Case Study: Using System Tracing to Improve Packet Forwarding Performance

Sebastien Marineau-Mes, Senior Networking Architect,
QNX Software Systems
sebastien@qnx.com
Abstract
Symmetric multiprocessing (SMP) can offer enormous scaling benefits, particularly when applied to networking applications such as IP and MPLS route calculation, Layer 3 packet forwarding, storage caching, and call-processing encryption. However, the very complexity of these systems can mask common problems such as resource contention, which can reduce the performance gains expected from SMP implementations.

To visualize what’s going on in a complex embedded system without altering the system’s behavior, developers can employ nonintrusive trace analysis techniques. Unlike conventional debug methods that rely on breakpoints and other overhead-intensive techniques, trace analysis uses fast, selective logging of system events, including messages, kernel calls, state changes, and interrupts. User-written code doesn’t have to be modified, since event logging can be performed by an instrumented kernel.

This case study demonstrates the use of a trace analysis tool, the system profiler in the QNX® Momentics® development suite, to uncover hidden bottlenecks in an SMP system. For the case, we’ll review results from a benchmark study that measures Layer 3 packet forwarding, using a test system based on the QNX Neutrino® RTOS and Broadcom’s SiByte BCM1250 dual-processor SOC.
**Test Setup**

Measurement of Layer 3 packet forwarding performance is a common benchmark for routers, and as a result, the testing methodology has been standardized by the Internet Engineering Task Force (IETF) in RFC2544. For our case study, we followed this methodology, using an Adtech AX/4000 Broadband Test System that included two Gigabit Ethernet port interfaces and two 1 Gbps mAX IP and mAX IPex routing generator/analyzers. We connected the Ethernet ports to the device under test (DUT), which was configured as a router. See Figure 1.

The DUT consisted of the following:

- **Hardware** — The DUT used the Broadcom BCM91250A “SWARM” evaluation board, based on the SiByte BCM1250 dual-processor SOC (clocked at 600MHz for this study). The board includes two Gigabit Ethernet ports; both were used as the network interfaces in the packet forwarding tests.

- **Software** — The DUT software consisted of the QNX Neutrino RTOS v6.2.1B, including a TCP/IP stack, supporting software utilities, and an instrumented SMP kernel. QNX Neutrino’s SWARM board support package (BSP) provided boot support, including UART and PCI drivers, as well as Ethernet network drivers for the board’s Gigabit ports.

The RFC2544 testing methodology includes the measurement of throughput, latency, frame loss, and back-to-back packet handling. We focused on the first test, throughput, which RFC2544 defines as the highest rate at which packets of a given size can be sent to a DUT and forwarded back, without loss of a single frame.

The AX/4000 traffic generator/analyzers implement this benchmark by generating known packet streams through each interface and analyzing the forwarded packets to measure frame loss. The AX/4000 software automates measurement of throughput by iterating through different packet rates (via a binary search...
algorithm) to determine the packet rate at which no frames are dropped. The results are stored, and the test is repeated for all packet sizes.

We configured the two ports on the DUT to reside on different subnets (10.x.x.x and 11.x.x.x), and set up static route entries to allow forwarding between these two interfaces. We then set the AX/4000 to generate two traffic streams, in a fully meshed configuration; that is, traffic would be sent to both ports and the throughput would be the highest packet rate that both ports could sustain.

On the DUT, we used the following commands to start the QNX network manager (io-net), network driver, and TCP/IP stack:

```
Io-net -dbcm1250 receive=256,transmit=1024 -ptcpip-v6 fastforward -s &
  # Set number of RX and TX descriptor
  # Enable fastforward
  # Enable static io-net configuration
Ifconfig en0 10.1.1.1
Ifconfig en1 11.1.1.1
Ifconfig +ip4csun en0
  # Turn on HW checksum on port 0
Ifconfig +ip4csun en1
  # Turn on HW checksum on port 1
```

---

**Packet Forwarding Results with SMP**

To get the baseline measurement for our optimization study, we performed a packet transfer test, with both SMP processors enabled. See results below in Table 1.

<table>
<thead>
<tr>
<th>Packet Size</th>
<th>Packet Rate (PPS per port)</th>
<th>Total Throughput (Mbit/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>323429</td>
<td>413</td>
</tr>
<tr>
<td>128</td>
<td>315630</td>
<td>808</td>
</tr>
<tr>
<td>256</td>
<td>312545</td>
<td>1600</td>
</tr>
<tr>
<td>512</td>
<td>195312</td>
<td>2000</td>
</tr>
<tr>
<td>768</td>
<td>130208</td>
<td>2000</td>
</tr>
<tr>
<td>1024</td>
<td>97656</td>
<td>2000</td>
</tr>
<tr>
<td>1280</td>
<td>78125</td>
<td>2000</td>
</tr>
<tr>
<td>1514</td>
<td>65876</td>
<td>2000</td>
</tr>
</tbody>
</table>

*Table 1 — Throughput results for dual-processor case, prior to optimization.*
Figure 2 shows a system profiler trace taken during the benchmark — the running thread on CPU 0 is light blue, the running thread on CPU 1 is magenta. We can see that two driver threads are running in parallel (threads 5 and 6), each driver forwarding packets from a separate interface.

**SMP optimizations for packet forwarding performance**

A common SMP bottleneck occurs when two or more threads contend for the same resource — for instance, two threads that share a data structure whose access is protected by a mutex or spinlock. If the two threads run in parallel on separate processors (SMP) and both access this data structure for significant amounts of time, they may end up spending considerable time contending for the lock, thereby serializing execution. This will diminish the benefits of parallel processing.

Returning to the system profiler trace in Figure 2, we can make some observations. For instance, the trace shows that each driver thread makes several calls to `mutex_lock()` and `mutex_unlock()`, a telltale sign of lock contention. This contention is for the same mutex object, which we can trace back to the mutex protecting the forwarding table in the TCP/IP fast-forwarding code. This lock is necessary, since the table has to be updated by the driver threads whenever packets are forwarded (to increment statistics, for instance). The table also has to be updated by the main TCP/IP stack thread when route table entries are added or deleted.
usually by external routing protocols such as RIP and OSPF. While these updates are infrequent, the mutex protecting the table is necessary to prevent the data structures from becoming corrupted during forwarding-table modifications.

To reduce the lock contention and improve scalability and performance with SMP, we modified the fast-forwarding code to use a separate forwarding table for each processor. So, in a two-way SMP system, we now maintain two separate copies of the forwarding table, each with its own forwarding entries (which are duplicated for each CPU), and a separate mutex protecting each table. In principle, this should reduce lock contention, as each driver thread, running on a separate processor, would no longer contend for a shared forwarding table. It also has the benefit of improving cache utilization. The modified stack architecture is shown at right in Figure 3.

We repeated our tests, this time with the new architecture. In Table 2 and Figure 4, you’ll see that the forwarding rate for small packets is now about 20 per cent faster than the original SMP results. Furthermore, a new system profiler trace, shown in Figure 5, reveals that resource contention has been virtually eliminated: threads 5 and 6 now run concurrently on each processor without any interruptions, rescheduling, or lock contention.
<table>
<thead>
<tr>
<th>Packet Size</th>
<th>Packet Rate (PPS per port)</th>
<th>Total Throughput (Mbit/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>341390</td>
<td>436</td>
</tr>
<tr>
<td>128</td>
<td>339602</td>
<td>869</td>
</tr>
<tr>
<td>256</td>
<td>330858</td>
<td>1693</td>
</tr>
<tr>
<td>512</td>
<td>195312</td>
<td>2000</td>
</tr>
<tr>
<td>768</td>
<td>130208</td>
<td>2000</td>
</tr>
<tr>
<td>1024</td>
<td>97656</td>
<td>2000</td>
</tr>
<tr>
<td>1280</td>
<td>78125</td>
<td>2000</td>
</tr>
<tr>
<td>1514</td>
<td>65876</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 2 — Throughput results after lock contention was reduced. Forwarding rate for small packets is now about 10 per cent faster than original SMP results.

Figure 4 — Throughput results after lock contention was reduced. (See comments for Table 2.)
Case Study: System Tracing

Causing QNX Software Systems

Case Summary
Since the system profiler in the QNX Momentics development suite hooks directly into the instrumented kernel, we were able to capture system trace events as the test progressed. By using the system profiler to graphically visualize our system, we easily pinpointed resource contention as the cause of the bottleneck and quickly corrected the problem.

A good system profiler is nonintrusive; it provides insight without requiring code modifications, and has little or no effect on system behavior. Properly implemented, it can even be used to diagnose a live system without interrupting or unduly degrading the services provided by that system — a real boon for high-end routers, 9-1-1 dispatch systems, and other applications that must remain continuously available.

References
RFC 2544, “Benchmarking Methodology for Network Interconnect Devices”
http://www.faqs.org/rfcs/rfc2544.html

Adtech AX/4000 Broadband Test System

BCM91250A (SWARM) evaluation board product brief
http://www.broadcom.com

QNX Neutrino RTOS product brief
About QNX Software Systems

Founded in 1980, QNX Software Systems is the industry leader in realtime, microkernel OS technology. The inherent reliability, scalable architecture, and proven performance of the QNX Neutrino RTOS make it the most trusted foundation for future-ready applications in the networking, automotive, medical, and industrial automation markets. Companies worldwide like Cisco, Ford, Johnson Controls, Siemens, and Texaco depend on QNX technology for their mission- and life-critical applications. Headquartered in Ottawa, Canada, QNX Software Systems maintains offices in North America, Europe, and Asia, and distributes its products in more than 100 countries worldwide.

www.qnx.com