Low-latency Hard Real-Time Communication over Switched Ethernet

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

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

Abstract

Ethernet, the most widely used commodity network, increasingly moves toward switches as implementation technology thus replacing busses. This allows to use traf£c shaping techniques to implement hard real-time distributed systems on commodity networks. However, because Ethernet switches lack build-in policing features, nodes connected by Switched Ethernet need to be cooperative. Although the theory behind traf£c shaping for real-time communication is known for some time, it has not been considered for Ethernet so far.

In this paper we present the implementation of traf£c shaping on Switched Ethernet technology. We make thorough experiments to understand the cost and practical limits of using Fast and Gigabit Ethernet for hard real-time communication. We do measurements to analyze properties of switches and delays that we can achieve using these switches. We further analyze the in¤uences of non real-time Linux nodes sharing the network.

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. In the context of professional audio mastering (audio-LAN) Ethernet is experimented with: multiple nodes generate samples for hundreds of instruments in parallel and send them to a central mixer node. The process is interactively controlled and delays are expected to be less than 10ms. The bandwidth requirement for such a scenario is ten to hundred megabytes a second. Another application from the audio domain is DMIDI [30], an attempt to use Ethernet LANs for MIDI control commands. Although the bandwidth demands are moderate, the delays are expected to be a few milliseconds too.

CSMA/CD Ethernet Real-time approaches using the original bus-based Ethernet basically fall in three categories: token-based medium access control protocols, time slot-based protocols and statistical approaches. Time slots and token passing techniques are used by cooperating nodes for both: to avoid collisions and to obey the limit of bandwidth allocated to the participating nodes. Intuition indicates that the use of such techniques to avoid collisions limits the achievable utilization and increases the CPU load of the nodes much more than using more relaxed forms of cooperation that only control bandwidth allocation. Related research supports our intuition on the high cost for collision avoidance by node cooperation (see Section 5).

Switched Ethernet is a star-based topology providing a private collision domain to each of the ports of a switch. Collisions do not occur, thus node cooperation is needed only for bandwidth control, not any more 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 time-slotted and token-passing approaches, even though Switched Ethernet does not support policing in the switches as for example in ATM switches.

Although we heard rumors on the usage of Switched Ethernet in hard real-time applications that do not rely on pure time-driven technology (as for example in MARS [12]), we are not aware of any practical analysis.

In this paper, we make an attempt to close that gap and to validate our assumption as stated previously. We show how commodity Switched Ethernet technology can be used for low-latency hard real-time communication, provided the right operating system support is available: In Section 2 we £rst adapt well-established scheduling theory to our needs. In Section 3 we show how the needed traf£c shaping can be practically done using the Dresden real-time operating system (DROPS) as a basis for experimentation. In Section 4 we present detailed measurements. Section 5 surveys other work in the area of real-time networking and relates it to our approach. In Section 6 we summarize the paper and give an outlook to future work.

2. Background

Figure 1 shows a typical Ethernet switch. The switch has N=4 receive ports, control logic, buffer space and N queued transmit ports[1]. 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. The memory to store pending frames is obtained from a shared memory pool. If no more memory is available, the received frame is dropped.

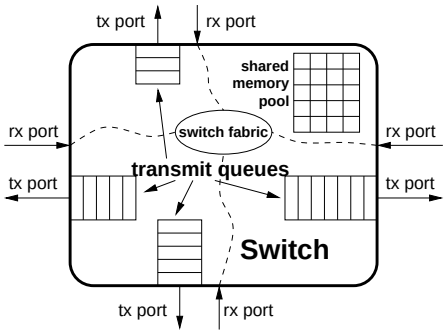

Figure 1: Buffering inside an output-queueing Switch. If queueing a frame is necessary, memory is allocated from a shared memory pool and assigned to the corresponding queue.

2.1. De£nitions

For the rest of this paper, we use the following terms to refer to times related to frame and packet transmission.

switch multiplexing delay ($\mathbf{t_{mux}}$) is 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.

queueing delay ($\mathbf{t_{queue}}$) is the time a queued frame sits in the queue of a switch plus the time needed to transmit it £nally. With £rst-in-£rst-out queues (FIFOs), queuing delays solely depend on the queue length, and bounding this length results in bounded queueing delays.

switch delay ($\mathbf{t_{switch}}$) is the time a frame is delayed at a switch. $t_{switch} = t_{mux} + t_{queue}$.

operating system delay ($\mathbf{t_{os}}$) covers the delays at the nodes due to interrupt handling and scheduling. It is the sum of the maximum delay at the sender and the maximum delay at the receiver.

frame transmission delay ($\mathbf{t_{frame}}$) is the time needed to transmit a frame over the Ethernet medium. For maximum sized frames (1514 bytes) t_{frame} is 121µs for Fast Ethernet and 12µs for Gigabit Ethernet.

packet transmission delay ($\mathbf{t_{trans}}$) is the application-to-application delay of a packet sent over the network. For two nodes connected by a switch, $t_{trans} = t_{switch} + t_{os} + t_{frame}$.

transmission delay bound ($\mathbf{t_{max}}$) is the upper bound of the packet transmission delay. This especially requires knowledge about the maximum queueing delay.

observed transmission delay ($\mathbf{t_{obs}}$) is the measured application-to-application delay of a packet sent over the network.

[1] For reasons given in the related work section we do not consider using multiple priority queues in switches.

2.2. Bounding delays

Obtaining the maximum queue lengths (backlog) and the maximum queueing delay in network switches has been intensively researched in the past, especially in the context of ATM networks. Cruz [6] was the £rst who developed a calculus on networking delays, and Boudec [2] later developed a more elegant calculus. Based on this, numerous work was done to calculate delays, buffer requirements and loss probabilities for statistical real-time systems [1, 13, 14, 19, 26, 27]. As described in detail in [15], we use the network calculus introduced by Boudec to derive bounds speci£cally for Ethernet and to derive rule-of-thumb formulae that deliver correct, but not necessarily tight bounds:

The delay and buffer bounds of a switch transmit port depend (i) on the traf£c arriving at the switch for that transmit port, described by its *arrival curve* α and (ii) on the availability of the switch to send that data, described by the *service curve* β. The arrival curve α is the sum of the arrival curves of the traf£c at the receive ports α_k, with k denoting the receive port. All α_k have the form of a *T-SPEC*, a speci£c form of traf£c description commonly used in the context of ATM: $\alpha_k(t) = \min(Ct + M, r_k t + b_k)$. C speci£es the maximum transmission rate and r_k the long term average rate. M is the maximum packet size and b_k expresses the burstiness of the traf£c. The service curve of an Ethernet switch is described by the *rate-latency function*: $\beta(t) = C * (t - t_{mux})^+$, with $(t - t_{mux})^+$ de£ned to be 0 for $t < t_{mux}$.

Obviously, the sum of the long term average input rates r_k of traf£c for one switch transmit port must not exceed the maximum rate of the network medium, thus it must hold:

$$\sum_{k=1}^{N} r_k \leq C \qquad (1)$$

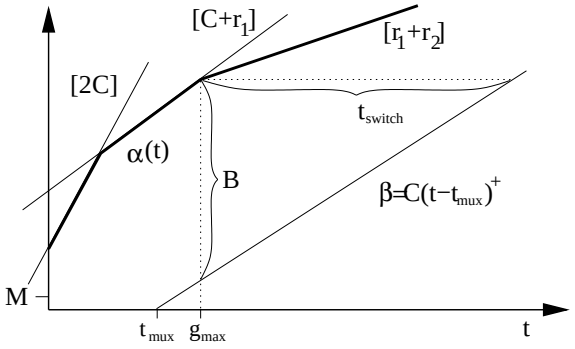

Figure 2: Illustration of the arrival curve $\alpha(t)$ (thick line) as the sum of two ¤ows (C,M,r_1,b) and (C,M,r_2,b). The slopes of the 3 parts of $\alpha(t)$ are $2C$, $C+r_1$ and r_1+r_2.

Figure 2 illustrates the arrival curve of a switch with two receive ports and its service curve. According to [2], the maximum backlog B is the maximum vertical distance between the arrival curve α and the service curve β. We de£ne g_k as the time of the in¤exion point of arrival curve α_k, thus

$$g_k = \frac{b_k - M}{C - r_k} \tag{2}$$

and de£ne g_{max} as the maximum of all g_k. It is easy to see that the maximum vertical distance between α and β is at g_{max}. As argued in [15], $t_{mux} \leq g_{max}$ in practice. Thus, the buffer bound is

$$B = \sum_{k=1}^{N} b_k + \sum_{k=1}^{N} r_k * g_{max} - C * (g_{max} - t_{mux}) \tag{3}$$

$$B = \sum_{k=1}^{N} b_k - g_{max} * (C - \sum_{k=1}^{N} r_k) + C * t_{mux} \tag{4}$$

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.

According to (1), the second addend in (4) is negative or zero, and hence a safe rule-of-thumb backlog formula is

$$B_{est} = \sum_{k=1}^{N} b_k + C * t_{mux}. \tag{5}$$

This means, the memory required in the switch can be estimated by the sum of the bursts in the T-SPECs plus a small £xed amount $(C * t_{mux})$.

According to Le Boudec [3], the maximum delay d of a system that offers a service curve β to a ¤ow that is constrained by an arrival curve α and serviced in FIFO order, is given by the maximum horizontal deviation between α and β. This is the distance between α and $C * (t - t_{mux})$ at g_{max}, divided by the slope of β, which is C.

Hence, the delay bound is

$$t_{switch} = \sum_{k=1}^{N} \frac{b_k}{C} - g_{max} * (1 - \sum_{k=1}^{N} \frac{r_k}{C}) + t_{mux} \tag{6}$$

By using equation (1) we £nd a rule-of-thumb bound:

$$t_{est} = \sum_{k=1}^{N} \frac{b_k}{C} + t_{mux}. \tag{7}$$

This means, an estimation for the maximum delay of the switch is given by the time needed to transmit the bursts of the T-SPECs with the ports maximum bandwidth plus the delay imposed by the electronics of the switch.

3. Shaping the traf£c

In contrast to other switched network architectures with real-time properties as an important design aspect (e.g., ATM), Ethernet switches typically have no notion of connections and speci£cally do no traf£c policing on their own. Hence, it must be ensured that traf£c arriving at the switch already conforms to previously de£ned T-SPECs.

To achieve this, all *sending nodes* apply token-bucket traf£c shapers [23] to all transmitted data. Using a token-bucket traf£c shaper with bucket size b and a £ll rate r to shape packets of maximum size M before sending them over a physical link with bandwidth C results in a traf£c stream conforming to the T-SPEC (C, M, r, b).

3.1. Traf£c shaper implementation aspects

In a real system, a node sends multiple streams, each with different bandwidth requirements, to different destination nodes. In other words, there are multiple connections at each node, with one traf£c shaper per connection.

For £nding bounds for the bucket sizes, let us look at the performance of a traf£c shaper implementation: When an ideal token-bucket shaper with parameters (r,b) is ¤ooded with packets of size s with an accumulated size bigger than b, it transmits a burst of length $\leq b$ immediately, and further packets after exact intervals of s/r time units. For example, a rate of 50MBit/s and a packet size of 1514 Bytes result in intervals of 242µs. Although this could be executed on modern CPUs, the resulting timer interrupt load could be prohibitive, depending on the application.

Therefore, we de£ne the traf£c shaping interval T_s and allow a traf£c shaper to replenish an empty bucket no more often than once every T_s time units. Thus, 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 $r * T_s$.

Summarizing, performance considerations lead to the de£nition of a traf£c shaping interval T_s. The bucket-size b of a connection with rate r is at least $r * T_s$. 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.

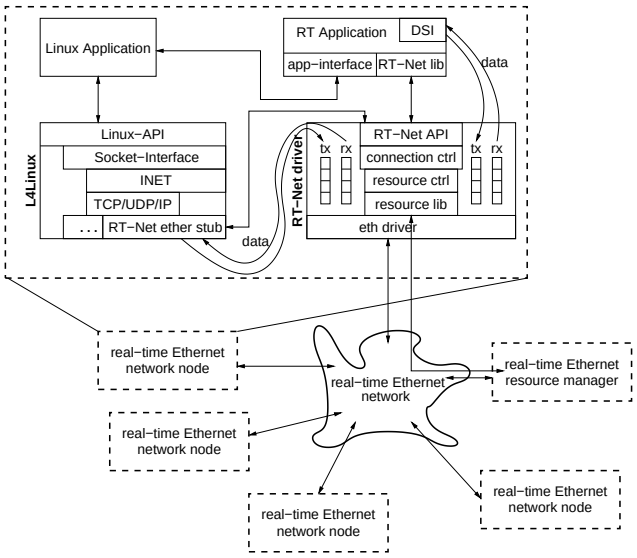

Figure 3: The network architecture and the application model.

3.2. 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.

Our system is built on the Dresden real-time operating system DROPS [11]. DROPS runs real-time applications that reserve the resources they need for proper operation. Remaining resources, including CPU cycles, memory, and network bandwidth, can be consumed by non real-time applications. A speci£c non-real-time application is L^{4}Linux[10], a variant of the Linux kernel that runs encapsulated in its own address space in user mode.

Application model The application model of our approach is shown in Figure 3. 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.

For L^{4}Linux we implemented a stub-driver emulating an Ethernet device. This allows L^{4}Linux to access the RT-Net driver.

For data transfer between the RT-Net driver and its clients, the DROPS Streaming Protocol (DSI) [17] is used. It allows a fast zero-copy, asynchronous interprocess communication for real-time and best-effort traf£c. In cases of overload it signals resulting data omission, which happens when incoming traf£c must be dropped.

Admission process An admission and reservation process prior to establishing a send connection at the RT-Net driver ensures that enough resources are available and given delay-guarantees are met. Therefore, each connection has its own parameter set containing its source and destination addresses, its bandwidth and its maximum acceptable delay. After local admission, a management instance at a dedicated node at the network is contacted. The management instance keeps track of all established connections. Upon admission, it veri£es that the parameters of the new connection can be met. It also veri£es that the increased switch delays resulting from the new connection are within the delay bounds of already established connections.

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 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.

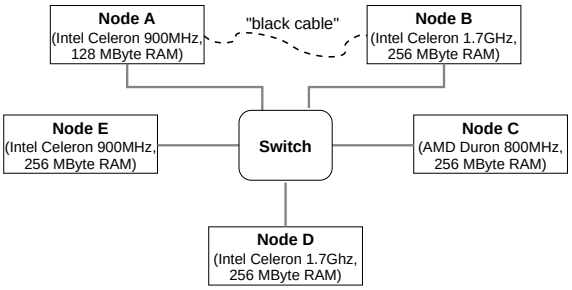

Figure 4: Our general measuring setup: £ve nodes connected to a switch. Nodes **A** and **B** are additionally connected by the "black cable" for precise time synchronization.

4. Measurements

The objectives of our measurements were:

- to reliably identify characteristics of Ethernet switches to be used for shaping decisions (Section 4.2)
- to £nd out the limits for utilization and delay guarantees that can be practically achieved (Section 4.3)
- to £nd the trade-off between CPU load and shaping granularity, which in¤uences the transmission delays (Sections 4.3.1 and 4.3.2)
- and to £nd whether the network can be shared between non real-time nodes and nodes doing real-time communication (Section 4.3.3)

4.1. Measurement setup

Figure 4 depicts our general measuring setup: a switch in the middle is connected to £ve nodes. Node **A** periodically generates test packets and sends them 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.

We analyzed three different 8-port Ethernet switches: a Fast Ethernet Level-One "FSW-2108TX" switch, a Fast Ethernet 3Com "Of£ceConnect Dual Speed Switch 8" switch and a Gigabit Ethernet Intel "Netstructure 470F" optical switch.

For Fast Ethernet measurements, all nodes are equipped with Intel EEPro/100 Fast Ethernet network cards. For Gigabit measurements, all nodes use 3Com 3C985B-SX type optical network cards (AceNIC II).

In some experiments we determine the CPU load, that is how many CPU cycles are consumed compared to the CPU cycles available during some time interval. Therefore we use a low-priority looper that consumes and counts all idle CPU cycles.

Achieving worst-case delays Oechslin encountered in [18] the dif£culties of reliably reproducing the worst-case with traf£c that is shaped according to a given set of T-SPECs. He found periodic traf£c patterns, *symmetric bursts*, that lead to maximum queue lengths with a high probability. Figure 5 shows the general pattern of symmetric bursts. Unless otherwise noted, we use these symmetric bursts in our experiments.

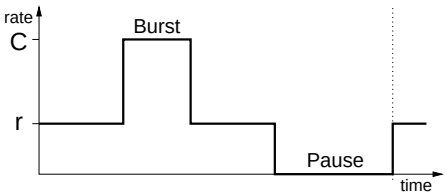

Figure 5: One period of a symmetric burst.

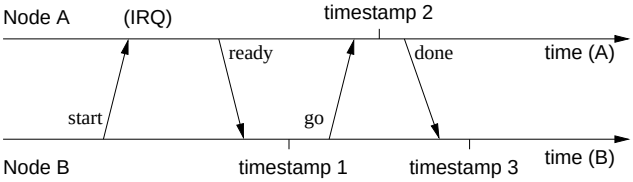

Figure 6: Resynchronization process

Measuring µ-second delays To measure transmission delays, we use test packets carrying timestamps and sequence numbers. A send application at node **A** generates the test packets. At node **B**, a receiving application compares the timestamps with its local clock and calculates the transmission delay (observed transmission delay).

We expect delays in the order of microseconds to milliseconds, and thus nodes **A** and **B** must be synchronized with an accuracy of a few microseconds. Therefore we connect the nodes by a parallel cable (the "black cable" in Figure 4) and applied a procedure similar to that of NTP as de£ned in RFC 1305 (Figure 6). A detailed description of the synchronization process as well as the derivation of its accuracy can be found in [16]. We achieve a clock accuracy $\leq$10µs. The resynchronization is required to run not more often than once every second, and less often in most cases. A resynchronization procedure takes not more than 20µs each.

Unless otherwise noted we generate the test packets from node **A** to node **B** the same way in all experiments. Node **A** uses the RT-Net driver to generate UDP-packets in minimum-sized Ethernet frames (64 bytes including all headers, 22 bytes UDP payload) every millisecond. At node **B** the RT-Net driver dispatches the test packets based on their UDP port and hands them over to the test application. Due to the regularity of the test traf£c and its small bandwidth the results of the experiments are mainly in¤uenced by the traf£c generated additionally at the other hosts.

4.2. Switch characteristics

For later interpretation of message delays or loss with respect to queueing, we £rst did basic measurements with

the switch and the network stacks.

Measuring switch multiplexing delays The equations in Section 2.2 for calculating switching delay and buffer bounds require a parameter t_{mux}. t_{mux} expresses the time it takes for a switch to start sending a packet after it received it, given the packet is not enqueued. To measure this time we compared the maximum observed transmission delay of a maximum sized Ethernet frame from nodes **A** to **B**, once directly connected and once connected by a switch. The three other nodes (**C**, **D**, **E**) mutually exchanged traf£c to put load on the switching fabric, but prevented queueing in the switch.

We collected one million samples for each test. We found that the Fast Ethernet switches add 45µs to the transmission delay. The Gigabit switch adds 25µs.

Test packet transmission delays Next we measured the maximum packet transmission delays of the 64-byte test packets between node **A** and **B** under the condition that the switch had not to queue any packets. As in the previous section, we loaded the switching fabric with parallel load. Table 1 shows the maximum transmission delays we observed.

100MBit, 3Com Of£ceConnect	65µs
100MBit, Level-One "FSW-2108TX"	80µs
1000MBit, AceNIC interrupt coalescing disabled	175µs
1000MBit, AceNIC interrupt coalescing enabled	238µs

Table 1: Maximum application-to-application packet transmission delays with different switches and driver features.

The AceNIC Gigabit Ethernet cards provide a sophisticated interrupt coalescing feature that reduces the interrupt load, although it possibly increases the delay on packet reception. For later comparison, the table contains the values for both con£gurations.

Measuring switch buffer capacities To use the traf£c shaping approach for real-time transfer, the switches must have enough buffer capacity for queueing packets. We analyzed whether the switches can be used by determining their buffer capacities available for queueing. Based on the general setup of Figure 4, nodes **C** and **D** sent two bursty traf£c streams to node **B**. The streams had a rate slightly under the half of the maximum medium bandwidth (100MBit/s and 1000MBit/s, respectively) each. They were shaped in an on-off form, thus a burst of an adjustable length b was followed by a pause. The maximum switch backlog required by these two streams is b [15].

We started with small burst lengths and increased the burst lengths until packet loss occurred. Table 2 shows the maximum burst lengths where no packet loss occurred.

The 3Com switch reliably buffers only 20KByte, thus it is ineligible for the traf£c-shaping approach. From the documentation of the Intel Gigabit switch we know that it can store up to 2MByte of data, and it turned out that it was node **B** that could not receive bursts of this size with Gigabit bandwidth.

Switch	(in 1514Byte-frames)	(in KByte)
100MBit, 3Com	14	20.5KByte
100MBit, Level-One	87	127.4KByte
1000MBit Intel	200	293KByte

Table 2: Maximum burst lengths without packet loss.

4.3. Measuring of traf£c-shaping effects

Knowing the parameters of the switches, we analyzed the effects of different traf£c shaping intervals to worst-case packet transmission delays and to CPU usage.

4.3.1. Fast Ethernet with DROPS

We began the measurements with the Fast Ethernet Level-One switch. All connected nodes executed the DROPS real-time system.

We performed three different experiments where nodes **C**, **D** and **E** sent data to node **B**. We varied the traf£c shaping intervals T_s at the senders, but kept the bandwidth reservation constant. Table 3 lists the reserved gross bandwidths and bucket sizes. The bucket size of a node is calculated as $b = r * T_s + M$ with r being the reserved bandwidth of that node and M the length of a maximum-sized Ethernet frame, which is 1514Bytes.

Node	**C** (40MBit/s)	**D** (32MBit/s)	**E** (20MBit/s)
T_s=10ms	51514 bytes	41514 bytes	26514 bytes
T_s=1ms	6515 bytes	5514 bytes	4014 bytes
T_s=100µs	2014 bytes	1914 bytes	1764 bytes

Table 3: Bucket sizes depending on the traf£c shaping interval T_s.

Buffer bounds and worst-case delays Table 4 shows the resulting buffer bounds and delays of the three con£gurations. The buffer bound is calculated from Equation (4). t_{max} is calculated from Equation (6), increased by the 80µs from Table 1. For calculating t_{est} the delay estimator (7) is used. t_{obs} is the maximum transmission delay we have observed in our experiments. In each experiment we collected 350,000 samples. No packets were lost.

	buffer bound	t_{max}	t_{est}	$t_{obs} \leq$
T_s=10ms	111.8KByte	9357µs	9731µs	8759µs
T_s=1ms	15.7KByte	1380µs	1345µs	1300µs
T_s=100µs	6.1KByte	582µs	506µs	438µs

Table 4: Buffer bounds in the switch and delay bounds for packet transmission from node **A** to node **B** depending on the traf£c shaping interval T_s.

The observed transmission delays are actually smaller than the theoretical bounds. This can be explained by the observation that the maximum queue length is only achieved in extremely rare situations at the switch, and these situations just did not happen during our experiments.

CPU usage To measure the CPU requirement of traf£c shaping, we repeated the experiments but modi£ed the send applications to generate traf£c as fast as possible. The symmetric burst generation required £ne and therefore expensive timers that would have falsi£ed our CPU measurements. Table 5 shows the CPU usage at nodes **C**, **D** and **E** with the modi£ed send applications.

Node	**C** (40MBit/s)	**D** (32MBit/s)	**E** (20MBit/s)
T_s=10ms	4.1%	2.9%	2.3%
T_s=1ms	11%	9%	7.2%
T_s=100µs	21.2%	17.2%	11.9%

Table 5: CPU load depending on the traf£c shaping interval.

The delay/CPU trade-off is demonstrated in Figure 7. Clearly, you can see the in¤uence of the decreased shaping intervals to the CPU usage. Thus, there is another trade-off between traf£c shaping accuracy, and hence transmission delay bounds, and CPU usage in the nodes connected to the network.

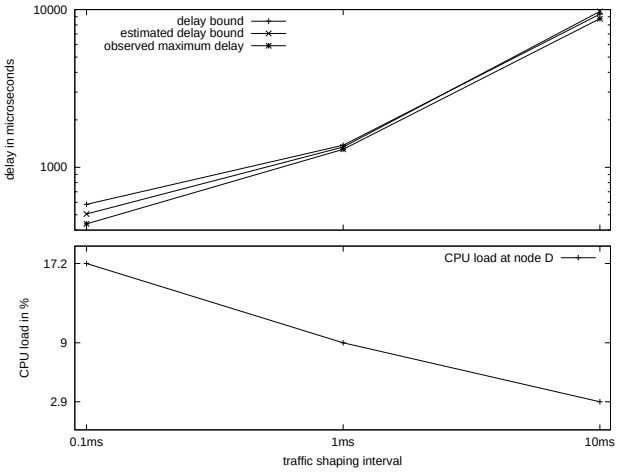

Figure 7: Delay/CPU trade-off with different traf£c shaping intervals. The depicted CPU load is obtained from node **D**.

Interpretation of results With maximum sized frames of 1514 bytes on Fast Ethernet the achievable bandwidth is limited to 98.6MBit/s due to framing overhead and interpacket gaps (corresponding to 8 + 12.5 Bytes). We actually sent slightly over 92MBit/s to node **B**, thus utilized its link to 93%. With this utilization, we can guarantee network delays of 9.4ms, 1.4ms and 0.582µs, respectively, depending on the amount of CPU cycles one is willing to spend.

The error by the delay estimator (7) in our experiments was less than 16%.

With a traf£c shaping interval of 10ms, nearly all the buffer capacity of the switch is needed for that one output port. In another experiment we tried to send two additional 30MBit-streams from node **C** and node **E** to node **D**, which immediately resulted in lost packets. With a traf£c shaping interval of 1ms no packet loss occurred.

4.3.2. Gigabit Ethernet with DROPS

For Gigabit Ethernet measurements we used the Intel Netstructure optical Switch. The AceNIC network cards had the interrupt coalescing feature enabled. We used a traf£c shaping interval of T_s=1ms. All nodes had the same bandwidth reservation of 160MBit/s each. Table 6 shows the bucket sizes and the CPU load at the sending nodes. According to Equation (4) 64KByte of switch buffer were needed. The delay bound of this con£guration was expected to be **687 µs**: the switch delay according to Equation (6) plus the 238µs maximum transmission delay at the non-queued switch from Table 1. We actually observed a maximum packet transmission delay of **906µs** with no packet loss.

Node	**C**	**D**	**E**
bucket size	21514 bytes	21514 bytes	21514 bytes
CPU load	48 %	30 %	39 %

Table 6: Bucket sizes and CPU load for the Gigabit Ethernet experiment with a traf£c shaping interval T_s=1ms.

We did another experiment with 100µs shaping intervals. To measure the expected small transmission delays with a better accuracy we disabled the interrupt coalescing features of the AceNIC network cards. This resulted in lower delays on packet transmission and reception, but it increased the interrupt load at all nodes. We had to reduce the bandwidth reservations to 80MBit/s each, and the CPU load increased signi£cantly. The bucket size in all nodes was selected to 2114 Bytes. The resulting delay bound was **247 µs** (72µs switch delay and 175µs according to Table 1). We actually observed a maximum packet transmission delay of **341µs** with no packet loss.

Interpretation of results With Gigabit Ethernet we measured longer delays than we expected. In other experiments it turned out that the receiving node **B** was just overloaded and it deferred the packet delivery to the test application.

This shows a general problem at the receiver: the demultiplexer has to spend CPU cycles for demultiplexing each packet, sometimes just to £nd out that no application is waiting for them. A solution to this was already given by Dannowski [7]. He applied early demultiplexing at the £rmware level of an ATM card, and successfully removed load from the CPU.

Node	**C** (40MBit/s)	**D** (32MBit/s)	**E** (20MBit/s)	$t_{max} \leq$	$t_{obs} \leq$
Linux 2.4.22, HTB, 10ms	54022 bytes	43536 bytes	27810 bytes	9807µs	6928µs
Linux 2.6.0-test9, TBF, 1ms	6450 bytes	5495 bytes	4000 bytes	1372µs	995µs

Table 7: Bucket sizes (in bytes), delay bound and observed delays for the Linux experiments.

With a maximum delay of 906µs we can utilize the output port to 49%. The limitation are the attached nodes, not the network switch. The buffer requirement of the one output port would allow for higher bandwidths on more output ports.

This result compares favorable to time-slotted approaches, which are very sensitive to the network jitter and delays inherent to switches. Schwarz reports in [22] about an implementation of the time triggered TTP/C protocol on Gigabit Ethernet. The delay and jitter of the network he analyzed result in a maximum overall utilization of 37MBit/s, corresponding to a 3.7% utilization.

4.3.3. Sharing a Fast Ethernet network with Linux

We analyzed whether the network can be shared between non real-time nodes and nodes doing real-time communication. Of course, the non real-time nodes must provide some sort of traf£c shaping, otherwise they could easily ¤ood the buffers in the switch. Beginning with kernel version 2.4 Linux includes an QoS subsystem and provides a number of queueing disciplines for this [28]. Among them are a token-bucket traf£c shaper and a hierarchical token-bucket traf£c shaper.

We looked at two Linux versions, the stable Linux-2.4.22 and the new Linux-2.6.0-test9. For traf£c shaping both versions use the periodic clock interrupt. Linux-2.4 kernels on the x86 architecture generate the clock interrupt with a frequency of 100Hz. With Linux-2.6 the frequency is 1kHz. Thus with Linux-2.4.22 we expect a shaping interval of 10ms resulting in switching delays in the order of 10ms. For Linux-2.6.0-test9 we expect a 1ms traf£c shaping interval and delays in the order of 1ms.

For traf£c shaping, we £rst looked at the hierarchical token-bucket traf£c shaper *HTB* [29]. It is often used in conjunction with DSL- and cable modems to minimize queueing delays inside the modems. When we £rst analyzed the results of HTB from the standard kernel, we found that it shapes the traf£c in intervals of 20ms, not in intervals of 10ms. After contacting the HTB author and tuning the kernel[2], it £nally shaped the traf£c periodically with an interval of 10ms, for both kernel versions 2.4 and 2.6.

Linux-2.4.22 with HTB traf£c shaper On nodes **C**, **D** and **E** we replaced DROPS with Linux-2.4.22 and used its HTB traf£c shaper. We con£gured HTB with the same bandwidths as in the previous experiments: 40MBit/s, 32MBit/s and 20MBit/s. HTB determined the buffer sizes for itself, they are displayed in Table 7. The table also contains the transmission delay bound according to Equation (6), increased by the 80µs from Table 1 for packet transmission delay without queueing.

We tried to generate symmetric bursts as we did with the DROPS setup. But for reasons we did not £nd out this did not work – the HTB traf£c shaper sooner or later delayed the traf£c for 10ms preventing any useful results. This should not happen, as the traf£c was generated conforming to the reservation. Thus, we randomly generated bursts with lengths according to the bucket size and breaks in between.

During the experiment node **A** sent 3 million test packets to node **B**. No packets were lost. The maximum observed packet transmission delay for the test packets was 7ms (last column in Table 7). The bigger difference to the theoretical bound compared to the DROPS experiments can be explained by the on-off shape of the bursts used.

Linux 2.6.0-test9 with TBF traf£c shaper In the second experiment we used Linux-2.6.0-test9 with its 1kHz timer. The traf£c was shaped by the token-bucket traf£c shaper TBF. We con£gured TBF with the same bandwidths as in the previous experiments. In contrast to HTB the TBF traf£c shaper requires the user to additionally specify the bucket sizes. We £gured out by experiments what bucket sizes are needed to achieve the desired bandwidth, details are given in the second row of Table 7. We had to generate random bursts as with the Linux 2.4 kernel.

During the experiment node **A** sent half a million test packets to node **B**. Again we observed no packet loss, and the maximum packet transmission delay was slightly under 1ms.

Robustness of a shared network We repeated both experiments and tried to distort the traf£c shaping process that at some point in time it starts to generate a higher network load than allowed. The idea was to use high interrupt load to force the system into a state where the shaper cannot send packets for a while. We expected that the shaper will catch up this lag later on by generating larger bursts. We observed however, that the traf£c shaper cannot be in¤uenced to generate a higher traf£c than expected.

Interpretation of results We conclude that Linux nodes can share a network with real-time nodes, although the different Linux kernel versions lead to different transmission

[2] changing de£nes: net/sched/sch_htb.c: HTB_HYSTERESIS=0, and include/net/pkt_sched.h: PSCHED_CLOCK_SOURCE=PSCHED_CPU

delays and switch buffer requirements. With the 2.4 series Linux, the 127KByte buffer need in the switch do not allow for more than one fully utilized switch output port. With the 2.6 series Linux, there is no such limitation.

5. Related work

Various real-time transfer solutions exist for the original CSMA/CD Ethernet, all of them ensure an exclusive access to the network medium. They often use a **token-based** approach, where a circulating token represents the permission to transmit data. The advantage of token-based solutions is their ¤exibility, as they can be used with almost any network architecture. The downside is that only one station owns the transmit right at a given time. This unnecessarily limits the performance on modern switched networks. Also, time and network bandwidth for the token management affects the overall performance. In [25] Venkatramani and Chiueh present results from the "RETHER" project. They simulated a 10MBit/s CSMA/CD Ethernet with a maximum token rotation time of 33ms (thus delays bounds are $\geq$33ms) and achieve a network utilization of 60%. With 100MBit/s Ethernet they gain only a small throughput increase, which they attribute to the dominance of software overheads.

Another method for controlling access to the network is the **time-slotted** approach. Examples can be found in [12, 22]. There are several problems with this approach: the longer the time-slots, the higher is the worst-case delay of messages transferred. Therefore, time-slots should be as short as possible. On the other side, time-slots must be long enough to prevent overlapping of messages due to delays and jitter imposed by the network. This may lead to a severe performance cut, as the 4% overall network utilization for Switched Gigabit Ethernet reported by Schwarz in [22].

A mixed approach is presented by Pedreiras and Almeida in **FTT-Ethernet** [20], where a central master periodically distributes tokens that allow to send data for a speci£c amount of time.

Kweon and Shin describe in [13] a method to achieve **statistical real-time** guarantees. By keeping the overall network traf£c below a certain limit, the probability of network collisions is bounded. Hence, a statistical guarantee for the transmission time and bandwidth can be given. Besides the probabilistic behavior of their approach, the overall network utilization decreases with stronger statistical guarantees. They report an experiment of 10 nodes connected by a 10MBit/s CSMA/CD Ethernet, exchanging real-time traf£c with a total bandwidth of 53KBit/s and non real-time traf£c with a total bandwidth of 4.4MBit/s. The deadline-miss ratio was 10^{-4} with a deadline of 129msec.

Lo Bello and others extended the statistical approach by **fuzzy traf£c smoothing** [4]. They use the overall throughput together with the number of collisions as network load indicators and feed them into fuzzy traf£c smoothers. This allows them to handle sporadic traf£c more ¤exible.

Schedulability conditions for different packet scheduling methods in network switches are given in [14]. The schedulability conditions allow to guarantee **delay bounds depending on the input traf£c characteristics** and the selected scheduling method in the switch. They apply their theory to single token-bucket shaped traf£c, and their result for FIFO scheduling coincides with Equation (7).

In [27], Watson and others apply the theory of Boudec [2] to switched Ethernet and discuss multiple switches in a line topology. In contrast to our traf£c model, they use the simpler model of single leaky-bucket shaped streams, and further neglect the processing time in the switch. For one switch, their results coincide with our estimators in Equations (7) and (5). Watson provides analytical and simulation results of a 100MBit/s switched Ethernet. With 50 nodes connected in a line topology and producing bursts of 15KByte with a high ($\geq$90%) network utilization, the delay bounds for message transfer are in the order of 400ms.

Several projects aim at building QoS-enabled switches or routers from scratch [5, 9]. In [24] Chiueh and Varadarajan present **EtheReal**, a switch that allows connection establishment with policing. In an experiment they con£gure an Intel PPro/200-based PC as a 4-port 100MBit/s Ethernet switch. They report switch delays of 10-60µs for a 50MBit/s non real-time connection with a parallel 100MBit/s real-time connection. The overall bandwidth through the switch is limited to 40.25MByte/s.

Guérin and others present in [8] a mechanism for providing quality of service (QoS) through **buffer management**. The idea is to limit explicitly the amount of buffer space the switch provides for a speci£c connection. With their approach bandwidth allocations can easily being managed, and free bandwidth is distributed fairly among best-effort traf£c. In contrast to our approach, this proposal requires installation of a £ltering instance inside the switch, thus it does not work with off-the-shelf hardware. They report a simulation of a 48MBit/s switched network. With a switch buffer size of 500KByte and FIFO scheduling, they achieve a network utilization of 85% without loosing data. For a 99% utilization, 5MByte switch buffer are needed.

The IEEE 802.3 extension 802.1p allows to assign **priorities to individual network frames**, which are used by a priority-based scheduler inside a switch. However, the prioritized traf£c still must be shaped to prevent overload situations and mutual interactions. Further, Pedreiras and others report in [21] that lower prioritized traf£c may lock switch memory that cannot be used for higher prioritized traf£c then. Thus, there is no real isolation between the different priorities.

6. Summary

We have shown, both theoretically and in experiments that traf£c shaping can be used to achieve reliable packet transmission with bounded transmission delay. Besides exact delay and buffer bounds, we also provided rule-of-thumb formulae for a quick estimation. In experiments with both Fast and Gigabit Ethernet, we were able to guarantee sub-millisecond delays for a network utilization of 93% and 49%, respectively.

The achievable maximum transmission delay mainly depends on the granularity of the traf£c shaping, that is how often the traf£c shapers are run. This results in an CPU/delay bound trade-off.

It is worth to note that every node connected to the network must shape its traf£c accordingly, but otherwise can send its traf£c whenever it wants. This allows the network to be shared with non real-time nodes, although there traf£c shaping capabilities affect the delay bounds. We successfully shared the network with Linux nodes of different kernel versions resulting in different delay bounds.

The small sensitivity of traf£c shaping to jitter also results in a signi£cant throughput bene£t compared to the time triggered approach, which was reported to achieve a 4% utilization for a TTP/C implementation on Gigabit Ethernet [22].

Acknowledgements We would like to thank Gerhard Fohler for his guidance, our anonymous reviewers for their comments and our colleges, especially Ronald Aigner, Adam Lackorzynski, Frank Mehnert and Lars Reuther for their supporting work on DROPS.

References

[1] J.-Y. L. Boudec and G. Hebuterne. Comment on a deterministic approach to the end-to-end analysis of packet ¤ows in connection oriented network. . *IEEE/ACM transactions on networking*, Feb. 2000.

[2] J.-Y. L. Boudec and P. Thiran. *Network Calculus*. Springer Verlag, LNCS volume 2050, July 2001.

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

[4] R. Caponetto, L. L. Bello, and O. Mirabella. Fuzzy Traf£c Smoothing: Another Step towards Real-Time Communication over Ethernet Networks. In *1st RTLIA*, Vienna, Austria, June 2002.

[5] N. Christin and J. Liebeherr. The QoSbox: A PC-Router for Quantitative Service Differentiation. Technical Report CS-2001-28, University of Virginia, Nov. 2001.

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

[7] U. Dannowski and H. Härtig. Policing of¤oaded. In *Proceedings of IEEE RTAS*, Washington D.C., May 2000.

[8] R. Guérin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Trough Buffer Management. In *Proceedings of ACM SIGCOMM98*, Vancouver, Canada, Aug. 1998.

[9] R. Guerin, L. Li, S. Nadas, P. Pan, and V. Peris. The Cost of QoS Support in Edge Devices: An Experimental Study. In *Proceedings of the IEEE Infocom*, New York, Mar. 1999.

[10] H. Härtig, M. Hohmuth, and J. Wolter. Taming Linux. In *Proceedings of PART'98*, Adelaide, Australia, Sept. 1998.

[11] H. Härtig, L. Reuther, J. Wolter, M. Borriss, and T. Paul. Cooperating resource managers. In *RTAS*, June 1999.

[12] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: the Mars approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[13] S.-K. Kweon and K. G. Shin. Achieving Real-Time Communication over Ethernet with Adaptive Traf£c Smoothing . In *RTAS*, Washington D.C., May 2000.

[14] J. Liebeherr, D. E. Wrege, and F. D. Exact Admission Control for Networks with a Bounded Delay Service. *IEEE/ACM Transactions on Networking*, 4(6), Nov. 1996.

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

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

[17] J. Löser, L. Reuther, and H. Härtig. Position summary: A streaming interface for real-time interprocess communication. In *HotOS*, Elmau, Germany, May 2001.

[18] P. Oechslin. Worst Case Arrivals of Leaky Bucket Constrained Sources: The Myth of the On-Off source . In *IWQoS*, New York, May 1997.

[19] S. D. Patek and J. Liebeherr. Position Paper on Networs with Aggregate Qualitiy-of-Service . In *Proceedings of the SPIE Conference #4526*, Oct. 2001.

[20] P. Pedreiras, L. Almeida, and P. Gai. The ftt-ethernet protocol: Merging ¤exibility, timeliness and ef£ciency. In *Euromicro ECRTS'02*. IEEE Press, June 2002.

[21] P. Pedreiras, R. Leite, and L. Almeida. Characterizing the Real-Time Behavior of Prioritized Switched-Ethernet . In *2nd RTLIA*, Porto, Portugal, June 2003.

[22] M. Schwarz. Implementation of a ttp/c cluster based on commercial gigabit ethernet components. Master's thesis, Technische Universität Wien, 2002.

[23] J. S. Turner. New Directions in Communications (or Which Way to the Information Age?). *IEEE Comm. Magazine*, 24(10):pp. 8–15, Oct. 1986.

[24] S. Varadarajan and T. Chiueh. EtheReal: A Host-Transparent Real-Time Fast Ethernet Switch. In *Proceedings of ICNP*, Austin, TX, Oct. 1998.

[25] C. Venkatramani and T. Chiueh. Supporting real-time traf£c on ethernet. In *Proceedings of IEEE RTSS*, Dec. 1994.

[26] S. Wang, D. Xuang, R. Bettati, and W. Zhao. Providing Absolute Differentiated Services for Real-Time Applications in Static-Priority Scheduling Networks . In *Infocom*, 2001.

[27] K. Watson and J. Jasperneite. Determining end-to-end delays using network calculus. In *Proceedings of IFAC FET*, Aveiro, Portugal, 2003.

[28] `http://www.lartc.org`.

[29] `http://www.luxik.cdi.cz/~devik/qos/htb`.

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