EEMBC®

An Industry Standard Benchmark Consortium

Networking Version 2.0

Benchmark Name: Transmission Control Protocol (TCP)

Highlights

- **Captures most frequently used and most processing-intensive portion of RFC793 protocol**
- **Simulates TCP traffic characteristics in real networks**
- **De-couples processor speed from any randomness in TCP operation**
- **Uses three different data sets to simulate a variety of workloads**

Application

The ability of an embedded processor to handle Transmission Control Protocol (TCP) layer processing is an important consideration for avoiding bottlenecks in network equipment designs. Unlike ATM and some other network protocols that are mainly processed by network processors, ASICs, or specialized hardware blocks directly attached to general purpose processors, the TCP layer is often processed by the CPUs in general-purpose processors. The interest of benchmarking TCP performance on embedded general-purpose processors has increased with the connection of more and more embedded devices to the network. The flexibility of TCP is such that it is used in wireline and wireless applications.

The ISO reference model is commonly used when discussing protocol layering. This model depicts the TCP layer as sitting on top of the Internet Protocol (IP) layer and under the application layer. The function of IP is to provide a means of transferring TCP segments over inter-connected networks. IP has unique addressing information for each network element, and data communication is based on routing that provides best effort service to TCP and other transmission control layer protocols like UDP.

In contrast to IP, TCP service is a reliable, connection-oriented byte stream service. It typically interfaces with an unreliable network layer protocol. Unlike other connection-oriented protocols that are based on a reliable network layer, TCP has to implement a more complex transmission control scheme to overcome these seemingly contradictory philosophies between protocol layers.

The basic operation of TCP can be broken down into the following six areas:

1. Basic data transfer
2. Reliability
3. Flow control
4. Multiplexing
5. Connections
6. Precedence and security

Application

The core of the TCP protocol is to transfer data between two connection endpoints. Like data processing in most of the network protocols, large data blocks are chopped into optimized sizes (as deemed by TCP) and encapsulated in a TCP segment. Communications in TCP involve both data and control operations. Comparing data processing with other protocols, the biggest difference with TCP is a mandatory checksum across the entire segment. This is because TCP provides reliable communication service on top of an unreliable IP layer. For the same reason, TCP requires fairly complex control and signaling to achieve reliability, efficiency, and connection management. Compared with IP, data operations are simpler but are more expensive in terms of performance. The cost associated with the data block size is linear in most of the cases. (For example, computing IP style checksum and memory copy.) The benchmark captures all the costly data manipulations while some of the complex but rarely used control logic can be omitted.

Benchmark Description

This benchmark implementation captures the most frequently used and processing-intensive portion of the protocol described in RFC793. The benchmark measures the data and buffer management performance, which is common and expensive in TCP implementations. Also, because this benchmark targets embedded general-purpose processors, the execution environment should match code size and memory scale. Typically, execution environments include a reasonably-sized memory and high-performance RTOS with shared kernel and user addressing spaces. The scope of this benchmark does not include measuring overall network performance.

EEMBC's TCP benchmark follows these general requirement guidelines:

Accurate – the benchmark captures all major TCP operations in terms of processing cost

Realistic – the benchmark simulates TCP traffic characteristics in real networks

Deterministic – the benchmark de-couples processor speed from any randomness in TCP operation

Simplistic – the benchmark implementation allows for a simplified TCP implementation with reasonable assumptions

Application protocols that use bulk transfer contribute 90% of the traffic in terms of number of bytes but represent only about half the packets. The TCP benchmark is designed to be flexible enough to capture processor performance for both transfer types.

Benchmark performance metrics include:

1. Complete event-driven TCP state machine, connection management signaling
2. Transient behavior in short TCP conversations
3. Buffer management – Data manipulation in both ingress and egress directions
4. Queue management – Send queue, unacknowledged queues in egress direction
5. Separate re-entrant client-server task with context switching
6. Basic flow control
7. Multiple data stream (phase II)
8. Configurable packet size distribution for different traffic patterns

RFC793 requirements that are not included in benchmark include:

1. Real-time timer related – RTT estimation and update, RTO
2. Exception handling, out-of-order delivery, duplicates and lost packets

TCP behavior varies dramatically between different applications. Packet sizes, conversation length, and queue depth can all affect processing in different ways. To cope with different scenarios, the benchmark is configurable. In addition, the following four standard test cases were designed based on representative statistical data.

Parameters/Tests

The application data block processed between the client and server is constructed as a ring or circular buffer based on the segment size, and number of packets in the workload.

1. Bulk data transfer test uses maximum TCP segment size (which is typically used in FTP data channel)
2. Jumbo test uses MTU sizes, and numbers of packets applicable to a Gigabit Ethernet Backbone.
3. Mixed packet sizes is an average case Standard Ethernet – mixture of activity

Analysis of Computing Resources

Each workload simulates network traffic using the following steps:

1. Initiate server task.
2. Insert network channel effect
3. Initiate client task
4. Insert network channel operations of client.

This workload is repeated until all client connections are closed.