Networking Version 2.0 Benchmarks

William Bryant – *Engineering Manager, Sun Microsystems, Inc. and Chairman, EEMBC Networking Subcommittee*
Rick Foos – *Director of Engineering, EEMBC Certification Labs (ECL), LLC*
Markus Levy – *President and CEO, EEMBC*
Alan R. Weiss – *Chairman and CEO, EEMBC Certification Labs (ECL), LLC*

August 5, 2004

Introduction

Almost all networking equipment designs rely on embedded microprocessors to perform many of the tasks needed to make sure packets make it intact across the network. Hardwired logic is often too inflexible to deal with many of the error conditions and interactions that can occur with Internet Protocol packets. Although processors deliver the flexibility, performance is a key issue. There is continuing demand for faster processors to support next-generation routers and switches as well as protocol coprocessors for clients and servers. At the same time, equipment builders face pressure from customers to lower the cost of these hardware pieces.

To select the best processor for the job – one that couples high speed with low cost – the network-equipment designer needs hard information that shows how each candidate processor will perform. It is almost never practical to run production code on each candidate processor to see whether it offers the best price/performance ratio or support for the target application. So, benchmarks play a vital role in helping equipment designers in the network industry select the processors that can best support their applications. It is no simple task, however, to create benchmarks that provide an accurate reflection of the real-world performance of a given processor.

Benchmark creators must structure benchmarks carefully to ensure that the algorithms result in the maximum stress being applied to candidate processors. This approach helps expose the strengths and weaknesses of individual processors on different types of code. Even with a careful benchmark creation process, however, architectural changes and manufacturing process advances have the potential to eliminate consistency between benchmark scores and real-world performance. For example, increases in transistor cell density made possible by successive generations of silicon process technology have allowed processor designers to provide massive increases in on-chip cache memory. That makes it possible for some processors to run entire

benchmarks from cache instead of being forced to fetch data from main memory. This provides a benchmark score that does not tally well with real-world behavior.

For this reason, EEMBC has updated its networking benchmark suite with benchmarks that use novel data generation techniques to ensure that the processors under test will handle realistically large data sets and an increased number of transactions. As a result, processors with large caches must compete on a level playing field with those that have traded cache size for networking optimizations or have pipelines that have been finely tuned for handling the types of code that are commonly used in networking applications.

Expanding on the EEMBC Networking Version 1.0 benchmark suite, Version 2.0 combines new benchmarks with enhanced versions of benchmarks carried over from Version 1.0. EEMBC has updated the existing benchmarks to put more stress on processors with large caches and more efficient, faster cores. For benchmarking purposes, the Networking Version 2.0 suite replaces Networking Version 1.0, and the benchmark scores from these two suites are not comparable to one another (check the EEMBC website for more details).

By contrast, some competing benchmarks emphasize the raw transfer rate of large blocks of data between two users. This approach does not provide a balanced test for networking-oriented processors, as some will have optimizations for bulk transfers, flattering them in relation to their real-world performance, where there will be many different users with differing traffic types. EEMBC has worked to avoid this mismatch. For example, when the Networking benchmarks are initialized, the private data sets that are created can model many more users than would be possible on a typical local area network (LAN), ensuring that the workload is representative of high load conditions.

One further change that improves correlation with real-world performance is to ensure that the timing measurements reflect packets coming from a network connection, rather than from a buffer. To do this, EEMBC removed activities such as buffer-to-buffer copying from the timed portion of benchmark code. Those buffer-to-buffer copies would not happen in the real-world environment and, if timed during benchmarking, could skew the results towards processors optimized for high-speed data transfers. However, to maintain portability for the benchmark code, the transfers remain in the code.

Ensuring Benchmark Portability

EEMBC designed the Networking Version 2.0 benchmarks with portability in mind. EEMBC's benchmarks must run on 32- and 64-bit architectures, which may have big- or little-endian byte ordering. These considerations are central to the portability of many networking protocols. The use of bit-level masks and C data structures, such as unions, demands careful attention to register widths and byte ordering. The situation is complicated by the fact that networking source code

tends to employ highly optimized system-level functions and constructs that are on the edge of ANSI C compliance in order to obtain the maximum speed possible. EEMBC developed a code harness for the benchmarks that helps ensure that these issues do not lead to compilation problems. To ensure that, once compiled, the benchmarks run correctly, each benchmark has built-in error checks. These use checksums and other mechanisms to detect whether a benchmark contains an error that causes it to not complete its work or make errors when processing packets.

One further change in the move to Version 2 is a reflection of the differing needs of network equipment. Some of the benchmarks are designed to reflect the performance of client and server systems, while others are representations of functions predominantly carried out in infrastructure equipment. Typically, routers have no need to process information in the TCP layer, and results for tests associated with TCP performance will skew the results for those OEMs looking to select the best processor for work at the lower IP layer. Therefore, the Networking Version 2.0 benchmarks produce two new aggregate "mark" scores: the TCPmark and the IPmark. The IPmark is intended for developers of infrastructure equipment, while the TCPmark, which includes the TCP benchmark, focuses on client- and server-based network hardware.

IP Packet Processing

A fundamental part of a network router's workload is to correctly process IP packets. These packets can be out of order, have errors, and demand router actions, rather than just being forwarded to the next node in the network.

Whether a packet is to be forwarded to another router or processed and sent to a local machine, the first step for all packets is to validate the IP header information. This is why a key part of the Version 2 benchmark set is the Packet Check benchmark. This benchmark models a significant subset of the IP header validation work that is specified in the RFC1812 standard that defines the requirements for packet checks carried out by IP routers.

In operation, the Packet Check benchmark simulates a router with four network interfaces. During initialization, the code creates a buffer and generates the data that will represent a block of test packets. The initialization code creates a series of packets that indicate the IP version, checksum, and length in the header. During this process, the code introduces errors into some packet headers and produces a count of how many packets have deliberate errors. This count is checked at the end to ensure that the benchmark code has executed correctly. Using padding between packets, the benchmark specification allows packets to be aligned on the best natural word boundary of the microprocessor.

To maintain as much realism as possible, the benchmark has been designed to emulate the way in which actual systems process packet

headers. For example, the benchmark uses a scheme where descriptors are separated from the packet headers. This allows the descriptors to be managed as a linked list with each descriptor pointing to an individual IP packet header. This demonstrates the processor's ability to work with the branch- and pointer-intensive code found in typical packet-switch code bases. Further, by focusing on error handling rather than raw packet throughput, the benchmark provides a more realistic check on processor behavior rather than just cache-to-memory speed.

Routing

Each packet on a large IP network passes through a router, which determines whether it should be passed to another router closer to the final destination or if the packet should be processed and forwarded to a local machine. For the forwarding function, the router must determine which other routers are available for forwarding, find the shortest path to each, and detect configuration changes in the network of routers.

Open Shortest Path First (OSPF) is the most popular Internet routing protocol used to determine the correct route for packets within IP networks. The OSPF benchmark uses this protocol as a representation of a processor's ability to handle routing problems. The OSPF benchmark implements Edsger Dijkstra's shortest-path-first algorithm. Shortest-path-first algorithms are processor and memory intensive, making them good candidates for stressing high-performance processors with large cache subsystems.

The Dijkstra algorithm finds the shortest, or least-cost, path from a specific router to all other routers that the source knows about. It builds a table of nodes, where each node is a router. Each node has one or more arcs, where each arc is a directed, one-way link to another node. These arcs represent links between routers. Each arc has a cost value that represents the value of the link. The lower the cost number, the more desirable it is to use the link.

The OSPF benchmark begins by creating a set of nodes and arcs that are each connected with a cost computed by the data-generation routine. During execution, to emulate the dynamic behavior of OSPF in a real-world environment, the code re-initializes the table of arcs and nodes after each benchmark run. The benchmark performs a series of calculations using the OSPF algorithm to determine the destination port for each given route.

Once the route tables are built using protocols such as OSPF, efficient route lookups are fundamental to the performance of network routers. The Route Lookup benchmark, the second of the core routing-oriented benchmarks, is a distillation of the fundamental task of IP routers, which is to receive and forward IP packets based on the information found in lookup tables.

The Route Lookup benchmark uses a mechanism commonly applied to commercial network routers. It employs a data structure known as the

Patricia Tree. The Patricia Tree is a compact binary tree that allows fast and efficient searches with long or unbounded length strings and is often used in database searches as well as routing lookups. The benchmark monitors the processor's ability to check the tree for the presence of a valid route and walk through the tree to find the destination node to which to forward the packet.

Network Boundary Processing

The Internet is not a homogeneous network. Organizations will have their own network setups that may be incompatible with those used by their Internet service providers. A common requirement for network equipment is to convert packets as they pass from an internal network to the wider Internet. Two of the EEMBC benchmarks address these applications, which are typically processor- and memory-intensive.

An increasingly important function for IP routers that sit on the boundary between an organization's internal network and the Internet is Network Address Translation (NAT). NAT is an important function because it provides a method to work around the limited number of IP addresses and ports on the Internet. Additionally, NAT is normally required when a network's internal IP addresses cannot be used outside the network, either because they are not globally unique or because of privacy reasons.

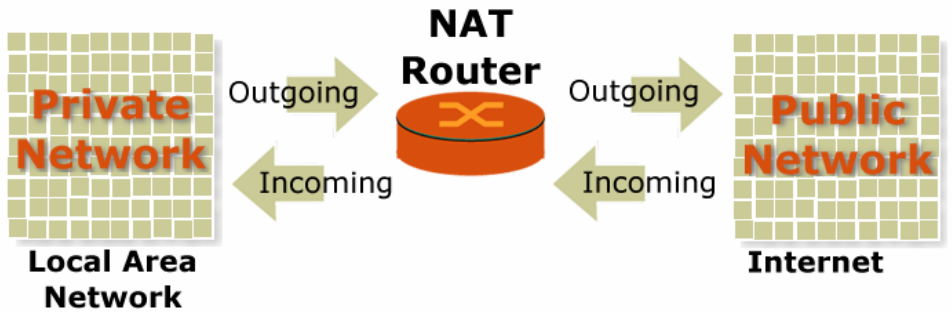

Figure 1. The NAT router remaps incoming and outgoing IP addresses.

There are two types of NAT, dynamic and static. Dynamic NAT allows a large number of clients connected in a LAN to access the Internet using a few public addresses available to the router. Static NAT provides a means for servers on a LAN to be accessed by clients on the Internet.

With Dynamic NAT, a client computer sends packets using its local IP address and port number to the destination port of "a_server.com". When this packet arrives at the NAT-enabled router, the router rewrites the packet, replacing the local IP and local port with its public IP address and an available port. The router saves this translation information in a table. When a packet arrives from "a_server.com", the router uses the translation information to again re-write the packet, restoring the local IP address and the local port number. In both the send and receive directions, checksums are also updated to reflect the

change in address.

With Static NAT, a local server is set up to provide service on a port: for example, a Web server on TCP port 80. Next, a Static NAT entry is defined in the router that maps incoming traffic from the public IP address to the local IP address. When a packet arriving from the public Internet does not match an IP and port in the Dynamic NAT table, the router rewrites the packet with the local IP address in the Static NAT entry.

Dynamic NAT routing represents the bulk of the workload, as all outgoing packets are processed through it, plus it has the additional complexity of port assignment on the incoming and outgoing packets to preserve connections between clients and servers.

The NAT benchmark creates a series of packets during initialization with various source addresses, destination addresses, and random packet sizes. Each packet is then wrapped with IP header information. Status information is included and the packets are assembled into a list for processing. Finally, the NAT rules are added to the table.

The benchmark then begins processing and rewriting the IP addresses and port numbers of packets based on the pre-defined NAT rules. Each rewritten packet will have a modified source IP address and source port chosen from the available ports of each IP address available to the router. In this way, the NAT benchmark simulates an important part of network processing for many router designs, performing many of the functions of a commercial NAT implementation.

As Internet traffic passes from one part of the network to another, the packets themselves may need to be altered. Each network technology has a maximum frame size that defines the Maximum Transfer Unit (MTU), or maximum packet size, that can be carried over the network. When an IP packet is too large to fit within the MTU of the egress interface, it can no longer be transmitted as a single frame. Rather, the IP packet must be split up and transmitted in multiple frames. At the other end of the link, fragmented packets need to be reassembled. This process can place significant resource requirements on systems, making processor performance on this type of workload a key consideration. Reassembly represents the largest workload, because the router must also deal with packets that arrive out of order and with random data sizes.

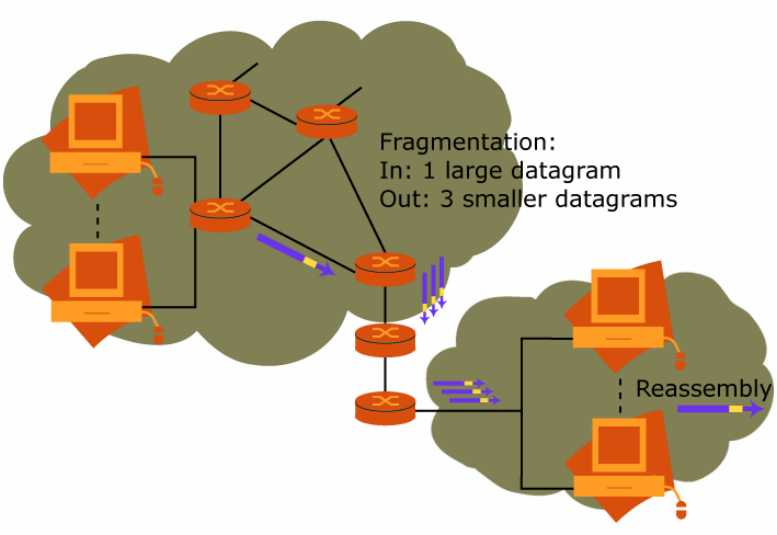

Figure 2. IP fragmentation and reassembly splits and recombines packets into smaller pieces to accommodate the maximum transfer unit of the network.

The IP Reassembly benchmark takes the asymmetric nature of fragmentation and reassembly into account and makes intensive use of out-of-order delivery and random source-packet sizes to stress the processor's ability to perform reassembly. During benchmark initialization, the code generates a list of packet fragments based on source packets with random lengths. Buffers are created to hold the fragmented packets and used to deliver the fragments so that they do not always arrive in order. The random arrival rate is set during data creation. The reassembly process is destructive: for memory efficiency, the networking code performs reassembly in the same memory space as the fragmented packets, simulating the behavior of most network routers.

Quality of Service

The data carried over the Internet has evolved from text and files—where timing and order of packet arrival are irrelevant—to voice, video and multimedia, where timing and order are critical. Applications that process real-time data streams are less tolerant of out-of-order packets and long delays between packet arrivals. Normally, when this occurs, a connection is dropped and requests to resend the data are initiated. This is fine for normal data but leads to breaks in audio and video playback.

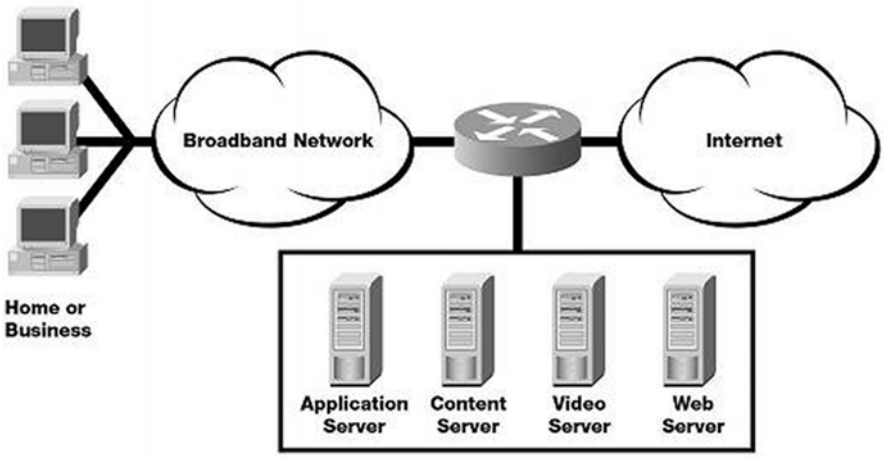

Figure 3. Many different types of data are carried over the Internet. The network must manage the bandwidth to determine priority of delivery.

As real-time traffic increases, the errors and resend requests can exceed the overall amount of data to be transferred, resulting in error-clogged networks and dropped connections for users. Quality of Service (QoS) processing addresses this problem by providing the ability to measure data-transfer rates, by giving the client guaranteed data transfer and error rates that are suitable to support a deterministic application. This QoS guarantee significantly reduces the network loading due to errors and retransmission and allows these new forms of data to flow over the Internet.

The QoS benchmark simulates the processing undertaken by bandwidth management software used to "shape" traffic flows to meet QoS requirements. Based on predefined rules, the system paces the delivery of the system to the desired speed. This shaping is achieved via the use of a variant of the Weighted Fair Queuing (WFQ) algorithm. Random Early Detection (RED) queue management is also supported to provide flow control.

The data generation stage of the benchmark constructs a bank of packets varying the data size, source, and destination addresses. Next, a set of rules that determine routing that support defined data rates are entered into the QoS tables.

The timed portion of the benchmark begins by processing packets against the rule set, which determines the routing and addressing needed to best preserve the QoS for each stream of packets. As the number of packets in the system increases, port diversions begin to occur to maintain the QoS, and queues are delayed to wait for available pipes as determined by the algorithm. As a result, the benchmark demonstrates close alignment with the behavior of real-world QoS

algorithms.

Client and Server Processing

The benchmarks covered so far have concentrated on the performance of processors dealing with traffic at the IP layer. Clients and servers need to process higher-level protocols, such as the Transmission Control Protocol (TCP). Forming the transport layer protocol used by Internet applications such as Telnet, the File Transfer Protocol (FTP), and the HyperText Transfer Protocol (HTTP), TCP provides a link that looks, to the application, as though it is a direct connection. Because TCP uses the services of IP to deliver packets, and IP does not care about the order in which complete IP packets are delivered, TCP is designed to handle packet re-ordering and re-sending for situations where a router may have dropped packets to be able to meet its overall service level requirements.

Different upper-level protocols stress TCP-handling hardware in different ways. For example, Telnet consists of short, small bursts of data in small packets that result from a user typing commands and receiving results. On the other hand, FTP consists of large amounts of data in large packets moving in one direction. HTTP is somewhere in the middle with bursts of files in one direction intermixed with control and handshaking traffic in both directions. This makes the consideration of traffic type essential when analyzing the performance of a processor that will process TCP-layer traffic.

EEMBC's Networking Version 2.0 benchmarks include a TCP benchmark that accounts for the different behavior of TCP-based protocols by measuring the performance of a processor that handles a workload derived from several application models. The TCP benchmark has three components to reflect performance in three different network scenarios. The first is a Gigabit Ethernet involving large packet transfers to represent the likely workload of Internet backbone equipment. The second assumes a standard Ethernet network for packet delivery and concentrates on large transfers using protocols such as FTP. The last uses a standard Ethernet network model for the relay of mixed traffic types, including Telnet, FTP, and HTTP.

The benchmark generates data based on the applications that need to be modeled, with packet queues built to simulate both client and server traffic. The main part of the benchmark involves processing all of the packet queues through a server task, network channel, and client task. These simulate the data transfers through the connection to provide a realistic view of how the processor will cope with various forms of TCP-based traffic.

Summary

The EEMBC Networking Version 2.0 benchmarks set a new standard for the measurement of processors by the combination of client-server

framework, multi-user data generation, and real-world application kernels defined by the industry. With these new benchmarks, network equipment OEMs can be confident that test results will reflect the performance that can be obtained from processors in production-quality hardware.