind of information, a resource allocation strategy which conforms to the true needs of applications may be developed.

D. Network-Aware Application Development

As discussed in Section II, MAGNeT captures data which network-aware applications can use to appropriately tune their performance. In our implementation, any application is able to open the MAGNeT device file and map a portion of their memory space to the MAGNeT data collection buffer. Thus, an daemon may be developed which monitors the MAGNeT collected data and provides a summary of the data for specific connections at the request of network-aware applications. This strategy consolidates all network monitoring activity to amortize the overhead across all network-aware applications running on the system.

VI. Future Work

Our implementation of MAGNeT can be improved in several ways. We would like to allow the user to set various MAGNeT parameters (e.g., the kinds of events to be recorded, the size of the kernel buffer, etc.) at run-time rather than at kernel compile-time. Allowing run-time user configuration of the MAGNeT toolkit could be accomplished by making the current `/proc` file writable. Run-time configuration would greatly increase the usability and flexibility of the MAGNeT toolkit.

Another potential area of improvement in MAGNeT is the mechanism used to store recorded data from the kernel buffer to disk. Rather then have a user-level process, a better approach may be to utilize kernel threads to perform all steps of the instrumentation. With this methodology, the need for the special device file, the file created by `mkmagnet`, and the kernel/user shared memory would be eliminated. In addition, kernel threads may lower MAGNeT's already low event loss rate by eliminating the overhead of `magnet-read`. However, implementing kernel threads may impede exporting real-time data to network-aware applications. The use of kernel threads may be explored for future versions of MAGNeT.

Timing with CPU cycle counters can be problematic on contemporary CPUs which may change their clock rate according to power management policies. If the kernel detects such changes, MAGNeT could easily hook into the clock-rate detection code and output new `MAGNET_SYSINFO` events. These events, containing new timing information, would allow correct post-processing in spite of CPU clock-rate changes. However, current Linux production kernels are unable to detect CPU clock rate changes at run-time. We do not consider this a serious problem as the MAGNeT toolkit is intended to be used primarily on high-performance computing systems which are unlikely to lower clock rates at run-time.

For applications which are compiled to use system shared libraries (rather than statically-compiled libraries), an alternative method of gathering application traffic patterns is to provide a shared library which instruments network calls before passing the calls on to the original system library. Since MAGNeT records network call events only within the kernel, the use of such an instrumented library (which records application events in user space, before any context switch) is complimentary to the approach taken in MAGNeT. Using such a library in conjunction with MAGNeT would allow system call overhead to be quantified while still requiring no change to applications.

VII. Conclusion

Current traffic libraries, network traces, and network models are based on measurements made by `tcpdump` (or similar tools such as 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 protocol-independent insight into the actual traffic patterns of an application.

Information regarding the networking needs of applications, as well as the operation of network protocol stacks, is essential to the construction of modern high-performance computing systems. Current network trace generation tools and archives are inadequate for this purpose. Designers and researchers are left with no substantive data regarding the networking needs of applications.

The MAGNeT toolkit fills the void by providing a flexible and low-overhead infrastructure to monitor network traffic anywhere in the protocol stack. The MAGNeT architecture provides a framework for generating a new kind of network traffic trace, giving high-performance computer designers and implementers new insight into potential bottlenecks of next-generation machines.

Availability

The MAGNeT toolkit (the Linux 2.4 kernel patch, the user-application programs, and supporting material) is available from our website, http://www.lanl.gov/radiant. Other documents relating to MAGNeT may also be found on our website.

References

[1] "The Beowulf Project," `http://www.beowulf.org`.

[2] "Accelerated Strategic Computing Initiative," `http://www.asci.doe.gov/index.htm`.

[3] "Distributed Systems Department," `http://grid.lbl.gov`.

[4] "Grid Computing Info Centre," `http://www.gridcomputing.com`.

[5] E. Weigle and W. Feng, "A Case for TCP Vegas in High-Performance Computational Grids," *Proc. of the 10th Annual Int'l Symposium on High Performance Distributed Computing*, August 2001.

[6] "`tcpdump`," `http://www.tcpdump.org`.

[7] CAIDA, "CoralReef Software Suite," `http://www.caida.org/tools/measurement/coralreef`.

[8] "The Internet Traffic Archive," `http://ita.ee.lbl.gov/html/traces.html`.

[9] A. Kato, J. Murai, and S. Katsuno, "An Internet Traffic Data Repository: The Architecture and the Design Policy," in *INET'99 Proceedings*.

[10] 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.

[11] 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.

[12] 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.

[13] J. Semke, "PSC TCP Kernel Monitor," Tech. Rep. CMU-PSC-TR-2000-0001, PSC/CMU, May 2000.

[14] J. Bolliger and R. Gross, "Bandwidth Monitoring for Network-Aware Applications," *Proc. of the 10th Annual Int'l Symposium on High Performance Distributed Computing*, August 2001.

[15] "Netperf," `http://www.netperf.org`.