Breaking up gigabit Ethernet's VoIP bottlenecks

Cedric Walravens and Benjamin Gaidioz

GIGABIT ETHERNET IS A POPULAR BROADBAND technology for good reason. It not only can transfer large amounts of data through a large network at high speeds, but it also is a mature technology with an appealing performance to cost-of-ownership ratio. Unfortunately, though, gigabit Ethernet suffers a performance drop when it is handling traffic with small packets: it tends to lose them.

This is a serious problem because small packets have become important in many real-time applications—in Voice over Internet Protocol (VoIP), for example, which carries voice conversations over the Internet. Every VoIP connection sends 50 packets over the network every second, packets that contain only 160 B or less. And since this small-packet, high-rate, real-time type of traffic is conveyed across the Internet, all Linux-based firewalls, routers, and Web caches suffer from the packet-loss problem too.

Fortunately, there is a cure. The Ethernet controller's driver is the culprit. Its negligent memory management is responsible for the small-packet loss. A technique called receive descriptor recycling (RDR) provides a solution. By actively reusing the software descriptors needed to administer the traffic flow, RDR alleviates the controller bottleneck. An RDR-enabled controller can reduce small-packet loss by 40%.

We have studied high-rate, small-packet traffic in an Ethernet controller, as implemented in an Intel e1000 network interface card (NIC) Linux driver, which is widely used in high-end systems and servers. We have also evaluated the performance boost provided by the RDR technique. (For background, see "Researching Gigabit Ethernet Performance.")

MEASURING NETWORK PERFORMANCE

The performance of a network can be measured in terms of packet rate, packet loss, response time, and throughput. In addition, important criteria for the network equipment are the efficient utilization of the central processing unit (CPU) and low resource requirements. When a network setup is optimized for high-rate small-packet traffic, inevitable limitations arise, e.g., CPU and resource requirements increase in order to accommodate high rates of small packets, as does the host bus utilization. Note that transmit performance is not affected by small-packet traffic to the same extent as receive performance. This asymmetry exists because the local host cannot usually overwhelm the Ethernet controller with outgoing traffic.

The setup we used in our study of network performance (Fig. 1) consisted

Digital Object Identifier 10.1109/MPOT.2007.913659

of a dual-CPU server connected through gigabit Ethernet to a network processor, which acted as a very powerful traffic generating source. Table 1 presents an overview of relevant system specifications.

The tests were performed on raw Ethernet frames in the IEEE 802.3 medium access control (MAC) format; see Fig. 2. They consist of a 14-B header followed by variable-length user data (the payload having a minimum of 46 B) and a 4 B cyclic redundancy check (CRC) at the end for data integrity, resulting in a total data overhead of 18 B per frame and a 20 B gap between consecutive frames.

RECEIVE THROUGHPUT

The expression for raw throughput is

$$\text{Throughput} = \frac{\text{payload} + \text{overload}}{\text{payload} + \text{overload} + \text{gap}} \times \text{bandwidth}.$$

Filling in the values imposed by the IEEE 802.3 standard results in a plot of the theoretical throughput as a function of payload size (the solid line in Fig. 3). As the measurements presented in the next paragraphs will show, this theoretical deduction closely models the behavior of send throughput of frames by an NIC. The measured receive throughput (the dotted line in Fig. 3), on the other hand, differs from what is sent, the result of some kind of bottleneck. It is this bottleneck that we investigated and analyzed.

Note that the send curve in Fig. 3 does not start at the origin, since a zero payload in the throughput equation results in a nonzero throughput because of the overhead present in MAC frames (see Fig. 2).

For the throughput measurements, the network processor (NP) acted as a source, flooding the server with packets (Fig. 1). Figure 4 shows the bit rate throughput measured when frames with variable payload size on one port of the server were received from one port of the NP. The bit rate measured at the output of the NP (send side) matches the theory well. In the range of 46 to 200 B of payload, the reception bit rate measured at the server (receiving side) is much lower (data are lost). In higher payload ranges, the sent bit rate is reached (no losses). Receiving on both ports of the dual card simultaneously produces an identical plot.

Figure 4 also shows the theoretical limit of the peripheral component

Researching gigabit Ethernet performance

Many researchers have evaluated gigabit Ethernet performance in various ways. A. Barczyk, A. Carbone, and coworkers studied full-link load effects, while M. L. Loeb and colleagues studied the influence of the Transport Control Protocol/Internet Protocol (TCP/IP), as P. Gray and A. Betz did, independently. (See "Read More About It.")

To discover the most fundamental factors affecting gigabit Ethernet performance, the authors of this article conducted measurements on raw Ethernet frames, avoiding any additional overhead imposed by higher-level network protocols such as TCP/IP. R. Hughes-Jones and coworkers took a similar approach, but their work did not focus on the performance of small packets. A group composed of A. Barczyk, D. Bortolotti, and others conducted some work on small-packet traffic effects using raw Ethernet frames, but no thorough low-level hardware-based analysis was performed.

Although Intel has released a document on small-packet traffic performance, it simply presents recommendations without solid measurements to prove any performance gain.

Table 1. Specifications of hardware used in the study.

Host	Chipset PCI-X bus	CPU System bus	Linux Kernel
SVR01	ServerWorks GC 64 b, 133 MHz	Dual Xeon 2.4 GHz 400 MHz	Scientific 3.0.4 2.6.12-smp
SVR02	Dell SC1425 64 b, 133 MHz	Dual Xeon 2.8 GHz 800 MHz	Scientific 3.0.4 2.611-smp
Type	**Ports**	**Chipset**	**Host bus (max.)**
Intel Pro/1000MT	Dual	82546EB	PCI-X 64 b, 133 MHz
Intel Pro/1000MT	Quad	82546EB	PCI-X 64 b, 133 MHz
IBM PowerNP NP4GS3	Tri	BCM5700	PCI

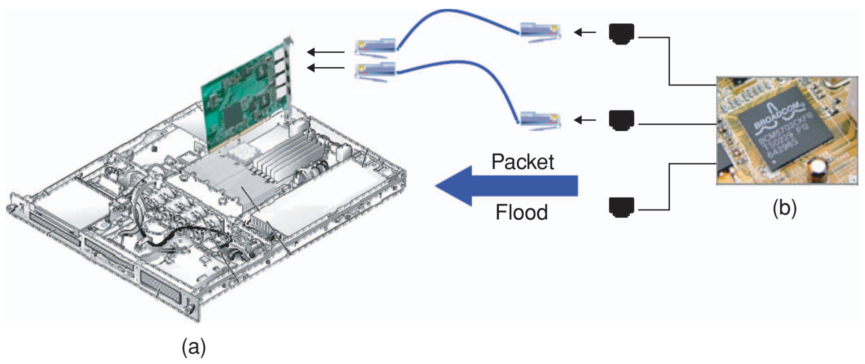

Fig. 1 Overview of experimental setup. The (a) server is connected via a gigabit Ethernet network to the (b) network processor, which acts as a traffic generating source.

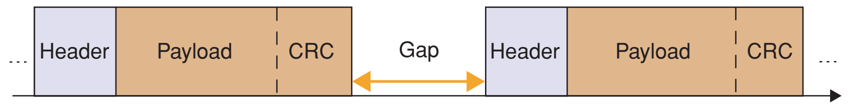

Fig. 2 Ethernet frames in the IEEE 802.3 MAC format

interconnect (PCI) bus. The PCI standard specifies a computer bus for attaching peripheral devices to a computer motherboard. It connects hardware devices such as NICs, sound cards, and graphic cards with the system bus and eventually the CPU. Clearly, for single-port operation, this PCI bus bandwidth will not pose any limit on gigabit Ethernet transfers, even though it is a theoretical value that will never actually be reached due to numerous nonidealities.

The slope of the first part (payload smaller than 200 B) of the receive throughput curve is 0.004, which is in agreement with measurements by A. Barczyk and coworkers (see "Read More About It"), who attributed it to the transfer setup on the PCI bus. We, however, reached a different conclusion about what contributes to this slope.

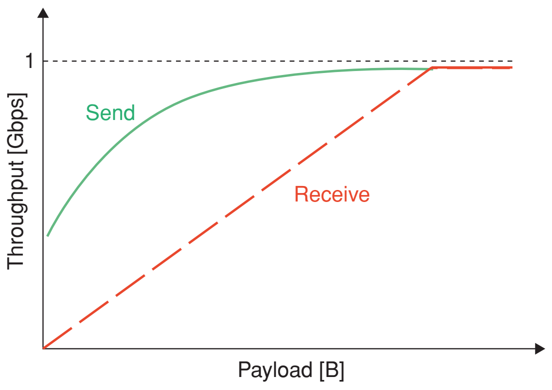

Fig. 3 Raw MAC network throughput

LOW-LEVEL MEASUREMENTS

Our receive throughput measurements indicate the real performance of the network but do not give us a clue about the nature of the bottlenecks in the network. Therefore, we carried out low-level tests, including PCI traffic analysis and kernel profiling, emphasizing the hardware and software, respectively.

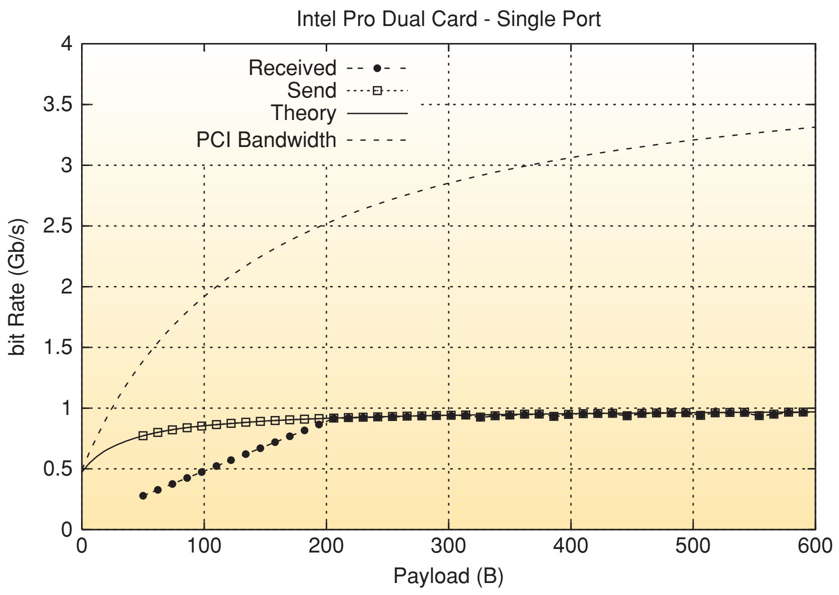

Fig. 4 Receive throughput from NP source to the server, plotted with theoretical gigabit Ethernet limit and PCI bandwidth

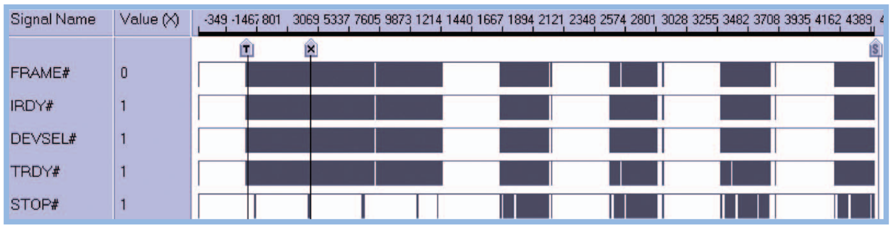

Fig. 5 Transition between smooth and stumble PCI traffic, with one of two ports receiving from NP in PCI mode (64 bit, 66 MHz)

Bus Analysis

Like the network overhead issues, PCI bus protocols also impose penalties on small-packet traffic. In addition to the actual data transfer, each bus transaction requires extra control cycles, which introduce overhead (e.g., arbitration latency, address phases, attribute phases, wait states). Bus overhead costs become more pronounced with small-packet traffic as bus control and wait cycles consume a larger fraction of each bus transaction.

We analyzed, in addition to simple PCI, the more advanced and faster PCI-Extended (PCI-X) bus protocol, which is found in many high-end servers. Compared to PCI, it provides a higher bus bandwidth and is more fault-tolerant.

Figure 5 shows the two regimes of transmission—smooth and stumble—across the PCI bus, again when the NIC is flooded by an NP in the setup of Fig. 1. The dark areas indicate that the signal varies a lot, i.e., frames are transmitted across the bus. This plot was taken in PCI mode, be we observed exactly the same behavior for PCI-X traffic.

During the smooth regime, all frames are nicely put on the PCI bus behind each other [with an interframe delay of 54 clock ticks (CLKs), or 0.8 μs at 66 MHz]. The Intel Pro/1000 MT Ethernet controller uses receive descriptors to keep the books. Such a descriptor is basically a pointer to a block of configurable size in the computer's memory where the NIC is to store a received frame.

Receive descriptors (RDs) are made available to the NIC in groups of 16 consecutive descriptors (see Fig. 6). Such a group is identified by its tail, i.e., the address of the last RD in this group (the receive descriptor tail, or RDT). When a frame is received, the Ethernet controller uses the next available RD in the current RDT to find out where in memory to store the frame (i). After the subsequent memory transfer of the frame finishes (ii), this RD is sent back to the driver (iii) to advertise the presence of a frame at that memory location (delay of 7 CLKs).

As Fig. 6 shows, the (ii) and (iii) transfers happen across the PCI(-X) bus. They can thus be identified in Fig. 7, which represents an interpreted version of Fig. 5. This trace shows, left to right and top to bottom, the advancing time axis (in clock ticks). The series of green dots are frames that are written to main memory. The following traffic pattern can be extracted for the smooth regime (0 to 14,500 CLKs):

1) Four frames (green dots) are transferred to main memory through memory transfer bursts with an interframe delay of about 45 CLKs, or 0.7 μs.

2) After 25 CLKs, or 0.4 μs, the four corresponding RDs are written back to the driver (one red spot).

3) Steps 1 and 2 are repeated 16 times (for a total of 64 frames), after which four new RDTs are buffered from the driver (small gaps in between series of 64 frames).

4) The NIC acquires a new RDT, allowing it to fetch 16 new RDs, and the cycle starts again from step 1 onward.

Under normal circumstances, the driver will provide the NIC with new RDTs (marked "rdt" in Fig. 7), allowing it to fetch newly allocated RDs in time. This transfer of new RDs continues until all allocated descriptors for that batch have been provided or until the card's frame receive buffer is nearly full and the card terminates the descriptor transfer by raising the STOP# signal in order not to lose any frames. Since an NP is flooding the network card during this test (with approximately 1 million packets per second), the termination will happen more and more, as seen on the STOP# signal line in Fig. 5. After the descriptor transfer, the frame receive buffer quickly empties to regain the time lost during this transfer. All frames are now put on the PCI bus with a minimum delay of seven CLKs in between.

Clearly, under persistent heavy traffic, the card will become unable to fetch enough descriptors to keep up with the high frame rate and at the same time send every single frame to the

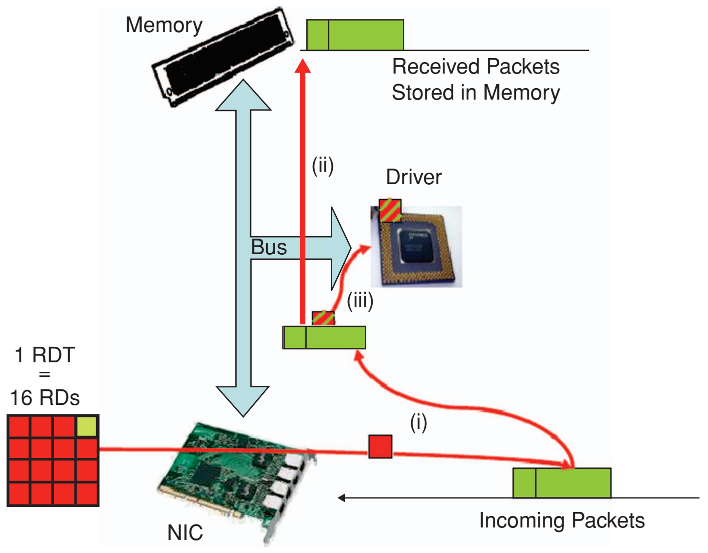

Fig. 6 Schematic overview of the path of receive descriptors

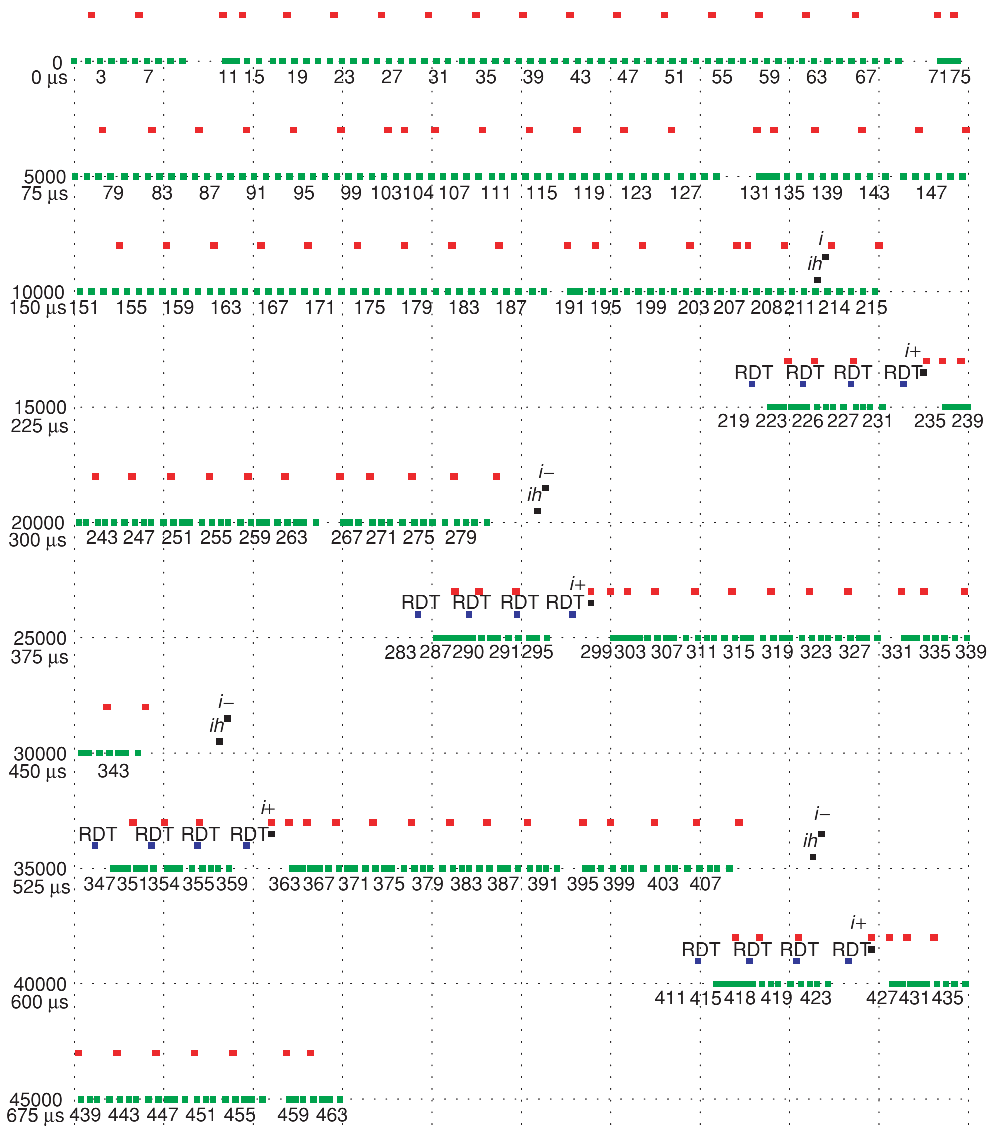

Fig. 7 Trace for the transmission from NP to server (from Fig. 5)

system. The card will have to allow a first-in, first-out (FIFO) receive buffer to overrun and silently drop frames until it has some more RDs to quickly empty the buffer and try to start receiving again. The relief is, however, of short duration, as the card will soon have exhausted its (already) very limited pool of RDs, and it will have to wait again.

This is what happens in the second—the so-called stumble regime (14,500 CLKs to end)—where huge gaps of 4,000 CLKs show the lack of RDs is preventing any further traffic until new RDTs are received. This results in many FIFO buffer overruns and a huge packet loss of up to 500,000 packets per second. Furthermore, the omnipresent STOP# indicates that any transfer that takes longer than absolutely necessary is abruptly terminated by the NIC.

In the smooth part, the system can process one frame per microsecond (i.e., what the NP sends). As the system tumbles down into stumble regime, this number degrades to a value of merely 0.3 frames per microsecond, or 0.28 Gb/s, in accordance with earlier measurements, e.g., Fig. 4. This indicates that the linear part for small packet sizes is caused by this stumble behavior.

While it might be tempting to think the PCI bus or NIC is the cause of this bottleneck, calculations of PCI bus utilization proved otherwise. During the whole trace, peak data rates did not reach any higher than 250 MB/s. Since the practical limit for a PCI bus is about 50% of the theoretical maximum, which is 533 MB/s in this case, this is acceptable and it was already a clear indication that neither the PCI bus nor the card was responsible for this stumble behavior. And since this trace was taken for a single port receiving frames, it is clear that a PCI (64-b, 66 MHz) will pose a bottleneck when more than one port and/or card is operating on the same bus.

As noted earlier, we performed similar measurements on the PCI-X bus, with comparable results and conclusions with regard to stumble behavior.

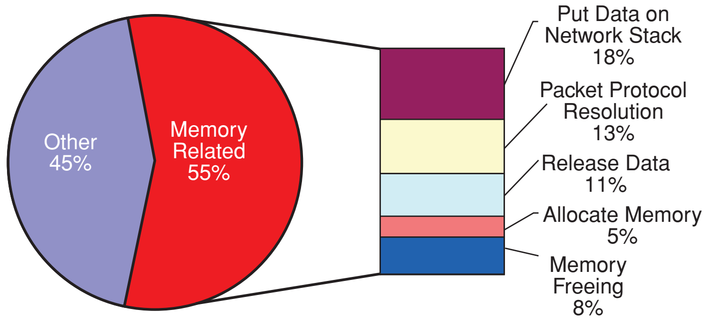

Fig. 8 Summary of relative occupation of the CPU during a 5-minute flood by the NP (one of two ports in PCI-X mode, 64-b, 133 MHz)

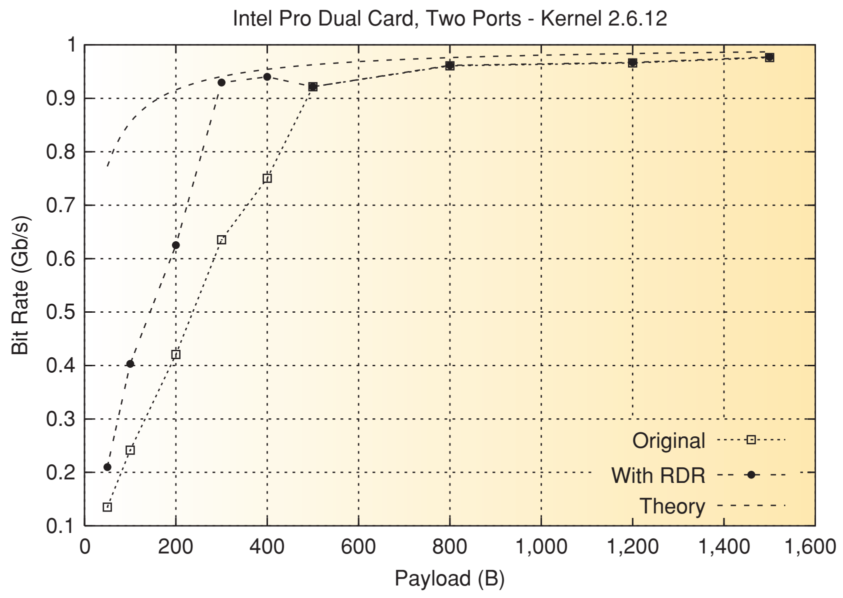

Fig. 9 Receive throughput from NP to server for two ports on a dual Intel NIC using the official driver (solid line) and the patched driver with RDR (dashed line)

A look at the kernel's memory management information made clear that memory access was responsible for the large gaps in stumble regime. These gaps result from the occasional reallocation of large quantities of main memory by the driver, during which, obviously, the main memory needs to be accessed. Meanwhile others—for example, memory transfers by the NIC—are prevented from accessing the main memory.

It is not the memory bank technology that is causing the bottleneck but rather the way the driver handles memory access. Simple on the back-of-the-envelope calculations show that standard DDR-2 400-MHz (PC2-3200) RAM memory modules provide enough raw bandwidth.

It is clear that the Intel Pro/1000 MT Ethernet Controller card, when used in combination with a fast PCI-X host bus, will not become a bottleneck, even for quad port operation. We therefore took a closer look at the software side of the network infrastructure, i.e., the Linux operating system and the Intel e1000 Ethernet Controller driver.

Kernel Profiling

Our software testing employed the OProfile package, a systemwide profiler for Linux systems, capable of profiling all running code at low overhead. It reports on the number of times a certain software function call (in user space, kernel, or driver, for example) has been made and what percentage of the total samples this represents.

Figure 8 summarizes results of the OProfile tests. Analyzing the exact content of the most frequently made software calls, OProfile established that these are involved with freeing and reallocating RDs. This gave us the idea to tune the e1000 driver to more cleverly handle this RD processing, and implement some kind of receive descriptor recycling.

RECYCLING DESCRIPTORS

Analyzing the NIC driver source code for packet reception handling pointed out that the most frequently called software functions are memory-related (Fig.

8). Especially for small payload sizes, the situation only worsens because there are more descriptors and thus more memory management overhead.

It is this overhead that prevents the driver from quickly sending new RDs to the Ethernet controller, as they need to be freed and reallocated first. This is why the receive descriptor recycling mechanism was devised. The idea is to allocate a fixed number of permanent descriptors, which are reused every time, effectively taking away the need for the costly reallocation overhead. The only processing that remains to be done is resetting some fields in the descriptors.

We implemented the RDR mechanism for the Intel standard e1000 Linux Ethernet driver, on which the original measurements described in this article were performed. The improvement afforded by RDR could then be seen readily. Our report for CERN, listed in "Read more About It," gives details of the code implementations.

The same tests described above were repeated with the RDR mechanism in operation. For small packets, the RDR-enabled driver was able to reduce packet loss by 40% (Fig. 9). The influence is most evident for short-delay packets. The performance of higher payloads remained unchanged with RDR.

CONCLUSIONS AND FUTURE WORK

In our study, we measured several benchmarks for the gigabit Ethernet hardware and the Ethernet controller driver. Among the parameters analyzed were throughput, packet loss, and efficiency of resource utilization. Emphasis was placed on high-rate, small-packet network traffic under near real-time constraints. Applying low-level analysis to the Intel Pro card, we showed that the current bottleneck lies in the way the driver handles the descriptor memory management, which proves to be fatal for small packets.

To remedy the situation, we implemented a receive descriptor recycling mechanism, the Intel Pro e1000 Ethernet driver. With that addition to the network, small-packet performance improved by 40% in terms of increased throughput and reduced packet loss.

By targeting high-rate, small-packet traffic, and including low-level PCI(-X) bus analysis and kernel profiling, the study extends gigabit Ethernet performance to the full spectrum of traffic possibilities. The mechanism developed can be employed to increase performance of all applications that require high throughput for small packet sizes, e.g., VoIP.

The high reliability of small-size transmissions is also crucial for the correctness of calibration runs of some experiments at the new Large Hadron Collider (LHC) particle accelerator in the CERN nuclear research center, Geneva, Switzerland. The receive descriptor recycling will be tested there in a real-life environment.

Plans for future work include verifying the results for other driver implementations that suffer similar bottlenecks and investigating the possibility of turning off cache coherency by requesting non-cacheable descriptor buffer memory so as to prevent numerous occasions of cache thrashing.

ACKNOWLEDGMENTS

This work was funded by the Summer Student Internship Committee of CERN, Geneva, within the framework of the Summer Student Program 2005.

READ MORE ABOUT IT

- Voice Over IP-per call bandwidth consumption. [Online]. Available: http://cisco.com/warp/public/788/pkt-voice-general/bwidth_consume.html
- A. Barczyk, A. Carbone, J.-P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, G. Peco, and V. Vagnoni, "Reliability of datagram transmission on gigabit Ethernet at full link load," CERN, Geneva, Rep. CERN/LHCb 2004-030, 2004.
- M.L. Loeb, A.J. Rindos, W.G. Holland, and S.P. Woolet, "Gigabit Ethernet PCI adapter performance," *IEEE Network*, Mar. 2001.
- P. Gray and A. Betz, "Performance evaluation of copper-based gigabit Ethernet interfaces," in *Proc. 27th Annu. IEEE Conf. Local Computer Network*, Tampa, FL, 2002, pp. 679–690.
- R. Hughes-Jones, P. Clarke, and S. Dallison, "Performance of 1 and 10 gigabit Ethernet cards with server quality motherboards," *Future Gener. Comput. Syst.*, vol. 21, pp. 469–488, Apr. 2005.
- A. Barczyk, D. Bortolotti, A. Carbone, J.-P. Dufey, D. Galli, B. Gaidioz, D. Gregori, B. Jost, U. Marconi, N. Neufeld, G. Peco, and V. Vagnoni, "High rate packets transmission on Ethernet LAN using commodity hardware," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 810–816, June 2006.
- "Small packet traffic performance optimization for 8255x and 8254x Ethernet controllers," Intel, Santa Clara, CA, Appl. Note AP-453 rev. 1.0, Sept. 2003.
- *Ethernet LAN Medium Access Control (MAC) Specification*, IEEE Standard 802.3, 1985.
- J.R. Allen Jr. et al., "IBM PowerNP network processor: Hardware, software, and applications," *IBM J. Res. Dev.*, vol. 47, Mar. 2003.
- *PCI Local Bus Specification*, PCI Special Interest Group Standard rev. 2.1, 1995.
- *PCI-X Protocol Specification*, PCI Special Interest Group Standard rev. 2.0a, 2003.
- *PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer's Manual*," rev. 2.5, Intel, Santa Clara, CA, July 2005.
- C. Walravens and B. Gaidioz, "Low level gigabit Ethernet analysis for the LHCb computing farm," CERN, Geneva, Rep. CERN/LHCb 2005-091, 2005.
- OProfile development pages. [Online]. Available: http://oprofile.sourceforge.net
- e1000 source code at the Linux cross reference project. [Online]. Available: http://lxr.linux.no/source/drivers/net/e1000/
- C. Walravens and B. Gaidioz, "Receive descriptor recycling for small packet high speed Ethernet traffic," in *Proc. IEEE Mediterranean Electrotechnical Conf. (MELECON '06)*, Malaga, Spain, 2006, pp. 1252–1256.
- O.S. Bruning, P. Collier, et al., Eds., *LHC Design Report*. Geneva: CERN, 2004.

ABOUT THE AUTHORS

Cedric Walravens (c.walravens@ieee.org) received an M.S. in eletrical engineering, with a specialization in micro-electronics in 2006, from Katholieke Universiteit Leuven, Belgium. Currently, he is a research assistant at the MICAS group of the Katholieke Universiteit Leuven. He is working toward a Ph.D. degree on low power digital control and data paths.

Benjamin Gaidioz (benjamin.gaidioz@cern.ch) received his Ph.D. degree in computing from Ecole Normale Sup'erieure de Lyon, France, in 2003, where the topic of his Ph.D. was IP quality of service. He then entered CERN as a computing engineer and was involved for two years in developments for the data acquisition system of the LHCb experiment. In 2005, he became a member of the CERN IT department where he currently works on computing grid matters, mainly monitoring, in close contact with the ATLAS experiment.