Using Switched Ethernet for Hard Real-Time Communication

Jork Loeser
TU Dresden, Germany
jork@os.inf.tu-dresden.de

Hermann Haertig
Dresden, Germany
haertig@os.inf.tu-dresden.de

Abstract

Previous results on traf£c shaping on Switched Ethernet technology demonstrate its practicability and effectiveness for hard real-time communication [8]. The application-to-application delays on a 5-node network were reported to be less than a millisecond with both Fast Ethernet and Gigabit Ethernet technology with a link utilization of 93% and 49%.

In this paper we describe directions of research to extend this work, targeting performance bottlenecks we identi£ed in the current design. We present ideas on of¤oading the traf£c shaping process and the receive process into the £rmware of a network interface card (NIC) and we re£ne the one-shot reservation protocol for best-effort traf£c.

Keywords: Real Time Parallel Computing

1. Motivation

Ethernet as de£ned in the IEEE 802.3 standard is **the** commodity network since decades, and has undergone a number of changes in its existence. It is used for hard real-time communication already, and demanding applications continue to emerge. A typical example is factory automation, where Ethernet replaces CAN for performance and cost reasons. Other applications can be found in the context of professional audio mastering (audio-LAN) or DMIDI [10], where multiple interactively controlled nodes generate audio data. The bandwidth requirements in these scenarios are ten to hundred megabytes a second and delays are expected to be a few milliseconds.

Switched Ethernet is a star-based topology, which in contrast to traditional, bus-based CSMA/CD Ethernet entirely avoids collisions. Thus for real-time communication, node cooperation is needed only for bandwidth control, but not to avoid collisions. It was our starting assumption that with £ne grained traf£c shaping as only means of node cooperation, we should be able to achieve lower guaranteed delays and higher bandwidth utilization than real-time approaches for CSMA/CD Ethernet such as time-slotted and token-passing approaches.

We validated this assumption in [8]. We demonstrated that commodity Switched Ethernet technology can be used for low-latency hard real-time communication.

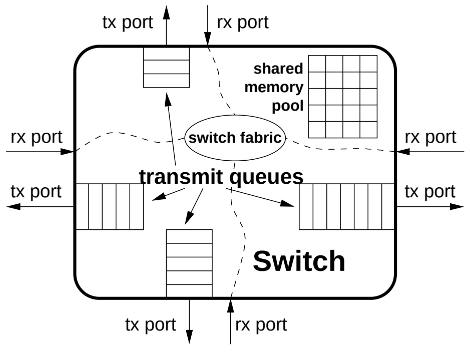

Figure 1: Buffering inside an output-queueing Switch. If queueing a frame is necessary, memory is allocated from a shared memory pool and assigned to the corresponding queue.

In this paper we target performance bottlenecks we identi£ed in our host-only based implementation. Therefore, we discuss ideas on of¤oading the traf£c shaping process and the receive process into an intelligent network interface card (NIC). Further, we re£ne the one-shot reservation protocol for best-effort traf£c.

2. Established Results

In this section we describe brie¤y the results we obtained so far, introducing the reader the traf£c shaping approach. We provide performance measurements to both show the effectiveness of the traf£c shaping approach and to motivate the of¤oading into the NIC.

2.1 Background

Figure 1 shows a typical Ethernet switch. The switch has *N*=4 receive ports, control logic, buffer and *N* queued transmit ports. When a frame arrives at the switch, the control logic determines the transmit port and tries to transmit the frame immediately. If the port is busy because another frame is already being sent, the frame is stored in the transmit ports queue, which is a £rst-in £rst-out (FIFO) queue. If no more memory is available for storing, the received frame is dropped.

Bounding delays

Obtaining the maximum queue lengths (backlog) and the maximum queueing delay in network switches has been researched in the past [3, 2]. In [6] we used the network calculus introduced by Boudec to derive bounds speci£cally for Ethernet. The results are the following:

For calculating the bounds for a speci£c output port, we describe the traf£c arriving at a speci£c switch input port k to be forwarded to this output port by a so-called T-SPEC (C, M, r_k, b_k). C is the network capacity, this is 100MBit/s for Fast Ethernet and 1000Mbit/s for Gigabit Ethernet. M is the maximum frame size, 1514 bytes for Ethernet. r_k describes the average bandwidth and b_k allows for some burstiness. The T-SPEC means that in any time interval of length t not more than $\min(C*t+M, r_k*t+b_k)$ bytes arrive at input port k for the considered output port.

For later reference we de£ne g_k for all $k = 1\ldots N$ as $g_k = \frac{b_k - M}{C - r_k}$ and g_{max} as the maximum of all g_k. t_{mux} denotes a switch-speci£c parameter describing the maximum delay (without queueing effects) after which the switch starts to transmit a frame once it is received.

Then the maximum delay t_{switch} of a frame for the considered output port at the switch is

$$t_{switch} \leq \sum_{k=1}^{N} \frac{b_k}{C} - g_{max} * (1 - \sum_{k=1}^{N} \frac{r_k}{C}) + t_{mux}. \tag{1}$$

The maximum queue length, this is the amount of memory needed to store the queued frames, is given by $t_{switch} * C$, or

$$B \leq \sum_{k=1}^{N} b - g_{max} * (C - \sum_{k=1}^{N} r) + C * t_{mux}. \tag{2}$$

If B exceeds the amount of memory the switch can use for buffering, frame loss may occur. For hard real-time systems this must be prevented.

2.2 Shaping the traf£c

With Ethernet, it must be ensured that traf£c leaving a node already conforms to previously de£ned T-SPECs. To achieve this, all *sending nodes* apply token-bucket traf£c shapers to all transmitted data. A bucket size b and a £ll rate r result in traf£c conforming to the T-SPEC (C, M, r, b).

Practicability considerations suggest to have multiple connections at each node, with one traf£c shaper per connection. Considering the execution-time of these traf£c shapers makes clear that a minimum traf£c shaping interval T_s must be de£ned: Once the bucket gets empty, the next packet is generated not earlier than T_s time units later. Note that T_s determines the bucket size, and hence the burst size of a connection: The bucket must hold at least the amount of data that can arrive in an interval of length T_s, which is $r * T_s$. As a result, the maximum queueing delay at the switch is in¤uenced by T_s, leading to a trade-off between delay and CPU usage.

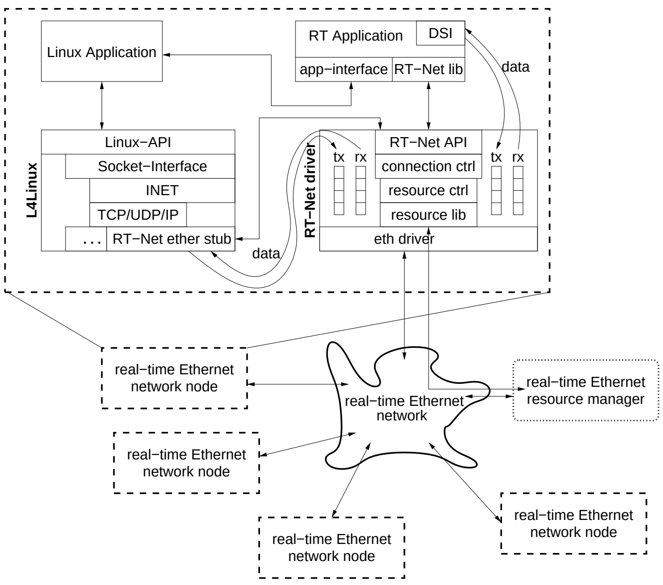

Figure 2: The network architecture and the application model.

Implementation in a real-time OS

We implemented the described network access model in our own network stack to provide application-to-application real-time data transfer.

The application model of our approach is shown in Figure 2. An *RT-Net driver* directly interacts with the network interface card (NIC). The RT-Net driver shapes the outgoing traf£c and polices incoming traf£c to avoid CPU overload situations. It offers connection-oriented packet-based interfaces to its clients. This allows accounting of transmitted traf£c and early demultiplexing of received traf£c, both for real-time and best-effort traf£c.

Each connection has its own token bucket parameter set including the current state of the bucket. The granularity of bandwidth reservation is 1 byte/ms. Subject to the traf£c shaping process is the overall length of a frame, including its MAC header and higher-level protocol headers such as IP and UDP. The minimum bandwidth that can be reserved corresponds to one minimal-sized packet per millisecond, which is about 100KByte/s.

Real-time traf£c is transferred to and from *real-time clients* using *real-time connections*. Real-time connections are unidirectional UDP/IP connections, so real-time applications can built there own protocol atop UDP/IP. The UDP/IP protocol handling is done at the RT-Net driver. Therefore, the source and destination addresses and ports of a connection are set at connection establishment.

Best-effort traf£c is transferred to and from *best-effort clients* using *best-effort connections*. The best-effort clients are allowed to send any desired frame to the network and they receive most of the frames coming from the network. Typically, best-effort clients implement IP-stacks.

To allow L^{4}Linux [5] to access the RT-Net driver, we implemented a L^{4}Linux stub-driver emulating an Ethernet device.

Best-effort send traf£c Best-effort traf£c should utilize all remaining bandwidth, which is not used by real-time traf£c. Multiple best-effort senders in a network should be able to share the free bandwidth. Therefore, we considered reserving a suf£ciently high and £xed bandwidth for each best-effort send connection not as an option.

Instead, we reserve only a small amount of bandwidth for every best-effort send connection. If the best-effort sender realizes its need for a higher bandwidth, it requests an additional *one-shot reservation*. This one-shot reservation is valid only for a few hundred milliseconds immediately after the reservation. During this time, the sender can transmit its data. If the time is over, and the sender still has to send data, it has to request another reservation.

2.3 Measurements

In [8] we did a number of measurements both with Fast Ethernet an Gigabit Ethernet to analyze (i) to what extend the theory behind real-time transfer trough traf£c shaping can be applied to existing hardware, (ii) what the costs are, especially for the host CPU, and (iii) whether host running a non real-time operating system can share the network. Especially we wanted to know if additional overheads in¤uencing the achievable delays are moderate and hence if the theory is applicable.

We found that traf£c shaping is a practical and effective method for hard real-time communication. We found also that in software-based implementations there is a trade-off between CPU usage and achievable delays at the network. The rest of this section contains the results of our Fast Ethernet measurements.

Setup

Figure 3 depicts our measuring setup: a switch (Level-One "FSW-2108TX", capable of buffering 86 1514-byte frames) in the middle is connected to £ve nodes. Node **A** generates a enumerated and timestamped test packet every millisecond and sends it to node **B**. Nodes **C**, **D** and **E** send traf£c of different shapes to node **B**. We measure the maximum packet transmission delay from node **A** to node **B** and test for packet loss. An additional "black cable" connects the nodes **A** and **B** for a precise clock synchronization [7] with a clock accuracy of $\leq$ 10µs.

We performed three experiments with different traf£c shaping intervals T_s at the senders (10ms, 1ms and 100µs). The bandwidths for nodes **C**, **D** and **E** were 30MBit/s, 32MBit/s and 20MBit/s in all experiments.

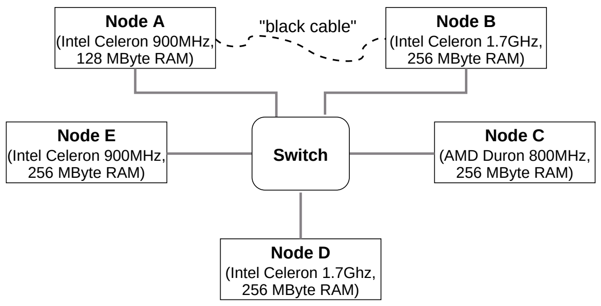
Node A
(Intel Celeron 900MHz, 128 MByte RAM)
"black cable"
Node B
(Intel Celeron 1.7GHz, 256 MByte RAM)
Node E
(Intel Celeron 900MHz, 256 MByte RAM)
Switch
Node C
(AMD Duron 800MHz, 256 MByte RAM)
Node D
(Intel Celeron 1.7Ghz, 256 MByte RAM)

Figure 3: Our general measuring setup: £ve nodes connected to a switch. Nodes **A** and **B** are additionally connected by the "black cable" for precise time synchronization.

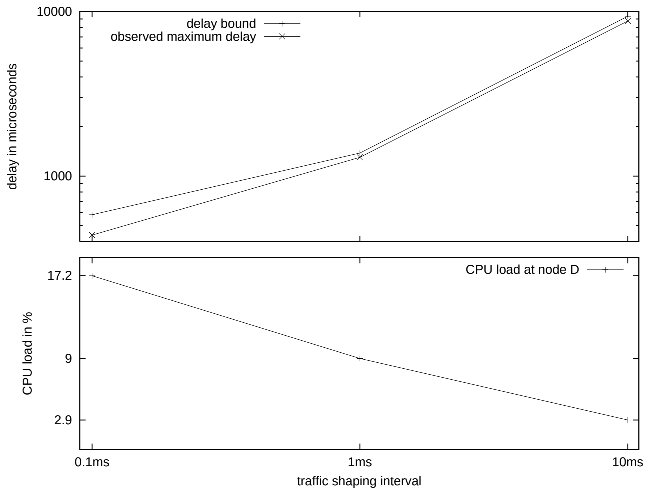

Figure 4: Delay/CPU trade-off with different traf£c shaping intervals. The depicted CPU load is obtained from node **D**.

Achieved Throughput and Delays

Due to framing overhead, the achievable bandwidth is limited to 98.6MBit/s with Fast Ethernet. We actually sent slightly over 92MBit/s to node **B**, thus utilized its link to 93%. With this utilization, we achieved maximum delays from **A** to **B** of 9.4ms, 1.4ms and 0.582µs, depending on T_s.

CPU utilization

We also measured the CPU requirement of traf£c shaping on nodes **C**, **D** and **E**. With our real-time system executing, the nodes need between 2% and 21% of their CPU time, depending on T_s, their speed and send bandwidth. The delay/CPU trade-off is demonstrated in Figure 4. Clearly, you can see the in¤uence of the decreased shaping intervals to the CPU usage. Thus, there is a trade-off between traf£c shaping accuracy, and hence transmission delay bounds, and CPU usage in the nodes connected to the network.

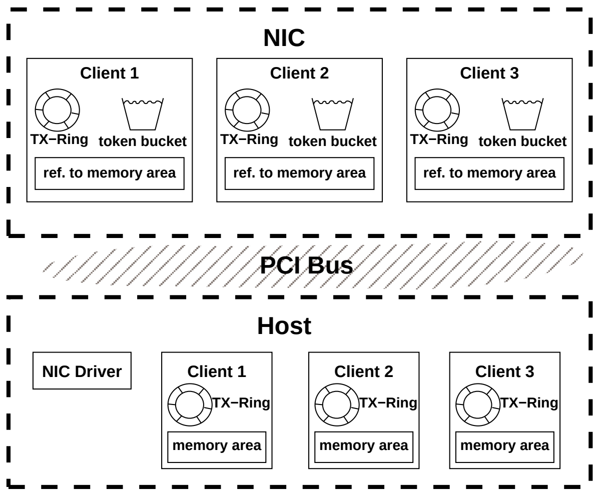

Figure 5: Of¤oaded transmission architecture. The TX-Rings and the memory areas are shared between the NIC and the clients.

3. Planned work

Figure 4 in Section 2.3 demonstrated that the CPU utilization increases substantially when a low network latency is required. This motivates a technology that has been applied successfully to lower the CPU utilization of network processes, this is £rmware of¤oading [1, 4, 9]. The idea is to instruct an intelligent network interface card (NIC) to perform some of the resource critical tasks, disburdening the host CPU.

While the U-Net project [1] and the Myrinet GM protocol [9] used £rmware of¤oading for performance reasons, Dannowski implemented the policing of incoming network traf£c at an ATM NIC [4] to bound the CPU utilization in a real-time system. A side-effect of the of¤oaded policing was an of¤oaded demultiplexing of received traf£c, allowing a real zero-copy receive process. As copying of network data is known to seriously impact the performance of network processing, zero-copy implementations should be favored whenever possible.

Regarding traf£c shaping on Switched Ethernet, £rmware of¤oading can be used for early demultiplexing in the receive path and for accelerating the traf£c shaping process in the transmit path. Furthermore, an interaction with the NIC driver for normal transmit operations might even be circumvented at all, meaning that no context switch to the driver is needed for sending data.

3.1 Data transmission

Our planned architecture is described in Figure 5. The NIC has a notion of a connection per client, and periodically polls the established connections for data to send. Assigned to each send connection is a set of parameters. This set contains

i) the memory area the client is allowed to send data from
ii) the send ring with references into the memory area
iii) the reserved bandwidth
iv) the current token bucket status.

The memory area describes a contiguous region of the physical memory. Together with the send ring it is mapped into the clients address space, and is thus shared between the NIC and the client. To isolate the clients from each other, different clients use different memory areas and send rings.

The transmission of data is similar to that of a normal send ring-based NIC: If a client wants to transmit data, it puts a reference into the contiguous region into the send ring. The NIC discovers this, veri£es the memory reference, checks whether the client can send traf£c (traf£c shaping), sends the data and optionally raises an interrupt to signals the successful send operation. In the case a client is not allowed to send traf£c as its token bucket is empty, the send request is silently ignored in this round, and the NIC continues with the next client. After some time and some iterations, the bucket will contain enough tokens to send the packet.

Thus, the following features must be added to a standard NIC implementation:

- support of multiple clients,
- shaping of the send traf£c.

Network driver tasks

Setting up a new connection at the NIC requires a prior bandwidth reservation and the reservation of the shared ring and the shared memory area. Obviously, these tasks should be executed by a trusted instance, which is the network driver. As the NIC typically does only has one interrupt line, dispatching of the sent-interrupt to the clients must be done by the driver too. Note that the clients typically do not request an interrupt for each send packet, instead interrupts are typically coalesced. Moreover, real-time connections typically generate traf£c according to their reservation, and thus should not observe a full send ring. Consequently, they may not need the signalling at all.

Summary To transmit data the clients directly interact with the NIC without switching the context to the network driver. Copying of data is not necessary. Best-effort connections need to interact with the driver for signalling of sent data. Compared to the current software implementation, this safes the client-driver interaction for sending data, and moves the traf£c shaping from the host CPU to the NIC.

3.2 Data Reception

The planned receive path architecture is depicted in Figure 6. As in the send path, the NIC has a notion of a connection for each client. The parameter set for each connection contains

i) the memory area the client is allowed to receive data into
ii) the receive ring with references into the memory area

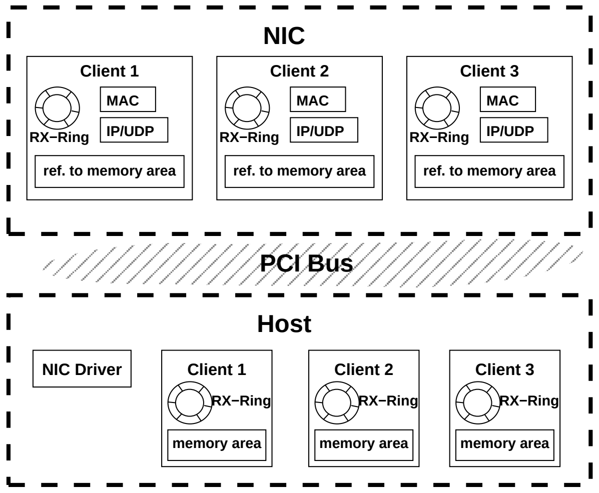

Figure 6: Of¤oaded receive architecture. The RX-Rings and the memory areas are shared between the NIC and the clients.

iii) MAC address
iv) real-time connections: IP address and UDP port number

The receive process is similar to that of a normal ring-based NIC: A client puts references into its memory area into the receive ring. When the NIC receives a packet, it identi£es the correct connection using the parameter sets, copies the data to the memory reference the client provided in its receive ring, and optionally raises an interrupt for signalling. To £nd the correct connection for a received UDP/IP packet, the NIC checks the packet against the real-time connections, which are UDP/IP connections. If this fails, or the packet was no UDP/IP packet, the NIC uses the MAC addresses to differentiate between the best-effort connections.

Thus, a standard NIC implementation must be extended to support multiple clients and to perform the demultiplexing.

Summary Compared to the current software implementation, the demultiplexing of data is done at the NIC before copying the data into the main memory. Thus, the data copy needed with the software implementation is avoided.

3.3 One-shot reservations

As stated in Section 2.2, we use one-shot reservations to dynamically adapt to the bandwidth need of best-effort send connections. The one-shot reservations are triggered by the clients of a connection (typically IP-stacks), as only the clients know whether they have a demand for a higher bandwidth.

Our current implementation uses TCP/IP connections to communicate with the resource manager at the network. With L^{4}Linux as client, this communication is sent over the same connection as the application traf£c of L^{4}Linux. Thus, once L^{4}Linux connection is congested, the one-shot reservations are delayed too. In experiments we found that L^{4}Linux with the current implementation utilizes the available bandwidth to 50%, with one-shot reservation times of 100ms.

Instead of using TCP/IP as the transport protocol for the reservation, we plan to switch to UDP/IP at a separate real-time connection. From the use if a lightweight protocol, combined with the £rmware of¤oading, we expect the average delays of the one-shot reservation messages to drop signi£cantly, resulting in a higher bandwidth utilization.

4. Conclusion

Firmware of¤oading has been successfully used by early adopters to place products with new features at the market before their competitors (Fore PCA-200E ATM card, 3Com 3C985 Gigabit Ethernet). Due to the high production costs of programmable cards, manufacturers often discontinued their production after a while and replaced them by nonprogrammable cards, once standards has been established and mass-production justi£ed the development of new hardware. However, in the £rst place they were used to explore new features.

We think, that the addition of traf£c shapers to Ethernet cards requires only moderate changes to current Ethernet chips, resulting in production costs comparable to those of normal Ethernet cards. After showing its practicability in a software implementation, the veri£cation of its effectiveness with £rmware of¤oading should be the next step to establish traf£c shaping on Switched Ethernet for low-latency hard real-time communication.

References

[1] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, Colorado, December 1995.

[2] J.-Y. Le Boudec and P. Thiran. *Network Calculus*. Springer Verlag Lecture Notes in Computer Science volume 2050, July 2001.

[3] Rene L. Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[4] U. Dannowski and H. Härtig. Policing of¤oaded. In *Proceedings of the Sixth IEEE Real-Time Technology and Application Symposium*, Washington D.C., May 2000.

[5] Hermann Härtig, Michael Hohmuth, and Jean Wolter. Taming Linux. In *Proceedings of the 5th Annual Australasian Conference on Parallel And Real-Time Systems (PART '98)*, Adelaide, Australia, September 1998.

[6] J. Loeser. Buffer Bounds of a FIFO Multiplexer . Technical Report TUD-FI03-15, Technische Universität Dresden, November 2003.

[7] J. Loeser. Measuring Microsecond Delays . Technical Report TUD-FI03-16, Technische Universität Dresden, November 2003.

[8] J. Loeser and H. Haertig. Low-latency hard real-time communication over switched ethernet. In *16th Euromicro Conference on Real-Time Systems*, Catania, Sicily, July 2004.

[9] Myricom Inc., 325 N. Santa Anita Ave, Arcadia, CA 91024. The GM message passing system. Available from `http://www.myri.com`.

[10] MIDI over Ethernet. `http://www.plus24.com/ieee1639/software.php`.