Integrated Network Performance Diagnostics

Srikanth Kandula Anees Shaikh Erich Nahum

Dept. of Computer Science
University of Illinois
Urbana, IL 61801

Network Services and Software
IBM T.J. Watson Research Center
Hawthorne, NY 10532

Problem Scenario

One of the most challenging parts of running a network-based service is monitoring and managing performance. End-to-end performance may be influenced by numerous factors, and problems are not easily attributable to their correct source [1]. When a performance problem arises, the service provider or customer has available a number of tools to aid in diagnosing performance issues, each of which test different aspects of the network service.

Client-perceived performance, for example, requires the use of application-layer tests to measure the application response time. Such tests can be conducted for Web-based applications using tools like *PageDetailer* [2] or services provided by companies such as Keynote Systems [3]. Low-level network properties such as connectivity and latency may be tested using tools like `traceroute` or `ping`. Network protocol behavior can be examined in detail using packet sniffing tools, as exemplified by `tcpdump`. Loss rate and bandwidth along paths can be measured using tools such as *sting* [4] and *pathrate* [5], respectively. While these low-level tools are useful for detecting relatively simple conditions, such as a server being unavailable, or the absence of a network path to the destination, it is not straightforward to directly relate the information from such tools to application behavior.

A common performance management approach involves monitoring the application (i.e., user-perceived) performance at a relatively coarse level, and then conducting further, more detailed, tests when a potential problem is detected. As discussed above, such an approach requires the use of multiple techniques and tools, operating at multiple levels. Hence, the problem remains of how to correlate the information to construct a more complete view of low-level network events (e.g., packet retransmission) and the application actions that triggered them (e.g., HTTP request).

In this abstract, we focus on the problem of integrating performance-related information from multiple network layers for the purpose of network performance diagnosis. Our approach is to provide application-level measurement tools with direct access to pertinent information from lower layers. This enables a diagnostician to detect specific packet-level events in various contexts of the application in an automated way, without having to unify multiple traces. Our initial objective is to embody this approach in a measurement and monitoring tool for identifying the causes of performance problems in Web-based applications. Below we describe our approach in further detail, with a discussion of our initial design and ongoing implementation. We also list some of the advantages and limitations of this scheme.

Solution Overview

To achieve our goal it is necessary to provide a measurement application with access to fine-grained information generated by the end host operating system's networking stack. One technique is to export such information through kernel interfaces such as socket options. For example, the Linux 2.4 kernel has added a `TCP_INFO` socket option that exports TCP connection-specific information such as the sender congestion window, number of packet losses, and round-trip time estimate. Another option is to use the statistics exported by the kernel to the `/proc` filesystem, though it contains limited information and is likely to require modification.

Since our intention is to develop a tool that can be used easily in a variety of settings, we wish to avoid reliance on a particular kernel configuration, or the presence of specific kernel support. In many cases it is not possible or desirable to to use tools that require changes to an underlying production platform. Hence, our approach is to make the bulk of the kernel networking stack available at the user level in the form of a library, with instrumentation to deliver notifications about events of interest to the measurement application.

Our architecture is shown in Figure 1. The measurement application is linked with the user-level TCP/IP library, and communicates using raw IP sockets. Although the library performs some input IP processing, outbound packet routing is handled by the kernel IP implementation. Note that, as with other user-level networking implemen-

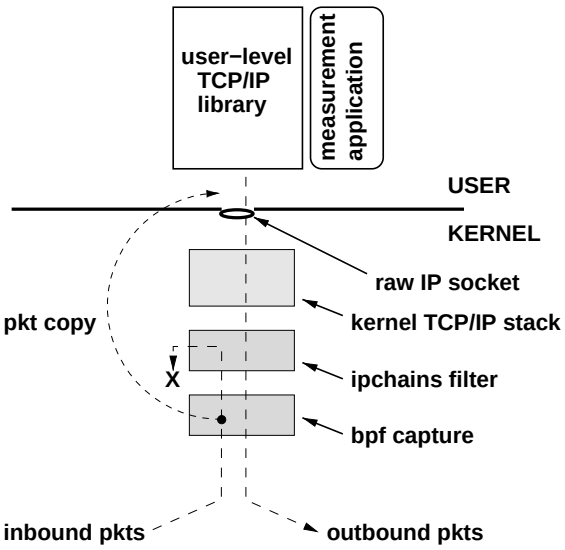

Fig. 1. Architecture diagram

tations, we must arrange for incoming packets to bypass the native kernel stack [4]. The kernel will not be able to find the corresponding connection state when demultiplexing since the state is kept in the user-level library. As shown in the figure, we achieve this with a combination of the Berkeley packet capture (BPF) facility and native firewall/filtering support (e.g., `ipchains` on Linux systems). The BPF filter is configured to deliver a copy of specified packets to the measurement application, and the firewall drops the same packets before they are processed by the kernel stack. Outbound packets are emitted via the raw socket and pass through the filters unmolested.

In order to effectively aid in problem diagnosis, the measurement application must have access to lower-level events of interest. In our initial architecture, the network library exports an API through which the measurement application can express its interest in events from a predefined set. When one of the specified events occurs, the library records the event and makes it available to the application for examination or logging. For example, if a measurement application is interested in network latency effects during a certain phase of application execution, it can request notifications of significant changes in the TCP RTT estimate which may indicate increased network congestion or path fluctuations.

Advantages and Limitations

This approach has several key benefits over other, non-integrated, diagnostic techniques. As we have access to the main protocols in the TCP/IP protocol suite, the measurement application can receive extensive diagnosis information. Changes in TCP state variables, bursty vs. isolated packet loss events, and information about the source and type of received ICMP messages, are examples of the type of additional details that are not otherwise available to an application-level measurement tool. As described earlier, this technique obviates the need to combine multiple sources of measurement or diagnosis data from the application and the packet level to account for application dependencies. Also, note that the amount of trace data generated can be flexibly adjusted, and can be different during different execution phases of the measurement application. The user-level networking stack handles only traffic to or from the measurement application, and thus does not introduce much overhead on other application traffic at the host. Finally, our requirement of avoiding kernel dependencies makes the tool relatively easy to move to different platforms.

There are, however, several limitations to the integrated approach. First, the application using the facility must be instrumented to use the event and user-level networking APIs. We envision, though, that the user-level stack will be integrated with applications specially designed for performance diagnosis, rather than with production applications which may be more difficult to modify. Second, since the user-level networking stack is just a regular application, it is subject to process scheduling that may cause significant jitter. This may be mitigated by giving the measurement application the highest priority, but we have not yet quantified the delay jitter. Third, the tool uses raw IP sockets, and thus requires superuser privileges to run.

Implementation and Status

One of the first tasks in implementing our approach was to identify a suitable user-level networking implementation. Several user-level protocol stack implementations have been developed for use in various contexts and on different platforms [6], [7], [8], [9], [10], [11]. We chose to use the Arsenic implementation by Pratt and Fraser [10], primarily because i) Arsenic is based on the Linux TCP/IP protocol implementation, and ii) it uses most of the protocol processing code from the kernel unmodified. Thus, it is representative of the networking behavior on the platform we are most interested in. However, Arsenic is also tightly coupled to specific features of the Alteon ACEnic gigabit Ethernet adaptor, and requires the use of loadable kernel modules. We therefore had to modify the interface between the library and the kernel significantly to remove these dependencies. These changes have been completed, resulting in a user-level Linux networking suite which does not have any dependency on specific NIC hardware or Linux kernel version.

We are currently instrumenting the networking library to export a sufficiently detailed set of events to the applications. In some cases the application needs only to know that an event occurred (e.g., packet retransmission, or receipt of a zero window advertisement). We are also considering allowing the application to receive notifications when a certain event is repeated, or persistent. For example, the measurement application may be interested in knowing about bursty or persistent packet loss, in contrast to individual loss events. Such notifications require the library to maintain additional state for the connection.

wwwprobe – a first application

As part of related work on Web performance measurement, we developed *wwwprobe*, an multi-threaded application-layer tool to measure client-perceived Web response time. *wwwprobe* acts as a mini-browser, downloading web pages, along with their embedded objects. It measures the time taken for each step in an HTTP transaction, including DNS name resolution, connection establishment, server response time, and download time for an entire page as well as for individual objects. These measurements are taken at the socket API level, by timing such system calls as `connect`, `read`, and `gethostbyname`. Currently *wwwprobe* supports only HTTP/1.0. We are planning to use *wwwprobe* as a proof-of-concept measurement application to explore the benefits of the integrated approach for enhanced diagnosis of Web performance.

In our ongoing work, we are completing instrumentation of the *wwwprobe* application. We are also conducting experiments in a controlled lab setting to introduce network events that exercise the notification mechanisms to *wwwprobe* from the user-level networking library. In addition, we are planning to deploy the *wwwprobe*-based tool in the wide-area to monitor and diagnose the performance of production Web applications.

REFERENCES

[1] Paul Barford and Mark Crovella, "Critical path analysis of TCP transactions," in *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[2] IBM Corporation, "Page Detailer," Distributed with IBM WebSphere Studio, `http://www.research.ibm.com/pagedetailer/`, June 2000.

[3] Keynote Systems, Inc., ," `http://www.keynote.com`.

[4] Stefan Savage, "Sting: a tcp-based network measurement tool," in *Proceedings of USENIX Symposium on Internet Technologies and Systems*, October 1999.

[5] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore, "Packet dispersion techniques and capacity estimation," available at `http://www.cis.udel.edu/~dovrolis/Papers/ton_dispersion.ps`.

[6] Aled Edwards and Steve Muir, "Experiences implementing a high-performance TCP in user-space," in *Proceedings of ACM SIGCOMM*, August 1995.

[7] Chandramohan A. Thekkath, Thu D. Nguyen, Evelyn Moy, and Edward D. Lazowska, "Implementing network protocols at user level," *IEEE/ACM Transactions on Networking*, vol. 1, no. 5, pp. 554–565, 1993.

[8] Torsten Braun, Cristophe Diot, Anna Hoglander, and Vincent Roca, "An experimental user level implementation of TCP," Tech. Rep. RR-2650, INRIA Sophia Antipolis, September 1995.

[9] Peter A. Dinda, "The Minet TCP/IP stack," Tech. Rep. NWU-CS-02-08, Department of Computer Science, Northwestern University, January 2002.

[10] Ian Pratt and Keir Fraser, "Arsenic: A user-accessible gigabit ethernet interface," in *Proceedings of IEEE INFOCOM*, 2001.

[11] David Ely, Stefan Savage, and David Wetherall, "Alpine: A user-level infrastructure for network protocol development," in *Proceedings of USENIX Symposium on Internet Technologies and Systems*, March 2001.