Journal of Supercomputing, Vol. 23, pp. 67-79, August 2002.

The MAGNeT Toolkit: Design, Implementation and Evaluation [*]

Wu-chun Feng (feng@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

Mark K. Gardner (mkg@lanl.gov)
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

Jeffrey R. Hay (jrhay@lanl.gov)
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

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.

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

1. Background

Modern high-performance computing environments, such as *Beowulf*-type clusters [6] and the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) [1], 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 [2, 3] 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. [14, 19] 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.

Traffic monitors such as `tcpdump` [5], Remote MONitoring (RMON) systems, and the CoralReef Software Suite [9] 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 [7] and the Internet Traffic Data Repository [16], 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. [12, 13, 18] 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-

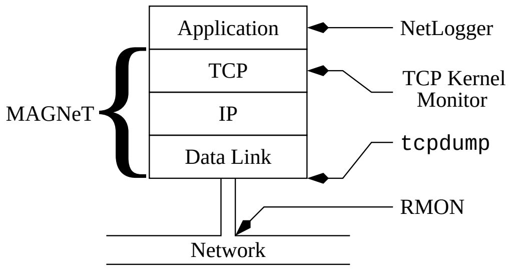

Figure 1. Monitoring Points of Various Tools

Generated Network Traffic (MAGNeT). The MAGNeT toolkit 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 developers with information necessary to improve the network performance of future virtual supercomputers and computational grids.

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.

1.1. Related Work

MAGNeT is a software-only solution to the problem of monitoring application-level network traffic. It requires no modifications to monitored applications, has low overhead and captures information throughout the protocol stack (Figure 1). Although there are a wealth of tools for monitoring network operations, We are aware of just three tools that are similar in nature to the MAGNeT toolkit.

One alternative is Pittsburgh Supercomputing Center's TCP kernel monitor [17]. MAGNeT differs from the TCP kernel monitor in least 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. [8] 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. It also has significant per-packet processing overhead and hence does not scale well.

The final tool is NetLogger [10]. Its purpose is to collect, correlate and present information about the state of a distributed system. It includes tools for instrumenting applications, host systems, and networks. It also presents tools for visualizing the collected data. Because of its focus on overall system dynamics, NetLogger is better than MAGNeT at presenting an overall view of complex distributed system behaviors that are the result of the interaction of multiple components such as network, disk and CPU activity.

On the other hand, MAGNeT monitors all applications without modification. It does not require applications to be recompiled or re-linked. MAGNeT also provides greater detail about the state of the network protocol stack than NetLogger. Furthermore, MAGNeT's timestamps are several decimal orders of magnitude more accurate. Thus, we view MAGNeT as complimentary to NetLogger and are considering making MAGNeT's output compatible to leverage NetLogger's visualization tools.

2. MAGNeT Design

The primary motivation of the MAGNeT toolkit is to capture network traffic patterns of real-world applications *at the application-level.* As such, no application modification or special user actions should be required. Furthermore, the performance of the system and network must not be significantly reduced in order for the traces collected by MAGNeT to be valid. In other words, the operation of the MAGNeT toolkit must be transparent to applications and users.

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, either statically or dynamically, 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.

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 [8], suggest that if applications

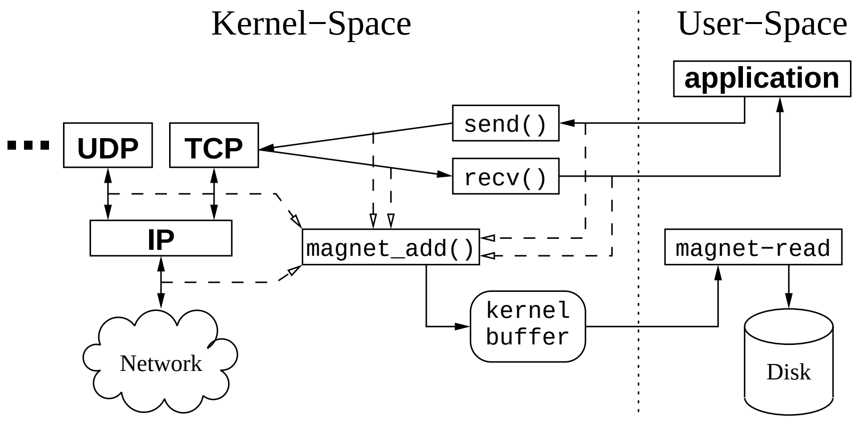

Figure 2. Overview of MAGNeT Operation

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, running without modification, make `send()` and `recv()` system calls that 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`).

2.1. 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.

2.2. Kernel-User Interface and Synchronization

The MAGNeT kernel modifications export a circular buffer to user-space via kernel-user 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.

Before writing to a slot in the circular buffer, the 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, indicates that the slot is now occupied 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 that the slot is ready to be copied to user space, the application reads the entire record and resets the synchronization field to signal to 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 place within the event stream.

3. MAGNeT Implementation

We now discuss our implementation of the design principles outlined in section 2. (For a more detailed discussion, see [15].) 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.

3.1. MAGNeT-izing the Kernel

The changes necessary to implement MAGNeT in the Linux kernel involve pinning an 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.

The circular buffer can be configured to any desired size. The default buffer size is 256 KB. Increasing the size of the buffer uses more physical memory but reduces the potential for lost events. An analysis of the effects of buffer size is conducted in Section 4.3.

`magnet_add()`, which adds a record to the circular buffer, is written to be lightweight so that it can be called at multiple points in the protocol stack without inducing a significant amount of overhead. The current implementation 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.

A user-space application, such as `magnet-read`, creates a shared memory region containing the circular buffer and maps the region into the application's address space. Thereafter, no additional kernel code is executed; the application program simply reads shared memory and writes it to disk.

3.1.1. *Instrumentation Records*

In our implementation, instrumentation records are of fixed-size to minimize time spent by `magnet_add()` recording individual events. Each instrumentation record contains an identifier which is unique across all open connections. This allows data traces to be separated into individual streams during post processing, while protecting the privacy of the application and user. Records also have a `timestamp` field 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 2.2. A `timestamp` of zero indicates the record has been copied to user-space; a non-zero value indicates the record has not yet been read. The remaining two mandatory fields, `event` and `size`, indicates the type of event and the number of bytes transferred. There is an optional `data` field, which can be included at kernel compilation time, that is a union of information specific to particular protocols. The optional data field provides a mechanism for recording protocol state information along with event transitions. (For more details, see [15].)

3.1.2. *MAGNeT System Information*

Our MAGNeT implementation uses the `get_cycles()` function of the Linux kernel to generate CPU cycle-counter timestamps. The first record stored by our toolkit is of type `MAGNET_SYSINFO` whose `size` field contains the processor clock speed in KHz. The record also contains data which allows the user-level process to determine if the trace was saved on a big-, little-, or mixed-endian machine.

Table I. Test Configurations

Configuration Number	Configuration
1	Linux 2.4.3
2	Linux 2.4.3 w/MAGNeT
3	Linux 2.4.3 w/MAGNeT, `magnet-read` on receiver
4	Linux 2.4.3 w/MAGNeT, `magnet-read` on sender
5	Linux 2.4.3, `tcpdump` on receiver
6	Linux 2.4.3, `tcpdump` on sender

4. MAGNeT Performance Analysis

In this section, we quantify the performance impact of our MAGNeT implementation. We compare attainable bandwidth and CPU utilization on a system running MAGNeT to the same system running `tcpdump`,[1] as well as the same system running no monitoring software. We use this comparison to show that our implementation of the MAGNeT toolkit does not cause a significant variation in the traffic pattern of live applications and does not appreciably affecting system usage.

4.1. Experimental Method

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

We conduct the tests between two identical dual 400MHz Pentium IIs with NetGear 100 Mbps and Alteon 1000 Mbps Ethernet cards. MAGNeT is configured to record only the transitions between protocol stack layers, not the optional information about the packets and the protocol state. The default 256 KB kernel buffer is also used to store event records.

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

[1] 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 the performance of MAGNeT with `tcpdump` as a commonly-available monitoring tool.

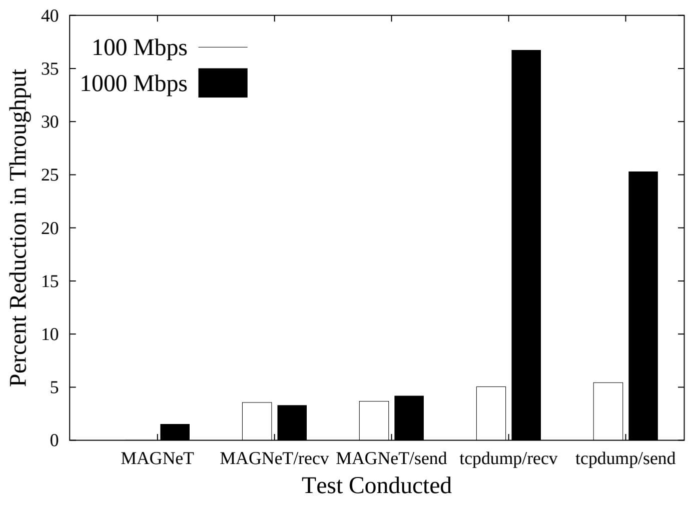

Figure 3. Percent Reduction in Network Throughput

4.2. Network Throughput

The kernel-resident portion of MAGNeT executes whether information is being saved to disk or not. The first data point in Figure 3, labeled "MAGNeT," shows virtually no penalty when data is not being saved to disk. The next two data points show MAGNeT incurs less than a 5% reduction in network throughput, *for a fully saturated network*, 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 100 Mbps networks, the penalty increases to 25%-35% of total throughput at 1000 Mbps. Thus, MAGNeT scales better than `tcpdump` with increasing link speeds.

4.3. Event Loss

Analysis of the MAGNeT-collected data for 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 100 Mbps trials, while for the 1000 Mbps tests the loss rate approached 15%. These losses are due to the 256 KB buffer in the kernel filling before `magnet-read` is able to drain it.

By comparison, loss rates for `tcpdump` are significantly higher — approximately 15% on a saturated 100 Mbps network under the conditions of our tests.

As noted in [IC3N2001], our implementation provides two methods for reducing the event loss rate: (1) increase the kernel buffer size or (2)

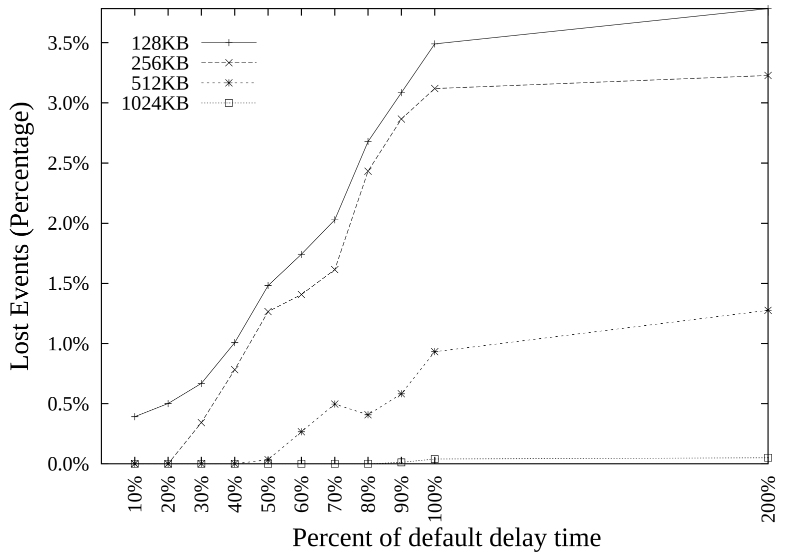

Figure 4. MAGNeT's Event-Loss Rate, 100 Mbs Ethernet

reduce the time `magnet-read` waits before draining the kernel buffer. Figure 4 shows the effect of these parameters on event loss rate for the 100 Mbps saturated network tests.

Increasing the kernel buffer size dramatically reduces MAGNeT's event loss rate, down to virtually no lost events under any network load with a 1 MB 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 256 KB 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` waits before draining the kernel buffer. Shorter delays cause the buffer to be drained more frequently, thus reducing the chance of lost events. However, shorter delays create more work (in terms of CPU usage and, possibly, disk write activity), and thus may interfere with the system's normal use.

Figure 5 shows the increase in average CPU utilization for different delays 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.) As the results show, CPU utilization is relatively insensitive to the range of kernel buffer sizes tested but it is sensitive to changes in delay.

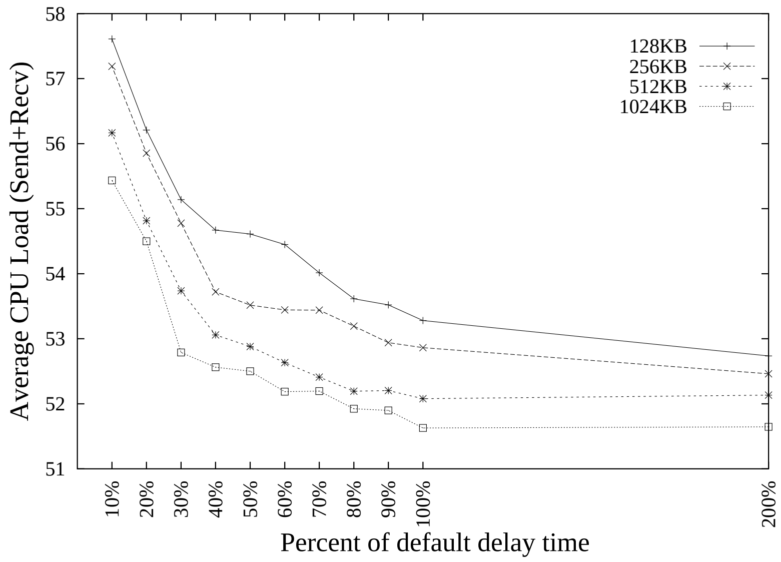

Figure 5. MAGNeT's Average CPU Utilization, 100 Mbs Ethernet

4.4. 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 capture 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 100 Mbps Ethernet an "empty" TCP packet will arrive no faster than $3.2\,\mu$sec. Our tests on a fully saturated network indicate that `magnet_add()` consumes 556 cycles, on average, while recording a packet, while `magnet-read` requires 425 cycles. Thus, in the worst case, MAGNeT takes $2.4\,\mu$sec to monitor a single network packet on our 400-MHz machines. 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.

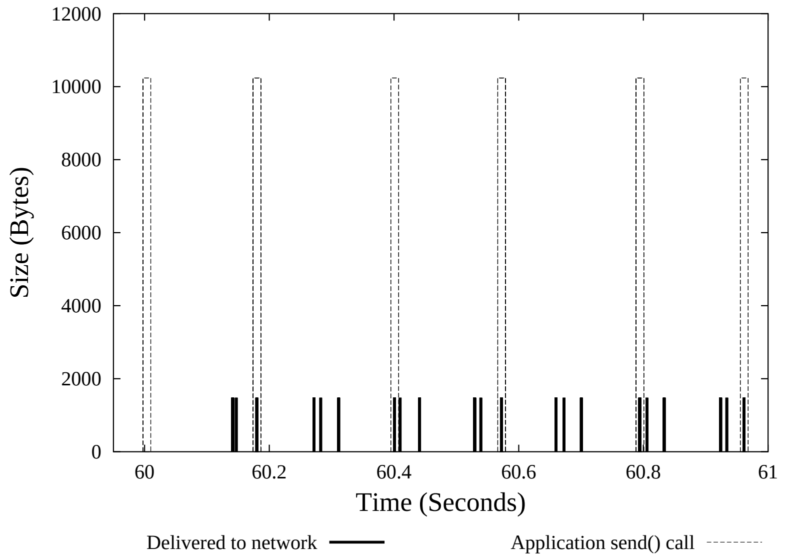

Figure 6. MAGNeT FTP trace

5. Implications and Applications

We have shown that the MAGNeT toolkit provides a transparent method of gathering application traffic data on individual machines. Thus 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.

5.1. 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 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 6.

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

5.2. 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.

5.3. Network-Aware Application Development

As discussed in Section 2, 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.

6. 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.

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.

7. 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.

acknowledgements

This work was funded in part by the U.S. Department of Energy's Laboratory-Directed Research and Development and by the U.S. Department of Energy's Office of Science.

Availability

The MAGNeT toolkit containing 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. 'Accelerated Strategic Computing Initiative'. `http://www.asci.doe.gov/index.htm`.
2. 'Distributed Systems Department'. `http://grid.lbl.gov`.
3. 'Grid Computing Info Centre'. `http://www.gridcomputing.com`.

4. ‘Netperf’. `http://www.netperf.org`.
5. ‘`tcpdump`’. `http://www.tcpdump.org`.
6. ‘The Beowulf Project’. `http://www.beowulf.org`.
7. ‘The Internet Traffic Archive’. `http://ita.ee.lbl.gov/html/traces.html`.
8. Bolliger, J. and R. Gross: 2001, ‘Bandwidth Monitoring for Network-Aware Applications’. *Proc. of the 10th Annual Int’l Symposium on High Performance Distributed Computing.*
9. CAIDA, ‘CoralReef Software Suite’. `http://www.caida.org/tools/measurement/coralreef`.
10. *et al.*, B. W. T.: 1998, ‘The NetLogger Methodology for High Performance Distributed Systems Performance Analysis’. In: *Proceedings of IEEE the High Performance Distributed Computing Conference (HPDC-7).*
11. Feng, W., J. Hay, and M. Gardner: 2001, ‘MAGNeT: Monitor for Application-Generated Network Traffic’. In: *Proceedings of the 10th International Conference on Computer Communication and Networking (IC3N’01).*
12. Feng, W. and P. Tinnakornsrisuphap: 2000a, ‘The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems’. In: *Proc. of the Int’l Conf. on Parallel Processing.*
13. Feng, W. and P. Tinnakornsrisuphap: 2000b, ‘The Failure of TCP in High-Performance Computational Grids’. In: *Proc. of SC 2000: High-Performance Networking and Computing Conf.*
14. Fisk, M. and W. Feng: 2001, ‘Dynamic Right-Sizing: TCP Flow-Control Adaptation’. In: *Proceedings of the 14th Annual ACM/IEEE SC2001 Conference.*
15. Hay, J., W. Feng, and M. Gardner: 2001, ‘Capturing Network Traffic with a MAGNeT’. In: *Proceedings of the 5th Annual Linux Showcase and Conference (ALS’01).*
16. Kato, A., J. Murai, and S. Katsuno, ‘An Internet Traffic Data Repository: The Architecture and the Design Policy’. In: *INET’99 Proceedings.*
17. Semke, J.: 2000, ‘PSC TCP Kernel Monitor’. Technical Report CMU-PSC-TR-2000-0001, PSC/CMU.
18. Tinnakornsrisuphap, P., W. Feng, and I. Philp: 2000, ‘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.*
19. Weigle, E. and W. Feng: 2001, ‘A Case for TCP Vegas in High-Performance Computational Grids’. *Proc. of the 10th Annual Int’l Symposium on High Performance Distributed Computing.*