APPROACHES FOR REAL-TIME ETHERNET

eingereicht von:

Rainer Poisel

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom Ingenieur (FH)
(Dipl. Ing. (FH))

Fachhochschule St. Pölten

Studienrichtung: Telekommunikation und Medien

Begutachter:
FH-Prof. Dipl. Ing. Johann Haag
ZT Dipl. Ing. Günter Zeiler

St. Pölten, im Juni 2007

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material, which, to a substantial extent, has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

St. Pölten, June 15, 2007 Rainer Poisel

Abstract

English

Fieldbus systems are used in automation industry to connect complex devices such as machine tool controls [WB05, cmp. p. 272]. One basic element of these facilities is the shared media of communication participants. With this shared media it is possible to save communication lines when compared to parallel cabling.

Fieldbus systems are developed with a focus on real-time capabilities [Kut02, cmp. p. 20]. Ethernet, when seen as an additional and as a replacement communication technology, is faced with the same requirements on predictable timing behavior for data transmission. Standard ethernet is not real-time capable because of equal rights for communication participants when accessing the media [JJ01, cmp.]. Therefore different approaches and extensions have been developed and standardized.

This documents determines requirements for real-time data transmission on ethernet networks and contrasts available implementations. An experiment tests the real-time behavior of two different approaches for real-time ethernet networks.

Deutsch

Mit Feldbussystemen werden in der Automatisierungsbranche Einrichtungen wie z.B. Werkzeugmaschinensteuerungen vernetzt [WB05, vgl. S. 272]. Wesentliches Element dieser Einrichtungen ist ein den Kommunikationsteilnehmern gemeinsames Medium. Dieses gemeinsam benutzte Medium ermöglicht, im Gegensatz zum Ansatz der Direktverkabelung, das Einsparen von Kommunikationsleitungen.

Feldbussysteme werden unter dem Gesichtspunkt der echtzeitfähigen Datenübertragung entwickelt [Kut02, vgl. S. 21]. An Ethernet werden dabei als Ergänzungs- und Ersatzkommunikationsmittel die gleichen Anforderungen hinsichtlich der zeitlich vorhersehbaren Datenübertragung gestellt. Bei Standard-Ethernet ist aufgrund des gleichberechtigten Buszugriffs durch die Kommunikationsteilnehmer keine Echtzeitfähigkeit gegeben [JJ01, vgl.]. Aus diesem Grund wurden verschiedene Ansätze und Erweiterungen ausgearbeitet und standardisiert.

Dieses Dokument ermittelt die Voraussetzungen zur Echtzeitkommunikation mit Ethernet und stellt verfügbare Implementierungen gegenüber. Ein Experiment prüft das Echtzeitverhalten zweier verschiedener Ansätze zum Aufbau eines Echtzeitethernetnetzes.

Acknowledgements

First of all I want to thank FH Prof. Dipl.-Ing. Johann Haag who gave me the opportunity to write this diploma thesis under his supervision. Thanks are also due to Dipl. Ing. Thomas Baier, my tutor during my internship at kirchner SOFT GmbH, for his support and advice on how to implement the experiment and the statistical analysis afterwards.

Further I want to express my thanks to the library team of the university of applied sciences in St. Pölten. Your assistance with interlendings was a great help. You organized all the books I required for completing this diploma thesis on time.

And last but not least I would also like to thank my parents, my sisters and friends for their ongoing encouragement during the time of writing. Thank you for wishing me the best and for your patience when I tried to explain what I am actually describing in this thesis.

"Science is a wonderful thing if one does not have to earn one's living at it."

–Albert Einstein (1879 - 1955)

St. Pölten, Austria
June 15, 2007

Rainer Poisel

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Focus of this publication

Digital communications can be categorized in several ways. This document focuses on real-time and non-real-time communications. File transfer, e-mailing, remote-access are examples of non-real-time applications. The performance metrics for these applications are typically the average packet delay and throughput. Much of the complexity arises from the data-oriented loss-free communication between the systems. With real-time communication the quality of the service additionally depends on the time when data arrives at its recipient. Each message is associated with a deadline. This deadline is an effect of the transportation delay or latency [AKRS94, cmp.].

[Jen07, cmp.] classifies real-time into two categories:

- **hard real-time**: the result of the operation is considered useless after the deadline
- **soft real-time**: the system responds with a decreased service quality in case of a delay

Ethernet is a network technology. It was developed in the early seventies of the 20^{th} century at the Palo Alto Research Center in California. The first version was called Alohanet and concepted for the shared radio frequency band in the atmosphere [B. 04, cmp. p. 247-248]. The name "ethernet" derives from the media which is shared between all the nodes that are connected to the network - the so called "aether". Twisted-pair wiring became a big success. Because of the big competition between the manufacturers, components used in an ethernet network are very cheap. This led to the broad spreading of this technology as we know it today.

As the medium is shared, access to it has to be regulated. The so called "backoff" algorithm describes what happens if two or more clients try to use the medium at the same time [Hei02, cmp. p. 162-163]. Simultaneous access to the medium leads to a collision, letting the clients wait for a random period. In the worst case, a client may not be able to place its data onto the bus at all. Further it is not possible to determine the point in time when the data arrives at its destination.

This led to the development of many approaches for adapting ethernet hard- and software for real-time applications. Till now this challenge has been incumbent on fieldbusses. Fieldbusses are standardized communications systems for industrial applications which are usually real-time capable and very expensive.

1.2 Organization of this diploma thesis

This diploma thesis is divided into three major parts:

- Theoretical aspects: this chapter describes which requirements have to fulfilled to gain real-time capability in ethernet communications. Both hard- and software requirements are mentioned.
- The experiment: A real-time ethernet network has been implemented to demonstrate how to apply theoretical knowledge. Two approaches are contrasted.
- Industrial solutions: enumerates available solutions on the European market and how theoretical aspects as mentioned in the first part are applied.

Chapter 2

Theoretical aspects

2.1 Terminology

2.1.1 Process and Task

In scope of this document a process provides functions as reaction on real-time events. These functions are also called tasks. In other words a process can be built of several tasks [Sta03, cmp. p. 519].

Tasks can be further categorized into aperiodic tasks and periodic tasks. The former is associated with an execution deadline and the latter has to be executed several times in a specific time period.

2.1.2 Notation of data packets

[Har02, cmp. p. 28] lists what data packets on the layered network are called. This is shown in table 2.1.

LAYER	NOTATION
application layers	Message
transport layer	segment
network layer	datagram
data link layer	frame
physical layer	bits (no packets)

Table 2.1: Notation of data packets

2.2 What is Real-Time?

There are several definitions for real-time computing. Some of them are presented here:

A. Silberschatz, P. Galvin and G. Gagne defined the term “real-time” as follows:
“A real-time system is a computer system that requires not only that the computing

results be 'correct' but also that the results be produced withing a specified deadline period. Results produced after the deadline has passed – even if correct – may be of no real value." [SGG05, p. 695]

D. Abbott states in his publication that "The essence of real-time computing is not only that the computer responds to its environment fast enough, but that it responds reliably fast enough" [Abb06, p. 1]. So real-time programming focuses on meeting timing constraints in the midst of random asynchronous events reliably.

The online encyclopedia Wikipedia defines the term as follows: "In computer science, real-time computing (RTC) is the study of hardware and software systems which are subject to a 'real-time constraint' – i.e. operational deadlines from event to system response." [Wik07m]

Typical fields of application range from "specialized devices" such as ordinary home appliances (e.g. microwave ovens, dishwashers, etc.) [SGG05, cmp. p. 696], consumer digital devices (e.g. digital cameras, MP3-players), communication devices (e.g. cellular phones) to larger entities such as automobiles and airplanes. Here antilock brakes are mentioned as a part of an automobiles real-time system. Each wheel has a sensor which measures how much sliding and traction are occurring. Each of these sensors continuously sends data to the system controller. This is were real-time communications come into play. The controller in turn tells the braking mechanism of each wheel how much braking pressure to apply.

Real-time computing can be divided into two forms: hard real-time systems and soft real-time systems:

Hard Real-Time systems are classified as systems which lead to major damage if they don't finish their tasks till a defined dead-line. Devices meeting hard real-time constraints are often called "safety-critical systems". Examples are pacemakers, engine-controls, etc [Sta03, cmp. p. 519].

Soft Real-Time systems are classified as systems with which it is still useful to complete a task after the time limit has passed by [Sta03, cmp. p. 519].

2.2.1 Deterministic behavior

Real-time and deterministic behavior of a computer system are strongly related. [Vir07, cmp.] states that the state of a deterministic system is always well-defined. In a deterministic communication system all states of a data transmission can be exactly predetermined.

Hard real-time systems are strongly deterministic. It is possible to predict the timing behavior of all states a system could be in [LM06, cmp. p. 10].

2.3 Real-Time data processing

2.3.1 Requirements for real-time systems

The expected requirements for real-time systems are identified in [WB05, p. 1].

2.3.1.1 Accurate Timing

Results of real-time operations have to be available in due time (before the so called deadline). Cyclic times and sampling instances have to be preserved. Another timing issue is the reaction upon events. A real-time system has to react in a defined period to randomly arising internal or external events.

Douglas Jensen expressed the value of the behavior of real-time systems in time/utility functions [Jen07, cmp.]. Figure 2.1 shows the function of usefulness which is also stated in [LM06, cmp. p. 1]. The discrete function of usefulness ($f_{use}(t)$) for hard real-time systems can be expressed as shown in formula 2.1.

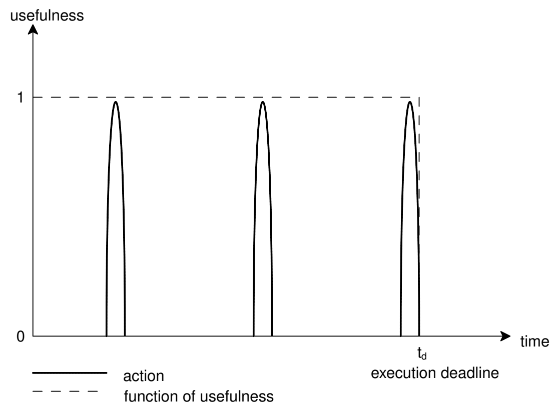


Figure 2.1: Graphical representation of the function of usefulness for accurate timing [LM06, p. 1]

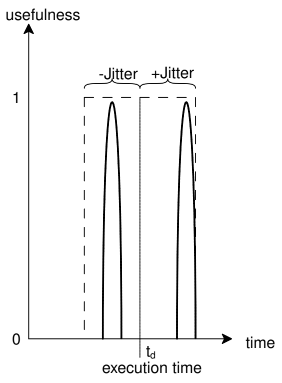


Figure 2.2: Graphical representation of the function of usefulness for concurrency [LM06, p. 1]

A system that handles events (or actions) after the deadline of execution (t_d) is of no value ($t > t_d$). In soft real-time systems events handled after the execution deadline are of limited value ($0 < f_{use}(t) \leq 1$).

$$f_{use}(t)\ _{[1]} = \begin{cases} 1, \ t \leq t_d \\ 0, \ t > t_d \end{cases} \tag{2.1}$$

2.3.1.2 Concurrence

Many tasks have to be managed at the same time, each with its own timing requirements [WB05, p. 1]. This characteristic defines a point in time when actions have to be executed. The deviation between actual time of execution and the optimal point of execution is called “jitter” and can be of positive and negative value [LM06, cmp. p. 1]. The timing window can be seen as the time period between negative and positive jitter. Actions that are performed too early are of no value as with actions that are executed too late.

Figure 2.2 depicts the relations between execution time and usefulness. Thus events have to occur within the timing window.

2.4 Real-Time in communication technology

This section describes the communications model which can be seen as a reference model for network communications in general. The implementation of how a network is integrated in an operating system is often called the “stack”. Implementations of real-time ethernets show differences in how their stack is implemented, which can be best compared with the generic ISO/OSI communications model.

2.4.1 The ISO/OSI communication model

[WB05, cmp. p. 257] states that in 1983 the ISO/OSI model was standardized in standard number ISO 7498. This led to improvements in communications between devices of different manufacturers.

The model splits communication into 7 layers. Each layer has clearly defined functions and offers some services for the next higher layer. Communication between layers happens through so called “primitives” [Hei02, cmp. p. 20-21]. As data perambulates through the stack (see “real communication path” in figure 2.3), header information is appended to user-data with each passed layer. On the receiver site this information is removed and interpreted by the regarding layer. So there are direct “virtual connections” (see figure 2.3) between the layers.

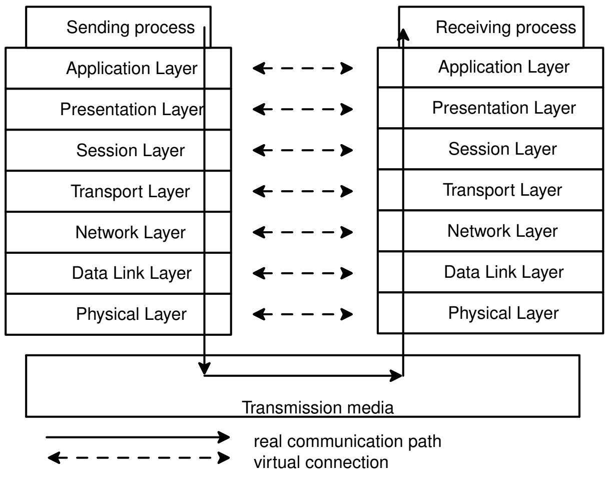


Figure 2.3: The ISO/OSI model [Hei02, cmp. p. 17 – 21]

The physical layer defines the communication media (cable, air, fiber, etc.). Data is transmitted as a bit stream. Typical physical layer devices are modems, repeaters and transceivers[1].

The data link layer is used to ensure flawless data transmission between nodes. The bitstream from layer 1 is packed into frames. The second layer can further be divided into two sublayers:

- the 'media access control layer' generally implements IEEE-Standards 802.3 (CSMA/CD), 802.4 (Token Bus), 802.5 (Token Ring) and FDDI and
- the 'logical link control layer' offers a unified interface for network layer protocols. This part of the data link layer is media independent.

The separation into two sublayers guarantees the interchangeability of media-access technologies. So it is e.g. possible to operate an IP network on top of ethernet, token-ring, etc. without any further intervention. Flow control is typically offered by the data link layer. Bridges and switches are layer 2 devices which are used for interconnection between network segments (see 2.5.4.2 for further explanations).

The network layer offers routing capabilities. Routing enables inter network connections. Many networks can be combined to a big logical network. The most common protocol on this layer is the internet protocol (IP). This protocol offers the possibility of a logical address scheme which can be configured by a network administrator.

The transmission layer provides a transparent data transmission between endsystems. Protocols on this layer can be categorized into connection-oriented and connection-less protocols. Connection-oriented protocols ensure the correct reception of segments. TCP is a connection-oriented protocol. UDP is a connection-less protocol. Compared to TCP the latter offers better performance, but there is no guarantee that packets have arrived at their destination (but this functionality could be implemented by upper layer protocols as is the case with NFS).

Layer 5 to 7 protocols can only be delineated vaguely. The major task of layer 5 protocols is to convert data for a convenient presentation. Application specific tasks (electronic mail, file transfer, etc.) are typically performed by layer 7 protocols. Characteristic protocols are RPCs of the NFS protocol (layer 5), ASN.1 (layer 6) and application specific protocols such as FTP, SMTP and HTTP (layer 7).

2.4.2 Topologies of real-time communications systems

The physical arrangement of nodes in a network is called the topology. [Flo05, cmp.] lists and displays four of them, whereas [WB05, cmp. p. 260] describes the advantages and the disadvantages of these topologies.

[1]this is actually a made-up word: transmitter-receiver

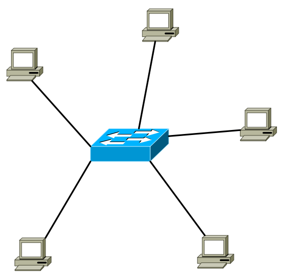


Figure 2.4: Star topology

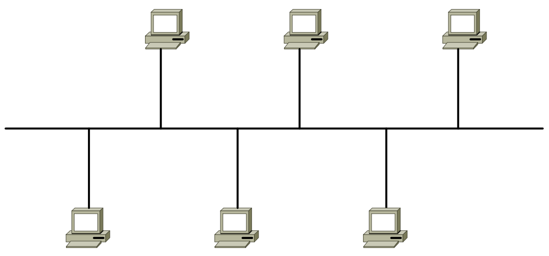


Figure 2.5: Linear Bus topology

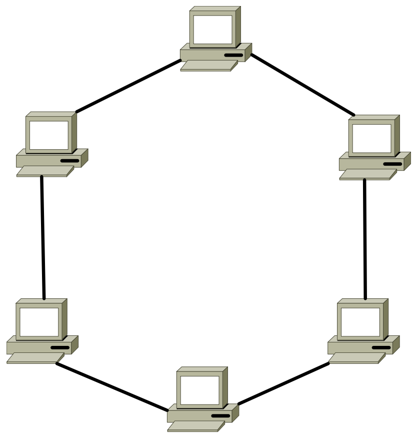


Figure 2.6: Ring topology

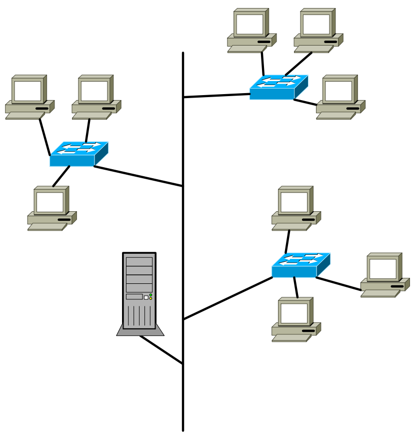


Figure 2.7: Tree topology

2.4.2.1 Star

Figure 2.4 shows a typical star topology. The central station is also called the master. In case of ethernet this is usually a hub or a switch which is connected to all other stations, the so called slaves. Each data transmission occurs over the master which in turn can get heavy loaded due to this fact. In case of a failure on the master no communication between the other nodes is possible. Therefore the star topology can be seen as a single point of failure system.

2.4.2.2 Bus

All slaves are connected to a shared medium, the so called bus (see figure 2.5). Messages on the bus can be received by all connected nodes. In case a slave detects its address in a received message it processes the contained message data.

2.4.2.3 Ring

Connections are constraint as point-to-point links and run from one node to the next (the structure is depicted in figure 2.6). As in the bus topology, only messages which are addressed to the specific node get processed. The ring topology is a multiple point of failure system. One failure in the ring leads to a system in which communication does not work at all. Compared to the star topology response times are longer in general because direct communication with the master is not possible in most cases.

2.4.2.4 Tree

A tree structure is a combination of star and bus topology (figure 2.7). Participants in one branch are able to communicate without the root communication object. With this topology it is possible to constrain communication hierarchies.

2.4.3 Media access control communication systems

Four different methods for media-access have established themselves in recent years:

- Polling
- CSMA/CD and CSMA/CA
- Token-Passing
- TDMA

2.4.3.1 Polling

Polling is realized in a master-slave relationship in most cases. Each slave is cyclically requested by the master to transfer data.

The arbitration function determines the strategy the master uses to manage media-access for all the nodes. E.g. the master queries all nodes sequentially for transmission requests. After that a priority algorithm in the master determines when a specific participant is allowed to use the bus by itself for a specific amount of time for its data transmissions. Nodes can also be queried many times in one cycle to reduce message response time.

[WB05, cmp. p. 263] states that the polling strategy can be implemented easily. One of its downsides is that the master's queries reduce the transmissibility of the medium. As such bandwidth gets wasted. An advantage is, that worst-case reaction time can easily be calculated (formula 2.2). The time for a communication cycle (t_{cycle}) is proportional to the number of participating nodes.

$$t_{worst}\ [s] = 2 * t_{cycle}\ [s] \tag{2.2}$$

Figure 2.8 to 2.10 shows how t_{worst} is composed. One requirement for the worst transmission time is that all nodes in the network have to transmit data at the same time. The first figure (2.8) displays that the leftmost slave initiates its request to transmit data (1a) when the second leftmost slave is acquired to disclose its request for data transmission (1b). The leftmost node has to wait until all other slaves have been acquired. This requires the period of one cycle time = t_{cycle} (figure 2.9) (2a). After that the second leftmost slave is allowed to transmit its data (2b). Processing all other nodes also lasts for the period of one cycle time t_{cycle}. Then the leftmost slave is allowed to transmit data (figure 2.10) (3). This sums up to a total of two times the cycle time t_{cycle}.

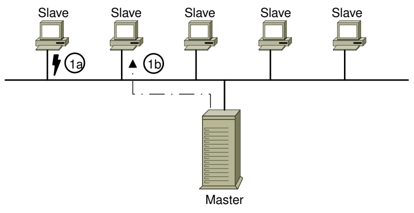


Figure 2.8: Transmission request of the leftmost node

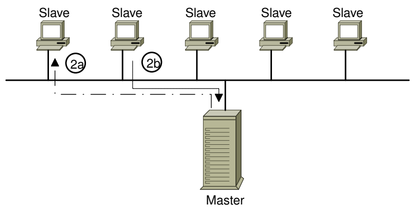


Figure 2.9: The master detects the transmission request

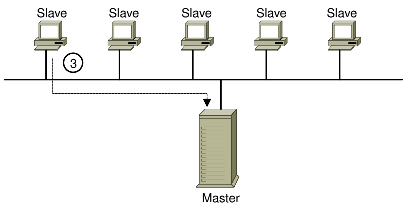


Figure 2.10: Data transmission of the leftmost node

The process image which represents the current state of Input/Outputs in an automated system is set in a polling fashion after the program cycle in a PLC . PLCs are used as controllers for industrial scenarios. Programs in a PLC are usually executed in a cyclic way, thus execution occurs in real-time.

2.4.3.2 CSMA/CD

Multiple users share the same medium. Media access therefore has to be limited in some way. The technology used by Ethernet is called "CSMA/CD", which is an abbreviation for "Carrier Sense Multiple Access with Collision Detection" [Gra03, cmp.]. "Carrier sense" means that nodes which want to access the medium, listen for ongoing transmissions. Collisions occur when two or more nodes want to access the medium at the same time. This case is recognized and leads to the "Collision Detection" part in CSMA/CD.

Media access is described in [oEI05, p. 62–63]. When transmitting a frame the sender "listens" to the medium (carrier sense) if it is available for transmission. If so, the sender places its data onto the bus and continues to listen if a collision happened. In the case of a collision the sender continues to transmit data (jamming signal) so that other nodes can detect the collision too. The number of bits transmitted in this case is called "jam-size". Usually 32 more Bits are transmitted after a collision [Gra03, cmp.]. The internal attempts counter is incremented. If the value of the increments counter reaches a specific value, the node aborts its request to transmit data (excessive collision error). After that the backoff is calculated. That is a random time period the sender has to wait for a retry. This process is depictured as a flowchart in figure 2.11.

The receiving procedure is pictured in figure 2.12. As with the transmission of frames, collision detection occurs during reception too. In case a frame has been corrupted during transmission it has to be ignored by the operating system and the ethernet hardware. This is recognized when the actual frame size differs from the frame size given in the ethernet header.

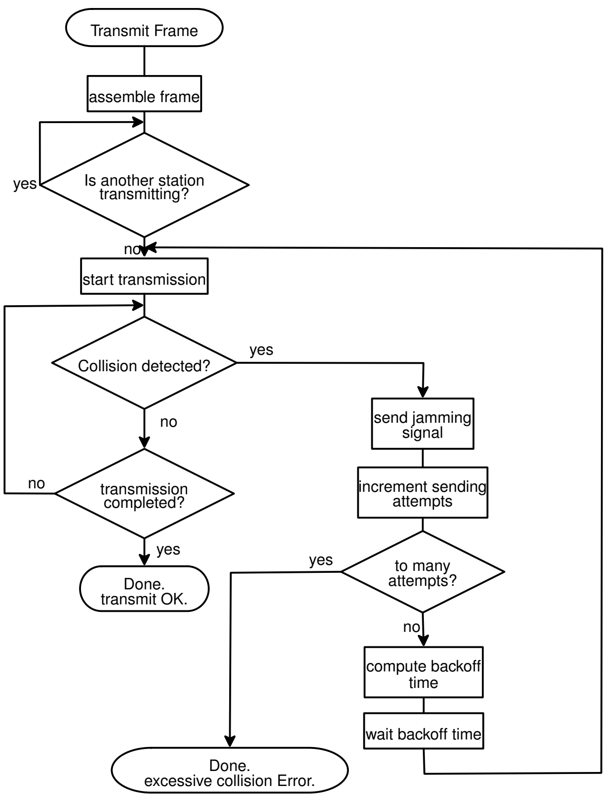


Figure 2.11: Frame transmission [oEI05, cmp. p. 62]

2.4.3.3 The segment size

A parameter regarding the dynamics of collision detection is the so called "slot time". It describes three aspects of collision handling [oEI05, p. 101]:

1. The acquisition time of the medium describes how long the medium has to be busy to be able to detect a collision.

2. The maximum length of a frame fragment in case of a collision. Transmission is aborted immediately if a collision occurred.

3. The scheduling quantum in case of a retransmission which has been caused by a collision. As mentioned above the random time period, the backoff time, is a multiple of the slot-time (51.2 μs in case of 10Mb ethernet).

Figure 2.13 demonstrates how the medium is acquired. When the first bit of transmitted data arrives at the most distant node in the same network segment it can be deemed to be busy (see digit "1" in the figure). In case of a collision the signal of the collision has to get back to the sender. In the worst case scenario this is the distance between the most distant nodes in the network segment (see digit "2" in the figure). Data has to

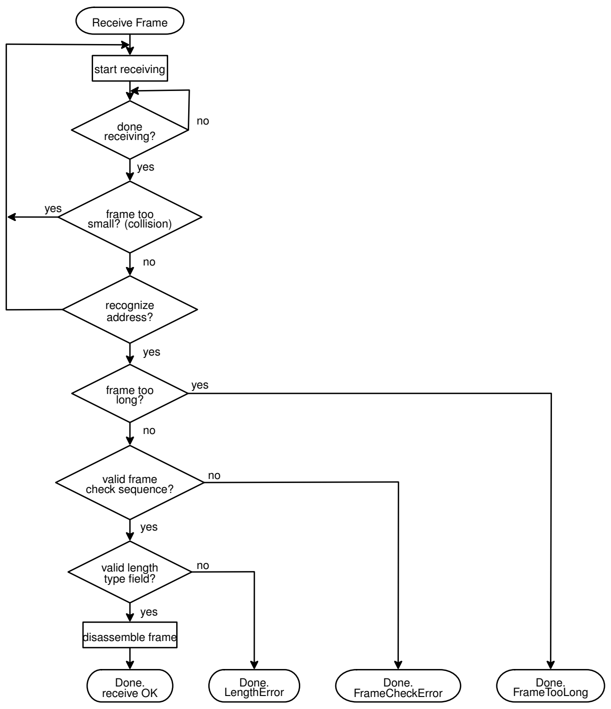


Figure 2.12: Frame reception [oEI05, cmp. p. 63]

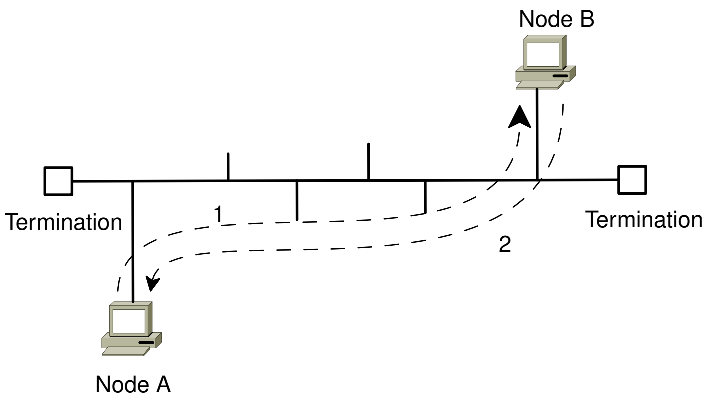


Figure 2.13: Influence of the slot time [Fis04, cmp. p. 28]

be transmitted as long as it takes the signal to propagate over the distance of “1” and “2” added together to be able to detect a collision correctly.

Collisions only occur in half duplex CSMA/CD networks. Four parameters collude therefore when determining the size of such an ethernet segment:

1. physical distance of the most distant nodes s_{phys}
2. the slot time t_{slot} (which can be equaled to the minimum frame size)
3. the velocity of propagation v_{prop}
4. the bitrate b

5. the minimal frame size f_{min}

This coherence can be expressed in a formula 2.3. The slot time (t_{slot}, formula 2.4) is the relation between the minimal frame size (f_{min}) and the bit rate (b).

$$s_{phys\ [m]} = t_{slot\ [s]} * v_{prop\ [m/s]} * \frac{1}{2} \tag{2.3}$$

$$t_{slot\ [s]} = \frac{f_{min\ [bit]}}{b_{\ [bit/s]}} \tag{2.4}$$

With 10 Mb Ethernet which has a minimal frame size of 512 Bits, a velocity of propagation of 2/3 * c [Hig98, p. 240] (with c being light speed = 3 * 10^8m/s) and taking into consideration the two way factor of 1/2 the physical size is:

$$s_{phys} = \frac{512}{10 * 10^6} * \frac{1}{3} * c * \frac{1}{2} \approx \underline{2500m} \tag{2.5}$$

Collisions only occur in half duplex ethernet. Switches and bridges overcome this situation. A requirement for a collision free ethernet network is microsegmentation. Here each device in a network has its own switch-port. In this case other factors like queueing in the switch fabric influence the real-time behavior. The random backoff is one of the factors that leads to non-real-time behavior of ethernet networks [JJ01, cmp.]. Real-time ethernet has to overcome this weak spot. This is shown in 2.4.3.

2.4.3.4 Performance analysis of CSMA/CD networks

The performance of a network with CSMA/CD media-access depends largely on three factors: S (the throughput), a (the effective channel length) and G (the offered load).

The effective channel length "a" can be seen as the ratio of propagation time (t_{prop}, e.g. 2/3 of light speed in copper cables) and the transmission time (t_{trans}) of a frame (see formula 2.6). The effective channel length decreases with the transmitted frame-length [Hig98, cmp. p. 238].

$$a_{\ [1]} = \frac{t_{prop\ [s]}}{t_{trans\ [s]}} \tag{2.6}$$

In [Hig98, cmp. p. 259] throughput "S" is defined in formula 2.7. The offered load "G" is defined in formula 2.8.

$$S_{\ [bit/s]} = \lambda_{\ [packet/s]} * x_{\ [bit/packet]} \tag{2.7}$$

$$G_{\ [bit/s]} = \gamma_{\ [packet/s]} * x_{\ [bit/packet]} \tag{2.8}$$

The average rate of actually transmitted traffic, including new and retransmitted packets is to be γ packets/s. Traffic in an error-free channel is generated with an average rate of λ packets/s. The packet length is x bits/packet.

[JH86, p. 177] defines the terms throughput "S" and offered load "G" as follows:

- S – “Average number of successful transmissions per packet transmission time, P”
- G – “Average number of attempted packet transmissions per packet transmission time, P”

These characteristic values are calculated with formulas 2.9 and 2.10 from [JH86, cmp. p. 158] and [JH86, cmp. p. 177] respectively.

$$S_{[1]} = \frac{\lambda_{[packet/s]} * \overline{X}_{[bit/packet]}}{R_{[bit/s]}} \tag{2.9}$$

In case of formula 2.9 and 2.10 “S” and “G” are a dimensionless values. $\overline{X}$ expresses the number of bits per packet and R is used as a symbol for the channel bandwidth. P in this case is the probability for a good transmission. It can further be stated as the probability of no additional transmissions in the vulerable interval of length $2P$ (in words: *2 times the packet transmission time*). The probability depends largely on how media access is implemented. “In the case of random access methods, the assumption of Poisson traffic is extended to include the retransmitted packets as well as new arrivals to the network.” [JH86, p. 170]. Formula 2.11 which has been taken from [JH86, p. 177] states how formula 2.10 is applied and throughput in a random access networks (pure ALOHA access scheme) can be calculated. With this access scheme collisions may occur. Therefore the probability for good transmission depends on offered load.

$$S_{[bit/s]} = G_{[bit/s]} * P\{good\,transmission\}_{[1]} \tag{2.10}$$

$$S_{[bit/s]} = G_{[bit/s]} * e^{-2G}_{[1]} \tag{2.11}$$

Thus the correlation between publication [Hig98] and [JH86] can be stated with the equation given in formulas 2.12 and 2.13.

$$\begin{aligned} x_{[bit/packet]} &= \frac{S_{[bit/s]}}{\lambda_{[packet/s]}} = \frac{G_{[bit/s]}}{\gamma_{[packet/s]}} \Rightarrow \\ & \frac{S_{[bit/s]}}{G_{[bit/s]}} = \frac{\lambda_{[packet/s]}}{\gamma_{[packet/s]}} = P\{good\,transmission\}_{[1]} \end{aligned} \tag{2.12}$$

$$x_{[bit/packet]} = \overline{X}_{[bit/packet]} \tag{2.13}$$

Throughput in an ethernet network depends largely on the average network load and frame size [JH86, cmp. p. 336]. Figure 2.14 shows the correlation between throughput and offered load in dependency of the used media access scheme. Larger frames (a = 0.005) gain a better throughput in networks with similar media access under heavy load (more offered load) than smaller ones (a = 0.05).
Figure 2.15 shows the correlation between “S” and and “G” with TDMA (see 2.4.3.8 and 2.10.2 for further explanations) under idealized central control. In this case the relation between throughput and offered load can be stated as shown in formula 2.14.

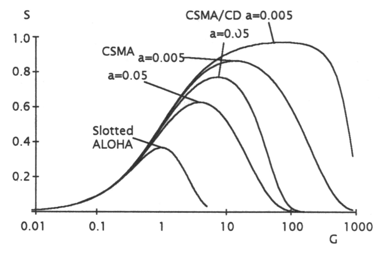


Figure 2.14: Throughput and offered load in CSMA and CSMA/CD networks [Hig98, p. 268]

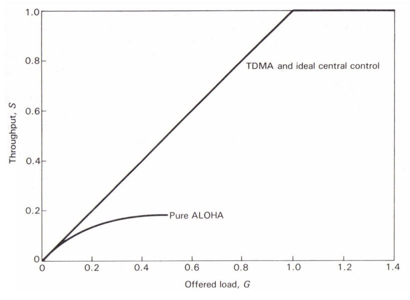


Figure 2.15: Throughput and offered load with different media access schemes [JH86, p. 187]

$$G_{[bit/s]} = S_{[bit/s]} \tag{2.14}$$

Thus the input and output packet rates are equal for the entire network. The channel is blocked if G becomes equal or greater than 1. The network can remain in this state only if packets are dropped [JH86, p. 184].

Latency or the average transfer delay which is defined as "the time from arrival of the last bit of a packet into the station of a network until the last bit of this packet is delivered through the network to its destination station" [JH86, p. 159]. Ethernet is a non-deterministic network and as such the average time it takes to convey a data frame from its source to its destination is unpredictable. The latency of a packet largely depends on the current network load. The likeliness of collisions rises with an increasing number of frames on the bus (Figure 2.16)

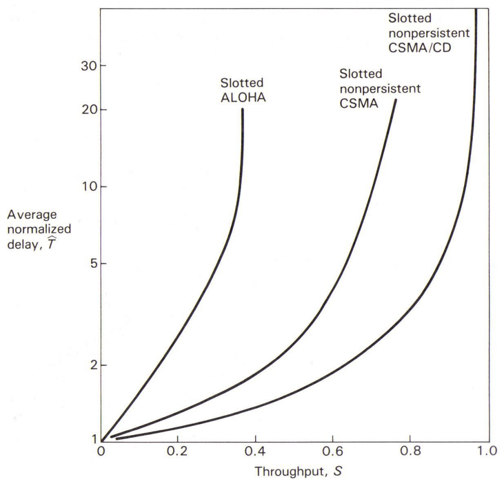


Figure 2.16: Transmission delay in dependency of throughput [JH86, p. 338]

Normalized average transfer delay can be expressed with formula 2.15.

$$\hat{T}_{[1]} = \frac{T_{[s/packet]}}{\frac{\overline{X}_{[bit/packet]}}{R_{[bit/s]}}} = \frac{R_{[bit/s]} * T_{[s/packet]}}{\overline{X}_{[bit/packet]}} \tag{2.15}$$

T expresses the average transfer delay per packet. The time it takes to convey a packet over the transmission channel is then $\overline{X}/R$. Another interpretation of the formula is that the normalized average transfer delay is the relation between the time it takes to transfer one bit from one end to the other and the time it takes to transfer one packet. The value of $\hat{T}$ is therefore dimensionless.

2.4.3.5 CSMA/CD in real-time applications

Ethernet has been designed to offer equality for all participating nodes in a network [Vir07]. Communication relations often change in such a network and therefore the timing-characteristics are not guaranteed to be of a cyclic fashion.

[LM06, cmp. p. 10] states that ethernet is convenient for accurately timed data transmission in most of the cases but not for simultaneous data exchange which is required for synchronization purposes (see 2.3.1.2). The jitter of classical ethernet data transmission is too unpredictable to be able to transport real-time data. This can be led back to unforseeable waiting times in packet or frame queues of network devices such as bridges, switches and routers.

2.4.3.6 CSMA/CA

Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) extends the CSMA/CD access scheme in the fact that no data is lost due to collisions. Like in CSMA/CD nodes listen to the medium until it is free for a transmission. A station has to defer its transmission when the medium is “busy”. If the medium is “idle”, the station is permitted to transmit data [Wik07d, cmp.].

Collisions are avoided with a special kind of signal transmission [WB05, p. 264] which is known as “wired and”. Here a digital ’0’ has precedence over a sent ’1’. So a digital ’1’ can be overwritten by a digital ’0’. Thus messages do not get destroyed by two or more nodes transmitting at the same time.

CSMA/CA implies a priority scheme. With the algorithm mentioned above a message containing more leading zeroes (’0’) than a “competing” message achieves itself. This makes CSMA/CA a valid media-access algorithm for real-time systems.

CSMA/CA is used in popular real-time communication systems such as CAN . During a data transmission nodes listen to the medium and compare their data to the data on the bus. In case of an irregularity the client stops to send data. The first part of a CAN message is the message identifier which is used by the arbitration scheme. Prioritization is accomplished by the message itself and not by the participant in the communication [WB05, p. 280].

A different field of application are wireless networks. Here a sender cannot listen to the channel during transmission thus collision detection is not possible. Another reason is the hidden station problem [Fis04, p. 22]. In this case node A is in range of receiver B, but not in range of sender C. Therefore node A cannot know, that receiver B and sender C are exchanging data.

2.4.3.7 Token-Passing

Media-access is deterministic with token-passing. Token ring usually operates as a multi-master network. A token, which can be understood as a bundle of bits, circulates in a logical ring between the masters. Only the master which holds the token is allowed to occupy the media for a specific time period. In case a master does not need to exchange data, the token is passed on to the next master [WB05, p. 264].

The possibility to build multimaster systems with communication directly between masters and a master with its associated slaves is an advance over other technologies such as ethernet. The response time based on the request to transmit data is predictable.

Token-Ring is a local area network technology and can be seen as a field of application for the token-passing media access. It was developed and promoted by IBM in the early 1980s and standardized as IEEE 802.5 by the Institute of Electrical and Electronics Engineers [Wik07p, cmp.]. The introduction of 10BASE-T ethernet (see 2.5.1.1) was a big success leading to IBM not promoting token ring anymore. The format of a token-ring token is shown in figure 2.17. Standardized bitrates for 802.5 token ring are 4Mb/s, 16Mb/s, 100Mb/s and 1Gb/s [Wik07p, cmp.]. Figure 2.18 shows the typical structure of a token-ring network (as shown in [oEI98, p. 17]).

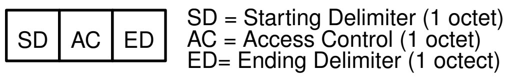

SD
AC
ED
SD = Starting Delimiter (1 octet)
AC = Access Control (1 octet)
ED= Ending Delimiter (1 octect)

Figure 2.17: The format of a token (as shown in [oEI98, p. 20])

With the introduction of switched ethernet, token ring lagged badly behind the former technology in both performance and reliability. Higher sales of ethernet components drove down prices further which led to a compelling price advantage over token ring.

2.4.3.8 TDMA

[Her91, cmp. p. 5pp] enumerates two categories of real-time systems:

- Event triggered systems with which processing activity is triggered through the occurrence of a specific event
- Time triggered systems which initiate activities periodically in real-time at predetermined points in time

Time Division Multiple Access (TDMA) breaks communiation down into so called cycles. These cycles start at predefined points in time and are further subdivided into slots. Slots represent a span in which a specific node in the network is allowed to access the medium. This approach is therefore a deterministic media-access technique. The

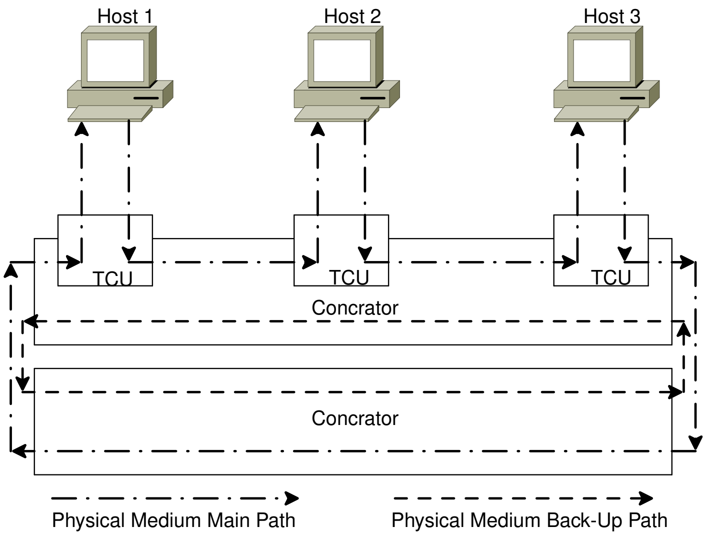


Figure 2.18: Physical structure of a token-ring network

channel is reserved for each node for a specific time period. Media-access happens mutual exclusive. In general there are two variants [Wik07l, cmp.]:

Statical assignment of time slots. This method is also known as "synchronous time division" (STD). Before participants begin to communicate they are assigned a fixed length time slot. These assignments can in turn happen in two ways [WB05, p. 265]:

1. time slots of constant length: one cycle is split up into time slots of equal size: $t_i = T/n$ with n being the number of nodes and T being the time period for a communication cycle. This method is easier to implement but in case a node has nothing to transmit, bandwidth is wasted. The start time of a specific time slot evaluates to: $t_{Start_i} = i * t_i = i * T/n$
2. time slots of variable length: here communication participants are assigned time slots of different sizes. This leads to a better allocation of available time. Nodes with higher requirements get more time to transmit their data, nodes with lower requirements still have enough time to send their data.

The cycle time can always be calculated as:

$$\sum_{i=1}^{n} t_{i\ [s]} \leq T_{[s]} \tag{2.16}$$

Dynamic assignment of timeslots overcomes the disadvantage of unused or wrong sized time slots. Time slots are used on demand. To be able to differentiate between data of the specific nodes channel information has to be added. This leads to an overhead that has to be taken into count when calculating the available bandwidth. This method is also known as "label-multiplexing" [Wik07l, cmp.].

2.5 Classical Ethernet

Ethernet is a large, diverse family of frame-based computer networking technologies for local area networks (LANs). The name comes from the physical concept of the ether. It defines a number of wiring and signalling standards for the physical layer, through means of network access at the Media Access Control (MAC)/Data Link Layer, and a common addressing format [Wik07f].

The technology has been developed in the late 1970s [Kut02, cmp.] by the American company "Xerox" in Palo Alto, California. It has been demonstrated with 100 computers which where connected together with a transmission speed of 3MBits/s.

Ethernet is now standardized by the Institute of Electrical and Electronics Engineers (the so called IEEE) in standard 802.3 and its sub standards. The ease of installation, its flexibility, versatility and the simple operation were keys to success of this technology [Hei02, p. 161]. [BS07, cmp.] states that ethernet was cheap compared to different technologies such as IBMs token ring technology.

2.5.1 Structure and Topology

The original Ethernet used a coaxial cable which can also be seen as a logical bus structure. Today the star structure is also common with Ethernet as a network technology.

2.5.1.1 Ethernet Media and Connectors

Cabling and connectors are currently standardized from 10Mb/s up to 10Gb/s. The ethernet standard (IEEE 802.3 Section 1, [oEI05, p. 14–17]) lists more than 30 of them. The most common were taken from [B. 04, p. 206, 207] and shown in table 2.2.

NAME	MEDIA	MAX. LENGTH	TOPOLOGY
10BASE2	50-ohm coaxial (thinnet)	185m	Bus
10BASE5	50-ohm coaxial (thicknet)	500m	Bus
10BASE-T	EIA/TIA CAT 3,4,5 UTP	100m	Star
100BASE-T	EIA/TIA CAT 5 UTP	100m	Star
100BASE-FX	62.5/125 multimode fiber	400m	Star
1000BASE-CX	STP	25m	Star
1000BASE-T	EIA/TIA CAT 5 UTP	100m	Star
1000BASE-SX	62.5/50 micro multimode fiber	275m (62.5 micro) 550m (50 micro)	Star
1000BASE-LX	62.5/50 micro multimode fiber	440m (62.5 micro fiber) 550m (50 micro)	Star

Table 2.2: Ethernet cabling and connectors

2.5.2 Ethernet and Media Access (CSMA/CD)

Ethernet uses CSMA/CD for media-acces. This technology has been described in 2.4.3.2.

2.5.3 Structure of ethernet frames

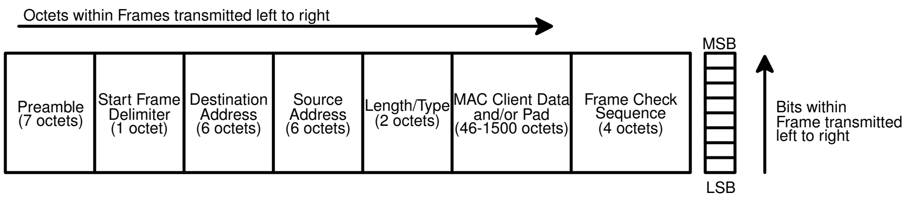

Octets within Frames transmitted left to right
Preamble (7 octets)
Start Frame Delimiter (1 octet)
Destination Address (6 octets)
Source Address (6 octets)
Length/Type (2 octets)
MAC Client Data and/or Pad (46-1500 octets)
Frame Check Sequence (4 octets)
MSB
LSB
Bits within Frame transmitted left to right

Figure 2.19: Structure of an ethernet frame [oEI05, cmp. p. 49]

7 octets make up the preamble of an ethernet frame. It is used for bit synchronization at the receiver of the frame and consists of a bit pattern of “101010101 ...”. The start frame delimiter signals (bit pattern: “10101011”) the receiver the beginning of MAC data. If necessary the data part is padded so that ethernet frames are at least 64 Bytes long. The tallest frame is 1518 Bytes long (without the preamble and the start frame delimiter octets).

2.5.3.1 Addressing

Source and destination of an ethernet frame are addressed via a MAC-Address. It is 48 Bits long and therefore officially called MAC-48. The address space is 2^{48} individual addresses [Wik07j, cmp.].

The first bit of a MAC-48 address determines, if this address is a group (1) or an individual (0) address. Group addresses identify one or more, or all of the stations connected to the LAN. The administration of MAC-Addresses can be divided into two groups [oEI05, cmp.]:

universally administered address is uniquely assigned to a device by its manufacturer. The first three bytes identify the manufacturer. This part is also known as Organizationally Unique Identifier (OUI). The last three bytes are assigned by the organization refered by the OUI. The only requirement is that the MAC address is unique.

locally administered address is assigned to an interface by a network administrator. In this case the MAC address does not contain an OUI.
The second bit of the MAC address decides whether it is a universally administered address (0) or not (1).

2.5.4 Typical Ethernet devices

2.5.4.1 Repeater and Hub

The size of an ethernet segment depends largely on two factors:

- signal degradation
- timing reasons (the slot time generally)

The former can be overcome by so called repeaters. These devices take the signal they receive to retransmit it on the other port. Beside the retransmission the signal gets refreshed and retimed at the bit level. Repeaters operate at the physical layer of the OSI model.

Hubs can be understood as multiport repeaters. The number of ports is currently typically 4 to 24. Hubs change the network topology from a linear bus to a star structure. Three major groups arose over the time ([B. 04, p. 215]):

Active hubs need power to amplify (refresh) the signal before passing it out to the other ports

Intelligent hubs include a microprocessor chip and diagnostic capabilities

Passive hubs are only used to share the media (no refresh)

[B. 04, p. 214] states that no more than five network segments can be connected end-to-end using four repeaters (or hubs), but only three segments can have hosts on them. This rule has no validity in switched networks.

With these devices collisions may still occur because the medium is just shared between all participating nodes. It is therefore only possible to transmit data half duplex.

2.5.4.2 Switch and Bridge

The following paragraphs apply to switches as well as bridges. For simplicity in some cases only bridges are mentioned.

Switches and Bridges operate at the data link layer of the OSI model and are used to link different network segments. Bridging is standardized in the 802.1D standard. Decisions are met intelligently. Switches and bridges have an address table (thus they are more intelligent than hubs or repeaters) installed. This table is consulted when a frame arrives at a switch-port. Three cases are differentiated [Fai01, cmp.]:

- The address is not found in the table. This may happen if this is the first frame that is received from a station or if the address record was deleted because it was too old. Another reason for this case is that the bridge ran out of addresses. This occurs in case of MAC-flooding, a network attack to let a bridge behave like a hub (sending out received frames on all switch-ports). The received frame is sent out on all ports except the one the frame was received on. This is also called flooding.

- If the address is found in the interface table of the receiving port it is discarded because it already has been received by the destination.
- If the address is found in an interface table different then the port on which it was received, the bridge forwards the frame to the port associated with the address.

In a pure switched network no collisions are possible. The medium is only shared in the bridge's fabric. Therefore a pure switched network offers one collision domain per switch-port. Such an ethernet network is still not real-time capable because overload may still happen. Additionally the behavior of bridges in case of congestion is not standardized. Most bridges hold back messages in a buffer of limited size. In case of a full buffer new messages are discarded.

The main difference between switches and bridges is the number of available ports. Bridges usually connect 2 ethernet segments [B. 04, cmp. p. 219].

2.5.5 Limitations of Classical Ethernet

Classical ethernet is not real-time capable as mentioned in 2.4.3.2. Using this technology is still useful when the system is not required to fulfill real-time requirements.

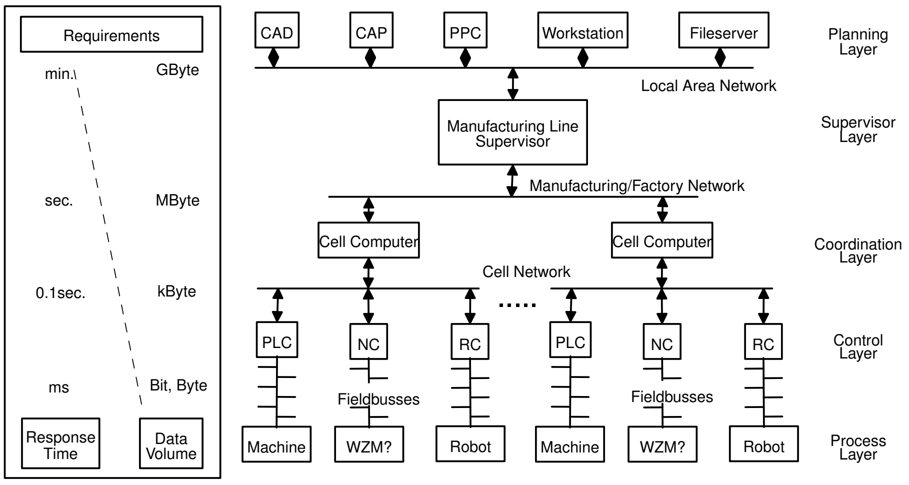


Figure 2.20: A typical automation network

Figure 2.20 shows a typical automation network [WB05, p. 255]. Devices are arranged and connected in a hierarchical fashion. Requirements on real-time data transmission are different. On the highest layer products are defined and drafted. Production Planning and Production Control (PPC) and Computer Aided Planning (CAP) tools are used to create and document the automated system. Data from this layer is passed on to the manufacturing line supervisor. This device handoffs its processed data to the cell computers which in turn distribute their data in real-time to tool-machines and robots. The latter is used to manufacture products and pass feedback data for visualization purposes back to devices in the higher layers.

The required response time is inversely proportional to the transmitted data volume. On the topmost layer transported data is often based on file transfers or blocks of data. The only timing-requirements are demanded by humans who use this system. On the lowermost layer only small data packets are exchanged. This is where real-time requirements have to be fulfilled. At present the trend goes towards ethernet-based real-time networks which use cheap off the shelf hardware to gain a continuous concept in factory networks.

2.6 Ethernet as an industrial communication system

Currently the trend goes towards the deployment of real-time ethernet as a replacement of fieldbus systems in automation technology. This has the advance of a consistent networking scheme in company and manufacturing networks. Fieldbus systems define the requirements which industrial ethernet has to meet in the future. Therefore a small introduction is given here.

2.6.1 Fieldbusses

"A fieldbus system is an industrial communication system that is used to connect a multitude of field devices such as sensors, actuators and drives."[Wik07g]

The intention for the development of fieldbusses was basically the reduction of cabling effort. Before the introduction of fieldbusses devices where connected to the controller directly. This was also called "parallel" cabling scheme. After that a "serial" cabling scheme established, many devices could be connected to the controller via one cable. Thus fieldbusses reduced cabling costs enormously.

On the other hand fieldbus systems are more complicated than the parallel cabling scheme. Data has to be transmitted according to a given protocol. During the years many fieldbus protocols have established, each for its own purpose. Protocols are not compatible among each other. Devices which meet fieldbus standards can be very expensive and response time is in general worse than with direct connections. This can be led back to the encapsulation of data into frames or packets and competiton between devices when accessing the transmission medium.

2.6.1.1 Common fieldbus systems

Table 2.3 lists a decent list fieldbus systems which have been developed. Each fieldbus protocol has specific characteristics for specific deployment scenarios. E.g. Interbus can be used to transmit data over distances of up to several kilometers, Safetybus can be used in hazard areas where e.g. explosions can occure and Profibus can be used where short response times are required, e.g. for motion control.

NAME	MAIN FIELD OF APPLICATION	STANDARD
CAN	automotive automation	EN 50325-4
Profibus	industry automation	IEC 61158/IEC 61784
Interbus	industry automation	EN 50254/IEC 61158
ASi	industry automation	EN 50295/IEC 62026-2
Modbus	industry automation	Modbus-TCP as pre-standard IEC PAS 62030
LON	central building control systems	EN14908/ANSI/EIA-709.x EIA-852
SERCOS Interface	drive engineering	IEC 61491/EN 61491
SafetyBUS	machine safety	not standardized

Table 2.3: Common fieldbus systems

2.6.1.2 Intentions for the development of fieldbusses

Fieldbusses have also been developed to replace the existing 4-20mA standard for analog data transmission. In the automotive industry for example the number of devices installed into a car increased, many cables (up to 2 kilometers per car) where necessary for interconnection. This led to the development of fieldbusses.

2.6.1.3 Requirements

- Real-Time capabilities: data transmission has to be completed within a predefined time period. The controlled process establishes rules regarding the timing requirements.

- Requirements regarding the physical layer: fieldbusses are usually deployed in an industrial environment. Here extreme conditions occur, such as electrical and magnetic interference, vibrations, heat, low temperatures, moistness, UV-rays and electro-magnetical fields near welding machines and huge engines. The physical layer involves cabling and plug standards.

- On the second layer, media-access is regulated. Especially when packets hold small amounts of data in the dimension of some bits, transmitted packets have a size of some bytes. Usually 75% of transported data originate from cyclic transmission (polling, as described in 2.4.3.1), the rest (25%) comes from random events. Framing (packing bits into groups of predifined size), error detection and error prevention also occur on layer 2. The latter is important in fields with extreme conditions (as described above in the requirements on layer 1).

- Devices connected to fieldbusses have to be interoperable: this goal can be reached with standards with which the device have to comply. As well software as hardware for industrial communications is standardized. Therefore it is possible to combine and exchange devices which have been manufactured by different vendors

- The possibility for diagnosis: early error detection and immediate support in case of a failure are often required when gaining high manufacturing quantities

- Meeting international guiding principles: protecting the environment in general, e.g. the EMC-compatibility

These requirements have been taken from [WB05, cmp. p. 273] and [Kut02, cmp. p. 21].

2.6.2 Ethernet

The main motivation for using ethernet for real-time purposes arose because devices are wide spread and generally cheap because of the big competition between manufacturers. The ethernet standard (802.3 and successors) is freely available. Licence fees are comparatively cheap when compared to that of fieldbusses. As a result internet technologies such as the web-technology (the HTTP in general), file transfers based on the FTP and electronic mail, which is based on the SMTP , can now be used easily in industrial environments.

2.7 Classification of requirements

A real-time system consists of hard- and software components. These components gather and process internal and external data and events. The timing behavior of such systems depends on all layers it consists of. Figure 2.21 shows a first simplified model which is accurately enough for demonstration of the problem domain. Hard real-time communications can only be achieved when all layers of this model are real-time capable [WB05, p. 130].

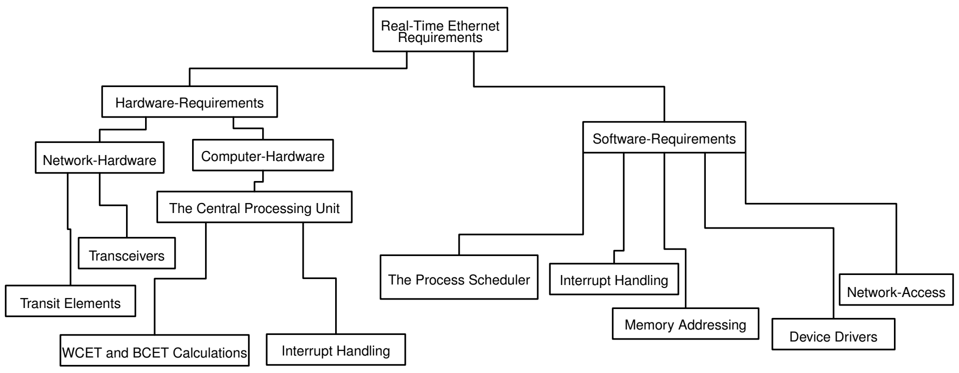


Figure 2.21: The real-time ethernet requirements model

2.8 Hardware requirements

2.8.1 The Central Processing Unit

“The processor controlls the operation of the computer system and is used to do the data processing. If there is only one processor available it is often called *central unit* or *central processing unit* (CPU).” [Sta03, p. 25]

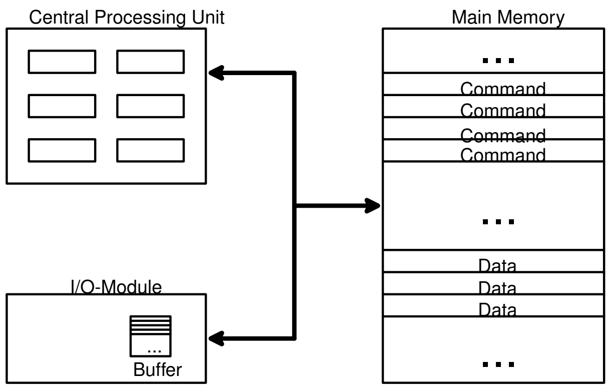


Figure 2.22: The main components of a computer system [Sta03, cmp. p. 26]

Figure 2.22 shows in a very simplified fashion the main parts of a computer system. It also shows different memory technologies working together. Memory technologies can be organized in a hierarchical style as shown in figure 2.23. Faster memory is usually more expensive [SGG05, cmp. p. 9]. The Fastest memory is built directly into the processor, the so called registers. Main memory is much slower compared to registers. Buffering or caching means to map state of in- and outputs into a specific area in main memory. Access to I/O -devices of the computer system is always buffered, to gain multi-tasking abilities. Parts of memory with lower speed is often cached in faster memory due to response-times improvements. This leads to unpredictable execution times of tasks as it is very difficult to determine whether the requested data can be found in the cache or whether it has to be requested directly in slower memory. The former is called “Cache-Hit” and the latter is called “Cache-Miss”.

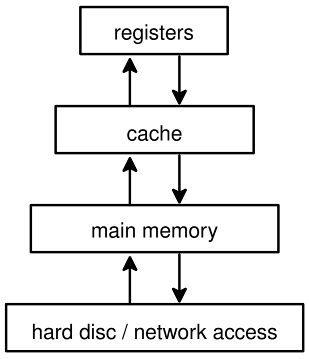


Figure 2.23: Storage-device hierarchy [SGG05, cmp. p. 9]

Another reason for unpredictable execution times comes from improvements in processor technology as described in [WB05, cmp. pp. 145]:

- Caching: access to slower technologies can be boosted when parts of already processed data gets stored in fast memory. This can be seen as a work-around for the “von-Neumann-bottleneck” which is an alternative name for the hierarchical organization of memory.
- Pipelining: means to fractionalize complex execution commandos into simpler

ones. These simple commandos can be processed by the in a parallel fashion. E.g. fetching the commando, decoding the commando, fetching the operators for the commando, executing the commando and storing the results can be processed like automotive parts on an assembly line.

- Speculative execution: if the result of a jump condition is not available in due time, the processor guesses how to proceed. Currently the probability for correct decisions is approximately 90%. This goal is reached when the results of the estimation are based on former decisions.

BCET and WCET is strongly related to the technologies mentioned above. The worst case for the execution of a program or a part of it is called "Worst Case Execution Time" (WCET). This factor is important for real-time systems as it determines the capabilities to meet the requirements for accurate timing. A similar factor is the "Best Case Execution Time" (BCET) which is a quantum for the fastest execution time that can be reached with available hardware.

With technologies (caching, pipelining, speculation) as mentioned above it is difficult to determine real-time capabilities because BCET and WCET differ significantly. The more BCET and WCET are similar, the more predictable is the timing behavior of a central processing unit.

More information on this topic can be found in [Jak], [Mül] and [Mue00].

2.8.1.1 Interrupt Handling

Interrupts can be understood as internal or external events to which a computer system has to respond to. Internal events may be a program which executes a division by zero and external events may be interrupts of external peripherals such as A/D-converters. How these events are handled is an important factor in real-time systems because they can happen coincidentally. Further interrupts can be divided into hard- and software (see 2.9.2) interrupts.

[WB05, cmp. p. 137] mentions three typical possibilities for reacting on interrupts:

- current work is aborted because of a critical event to which the processor has to respond to. After suspension an error processing program gets executed
- current work is suspended and another job gets processed. This usually happens if an events with higher priority than the currently processed program occurs. After the new job has been processed the suspended job is put into run state again.
- current work is not affected by the event, it is therefore not suspended. The interrupt which occurred was of less priority than the currently executed program and gets processed afterwards.

[Abb06, cmp. p. 144] explains how external interrupts get processed in hardware on the x86 platform (this is depicted in figure 2.24). The device tagged as "8259" is the

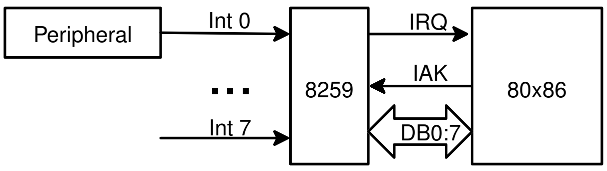


Figure 2.24: Interrupt handling hardware

so called “interrupt controller”. In case an interrupt occurs (on interrupt lines INT0 ... INT7) it is displayed to the processor (80x86) on the IRQ line. The processor then responds to this signal on the IAK line. After that the interrupt controller places the interrupt vector number on the data bus. The interrupt vector number can be understood as an address offset in a table[SGG05, cmp. p. 501]. This offset points to an ISR which in turn can be understood as a small function that is called upon the event of a specific interrupt.

The 8259 can be programmed and therfore it allows to prioritize interrupts. A 8259 can handle up to 8 interrupts and newer PCs have two of them built in. So they are able to handle up to 16 different interrupt requests.

As interrupts are unpredictable events the behavior of how the system responds to them is an important factor influencing the real-time capabilities. On the x86 platform interrupts can be disabled via Clear Interrupts (CLI) and Set Interrupts (STI) instructions. [Abb06, cmp. p. 16] states that using these statements can be as dangerous as using go tos and global variables.

2.8.2 Communication Elements

2.8.2.1 Hubs

As mentioned in 2.5.4.1, hubs or multi-port repeaters cause a half-duplex network. Therefore it is possible for collisions to occur. [LH04, cmp. p. 1] states that there are three approaches for bus-based real-time ethernet:

- token-based medium access (1) and time-slot based (2) protocols are used by cooperating nodes to avoid collisions and to restrain bandwidth limits allocated to the participating nodes in a network.
- (3) statistical approaches which typically only control bandwidth allocation. With these approaches CPU load can be kept lower than with the technology mentioned above. Results using this approach are usually less accurate than deterministic approaches. Thus they can only be used in scenarios where soft real-time behavior is a requirement.

2.8.2.2 Switches and Bridges

Using switches and/or bridges only avoids collisions. A switched network is full-duplex capable. Since all nodes in a network have equal rights when accessing the network, there are still factors which lead to scenarios with unpredictable timing behavior.

[oEI04, cmp. p. 38] describes switch and bridge operation in general. [LH04, cmp. p. 2] mentions the basic elements of a switch. Figure 2.25 shows how they are related to each other. Tyipcally a switch port consists of a transmit and/or a receive queue and two channels - transmit (TX) and receive (RX) - which enable the switch for full-duplex transmission. Logics are implemented in the switch fabric.

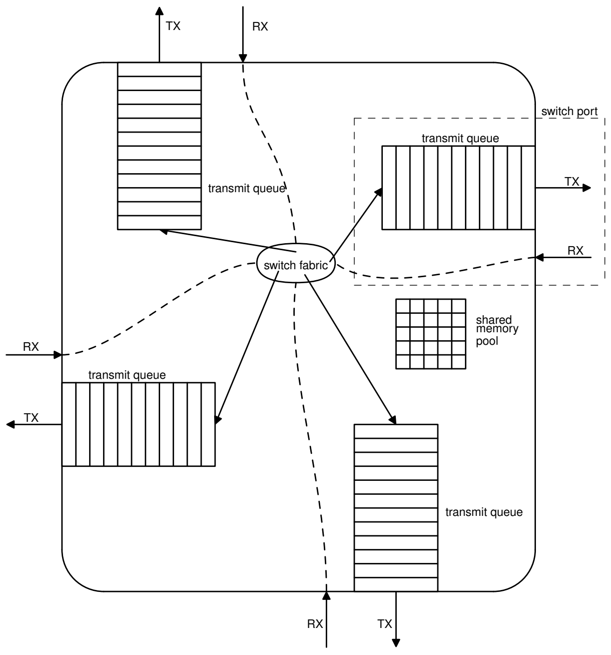


Figure 2.25: Basic elements of an output-queueing switch [LH04, cmp. p. 2]

Switch fabric The logic of the switch which is implemented inside the so called "switch fabric". When receiving a frame the logic determines the transmit port.
[Lim01] defines switching fabric as "the combination of hardware and software that moves data coming in to a network node out by the correct port (door) to the next node in the network. The term suggests that the near synonym, switch, tends to make switching seem like a simple hardware function. Switching fabric includes the switching units (individual boxes) in a node, the integrated circuits that they contain, and the programming that allows switching paths to be controlled. The switching fabric is independent of the bus technology and infrastructure used to move data between nodes and also separate from the router. The term is sometimes used to mean collectively all switching hardware and software in a network."

Before the development of switch fabrics bus structures dominated the market. These bus structures had the limitation that it was only possible to transmit one packet at a time. To overcome this situation, multi-port shared memory systems had been developed in the late 1980s. But this was only partially a solution as shared memory is both difficult and expensive to develop. In the 1990s serial point-to-point backplanes with fast transmission frequency became popular [Sma03, cmp. p. 2].

A switch fabric overcomes the limitations of the bus architecture: operating frequency, signal propagation delay (limits the physical distance a bus can span) and electrical loading (limits the number of devices which can be connected). [Sma03, p. 2.] states that "all switch fabrics are not equal, they generally provide both the power and flexibility needed to build cutting-edge communications equipment and offer a degree of scalability and reliability". Further it states that nodes of a network are connected through a data path which leads through the switch fabric. When implemented as point-to-point links, a single end-point failure does not affect the rest of the system. There are three basic topologies:

- Ring topology: single point of failure topology with the advantage of easy design and lack of data backlogs (bottlenecks)
- Star topology: lower bandwidth than the mesh topology and the possibility of bottlenecks, traffic is easier to control here because it originates from the hub of the star
- Mesh topology: all nodes are connected to each other, so it is equala peer-to-peer system

Queuing engine Switches operate as statistical multiplexers with which it is not possible to predict when packets arrive at the input ports. If more then one packet arrives for an output port, it has to be stored in the switch. Two approaches for packet buffer placement exist: buffering incoming packets and buffering outgoing packets. The former has the disadvantage that packets which could be mediated to a free output port have to wait for packets which are delayed because of a occupied input or output port [Jas02, cmp. p. 69].

[WB05, cmp. p. 262] mentions two different approaches for forwarding frames:

- Cut-Through: a received frame gets analyzed during the reception. This improves latency and response time for real-time applications.
- Store-and-Forward: a frame is stored as whole in the switch's memory before decisions are met. This is generally slower than the cut-through method.

[Jas02, cmp. p 69] mentions that only successfully received packets may be forwarded. Thus the "cut-through" approach does not comply with the standard. Further the publication exemplifies the following queuing algorithms as service disciplines in a networking device:

- First Come First Serve (FCFS): Packets get processed in the order in which they arrive at the receiving port. The maximum amount of time it takes to transmit a packet is the time to transmit the full queue.

- Priority Queuing: Packets which have been assigned higher priority get processed before packets with lower priority. Within a priority class packets get processed in FCFS order.
- Fair Queuing: Requirements for minimal guaranteed bandwidth can be met using this approach. Available bandwidth is assigned to specific traffic classes and queues get processed in cyclic order. A "Credit Counter" is assigned to each queue. This counter gets decremented with each sent packet. If the credit counter reaches a specific value, no further processing of the queue happens in the current cycle.

2.8.2.3 Transceivers

The word transceiver is a combination of the two words "transmitter" and "receiver". In simple words it refers to the bitstream sending entity on network interface cards. On receipt of a packet, an interrupt is activated which in turn causes an Interrupt Service Routine (ISR) to be executed. This ISR fetches the received message out of buffer memory. See section 2.8.1.1 for more information on how hardware-interrupts are handled.

2.9 Software requirements

Real-time operating systems have to meet additional requirements when compared to non-real-time operating systems. These requirements are mentioned in [Sta03, cmp. p. 520]:

- determinism: operations have to be executed at predefined points in time or within specific periods of time. This is especially not the case if several processes share ressources and computing time.
- responsiveness: describes the time span between the detection of an interrupt and the reaction upon this event. This is the sum of the time to detect an interrupt, to start the ISR, to execute the ISR and the behavior of convoluted interrupts (interrupts that occur during interrupt handling).
- user control: the user must be able to assign task priorities with a finer granularity as in common desktop operating systems. Further it is necessary to define if paging in general or suspension of processes is allowed.
- reliability: failures in desktop operating systems are often eliminated through a system reset. This is not possible with real-time operating systems. Further, in case of a defect processor, the system must still meet the requirements for accurate timing. A loss of performance is unportable.
- fail-soft operation: this term describes the behavior of the system in case of a defect. Real-time operating systems try to offer as much functionality as possible. The behavior of a typical (desktop) UNIX system is to write error messages to the console, dump memory to the harddisk and shut down immediately.

Therefore a real-time operating system usually has the following characteristics [SGG05, cmp. p. 700]:

- preemptive, priority-based scheduling: priority is assigned to processes based on their importance. More important tasks are assigned higher priorities than those deemed less important. Preemption means that processes can be interrupted during execution because a more important process has become available.
- preemptive kernel: these kernels[2] allow the preemptive - in other words - the immediate scheduling of a process or task running in kernel mode. Preemptive kernels are mandatory for hard real-time systems because real-time tasks might have to wait an arbitrarily long period of time while another process was active in the kernel.
- minimized latency: interrupts may occur in hardware (see 2.8.1.1) as well as in software (see 2.9.2). The system has to respond to events (interrupts) as fast as possible. Requirements differ in specific applications, so e.g. anti-lock brakes require response times of a few milliseconds and radar controllers for airliners require response times of several seconds.
- networking support: this depends if the system has to be interconnected to other devices. Further there is a distinction if the network to which the system is connected has to be real-time capable. If it has to be real-time capable, the network API is usually implemented on top of the kernel-layer of the operating system.

The remaining part of this chapter describes some of the facts mentioned here in more detail.

2.9.1 The role of the process scheduler

"Many commerical operating systems - as well as Linux - provide soft real-time support." [SGG05, p. 696]. The condition for this statement is that critical real-time tasks have to receive priority over other tasks and have to retain that priority until they complete. This can be explained by examining how the scheduler handles interrupts during the execution of a task.

2.9.1.1 Preemptive scheduling

Preemptive scheduling enables the suspension of lower priority tasks when a task with higher priority has become available. Figure 2.26 shows a task with higher priority waiting (blocking) for an event. This event is signalled by an interrupt which puts this task into the ready state. At the end of the ISR the lower priority task is suspended until the higher priority task is either finished or blocked again.

[2]the central component of most computer systems[Wik07i]

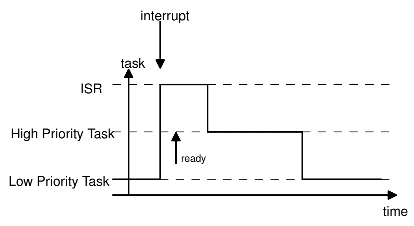


Figure 2.26: Preemptive scheduling [Abb06, p. 150]

Nonpreempetive scheduling requires all tasks being "good citizens" by voluntarily giving up the processor to be sure all tasks get a chance [Abb06, p. 151]. Figure 2.27 shows that the lower priority task still gets executed after the ISR which puts the higher priority task into the ready state. The higher priority task does not get executed until the lower priority task yields the processor or until it is blocked.

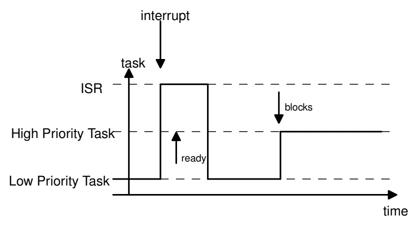


Figure 2.27: Nonpreemptive scheduling [Abb06, p. 151]

[Abb06, cmp. p. 151] further mentions that early versions (the ones that were based on the DOS-kernel) of Windows were nonpreemptive, thus multi-tasking abilities were very limited. Standard Linux is preemptive although it is not considered to be real-time capable due to long periods during which preemption is disabled.

2.9.2 Interrupt handling in real-time operating systems

Figure 2.28 has been taken from [SGG05, p. 702] and describes how interrupt latency is formed. The term interrupt latency can be understood as the period of time from the arrival of an interrupt at the cpu until the start of the execution of the interrupt service routine (ISR). Thus interrupt latency is largely hardware dependend because signals have to get over peripheral busses such as PCI or VME busses.
In the case an interrupt occurs the operating system has to determine the type of interrupt that has been reported. After that it must save the state of the currently executed task. This is also called "context switching".

Interrupts have to be disabled in some cases. For example updating the internal kernel data structures requires interrupts to be disabled. Disabled interrupts contribute to an increasing interrupt latency. Especially in hard real-time systems interrupt latency has not only to be kept small, it also has to be bounded to meet the requirements for deterministic behavior of the whole system.

[Abb06, cmp. p. 170] mentions two technologies which can be used to gain real-time behavior concerning interrupt handling:

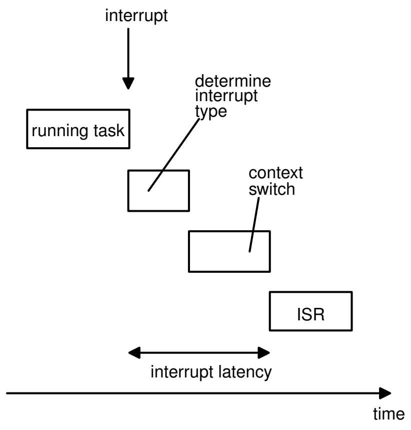


Figure 2.28: Interrupt latency

2.9.2.1 Preemption improvement

Preemption improvement aims to provide a scheduler that is able to guarantee a fixed overhead for a context switch with real-time tasks. This reduces interrupt latency drastically (e.g. from 60 ms down to 1 to 2 ms with a real-time Linux kernel). A further advantage is that real-time tasks can run in user space.

Preemption improvement alone does not make an operating system real-time capable, but it can be seen as a requirement for reducing response times in general.

2.9.2.2 Interrupt abstraction

Interrupt abstraction adds an additional abstraction layer between the hardware and an operating system's kernel. Thus the operating system has no direct control over the activation or deactivation of interrupts. Further the operating system is executed as a process that is under maintenance of the interrupt abstraction layer (which can also be seen as a real-time kernel). This shown in figure 2.29.

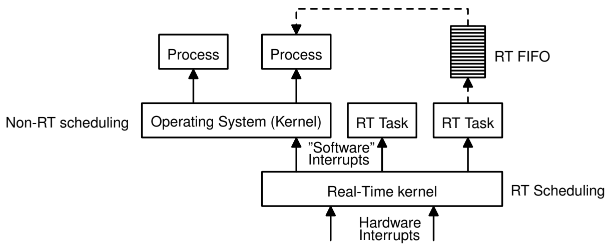


Figure 2.29: Interrupt abstraction [Wik07o, cmp.]

The technology of abstraction interrupts had been patented. This led to the development of a nano kernel architecture in conjunction with the open-source operating system Linux. The project has been called ADEOS and is used to share hardware interrupts between different operating system instances, so called domains. Interrupts are piped through all OS domains. An OS domain can handle an interrupt then or ignore it. The order in the pipeline defines which domain gets occurred interrupts first, thus more important systems should be placed at the beginning of the pipeline. The Xenomai RT Linux extensions are a branch of the RTAI project. The HAL of RTAI is replaced by ADEOS. More information on these projects is given in chapter 3.

2.9.3 Memory Addressing

[SGG05, cmp. p. 699] explains three different approaches to memory management in real-time operating systems:

- real-addressing mode: here physical addresses equal to logical addresses. This implies that a programmer is required to specify where in memory a program has to be loaded. The benefit for this in turn is, that no memory calculation has to take place, thus fast access is guaranteed.
- using a relocation register: the value of the relocation register R is added to the logical address L. The relocation register is set to the memory location where a program is loaded: $P = L + R$
- the real-time system provides full virtual memory functionality: address translation takes place via page tables and Table Lookaside Buffers (TLBs) . This is the most complex and thus slowest approach. One advantage is that it provides memory protection between processes. Examples for this approach is the LynxOS, OnCore Systems and the RTAI Linux extension.

2.9.4 Paging

Paging is strongly related to memory addressing in the fact that the latter enables an application to allocate memory that does not physically exist. In most cases virtual memory is realized as a swap file or a swap partition. As a user space program there is nearly no chance to determine if paging is allowed with the process or not. Thus access to memory (physical or virtual) is provided transparently. Not using virtual memory requires enough memory to be present.

2.10 Approaches for real-time ethernet

[Kos05] mentions three approaches to gain real-time behaviour in ethernet networks. The statistical approach has already be mentioned in 2.8.2.1 in correlation with half-duplex ethernet networks. This section focuses on ethernet that is convenient for hard real-time ethernet.

2.10.1 The QoS approach

This section gives a short introduction to quality of service with focus on use in an ethernet network. To understand the correlations this approach is introduced by identifying the requirements. After that the decision which traffic has to be prioritized is met. This is also called “classification”. Classified traffic has then to be handled accordingly to the requirements which have been defined in the first step.

2.10.1.1 Quality-Parameters for real-time services

[Kos05] mentions that QoS has originally been used in multimedia communications. The transport of audio and video data requires predictable latency timing behavior. The signal quality of transported data depends largely on packet or fragment delays and the achievable throughput.

Further some parameters concerning the quality of service are enumerated:

- Service Availability: the time a service is available. This can be declared as a ratio of time the service is fully functional and observation period.
- Packet Loss Rate: the number of packets or frames discarded during transmission from one node to another
- Delay, latency: the time it takes to propagate a signal through the network. This time is composed of a variable part and a constant part. The signal propagation time and the hardware processing time can be seen as constant parameters. Variable factors concern waiting times mainly in software such as in operating-systems, drivers or firmware of network elements.
- Jitter: the variance of latency. This can be stated as a signed variable that is negative if a frame arrives to early at its destination or positive in case of a late arrival.
- Throughput: the number of bits that can be transported from one node to another through the network

2.10.1.2 Classification of traffic

Traffic classification can be implemented on different layers of the layered network model (see 2.4.1). On the network layer different approaches have been developed [Kos05, cmp. p. 8]. In correlation with the Internet Protocol (IP) two major technologies arose:

Outbound Classification RFC 2205 [IET97, cmp.] and RFC 1633 [IET94, cmp.] together describe the outbound approach with which a protocol is used to reserve resources across the network before a transmission takes place. This requires that each device on the network manages a table which holds parameters of different traffic flows through the device.

Inbound Classification This approach is also called “DiffServ” which expands to “Differential Services”. Each packet is marked accordingly to its priority. A node, the packet passes, may or may not interpret the value of this mark. Inbound classification requires therefore fewer ressources than the outbound approach. This is delineated in RFC 2475 [IET98, cmp.].

On the data link layer especially for ethernet different approaches exist. The IEEE 802.1D and the 802.1Q standards introduced traffic classes. The standard ethernet frame (shown in figure 2.19) does not support these classes. It is then useful when ports of switches are merged to so called “VLANs”. VLANs help to reduce the size of broadcast domains because broadcast frames are only forwarded to nodes in the same segment. Traffic between VLANs has to be routed by a layer 3 device, a router.

To be able to distinguish between the different VLANs an additional header is appended to the ethernet header in case the frame is transported from one switch to another. This VLAN-header also contains information concerning the classification of these frames. Priority is expressed as a value ranging from 0 to 7. Therefore it allocates three bits of the additional VLAN-header which is two bytes long. Inter-switch transmission is also called “trunking” and ethernet frames which have been expanded by the VLAN-header are also called “tagged” frames [oEI06]. A tagged frame is shown in figure 2.30.

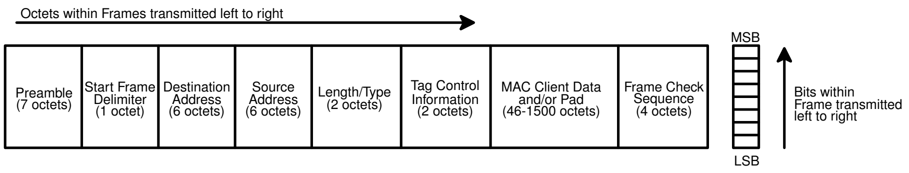

Octets within Frames transmitted left to right
Preamble (7 octets)
Start Frame Delimiter (1 octet)
Destination Address (6 octets)
Source Address (6 octets)
Length/Type (2 octets)
Tag Control Information (2 octets)
MAC Client Data and/or Pad (46-1500 octets)
Frame Check Sequence (4 octets)
MSB
LSB
Bits within Frame transmitted left to right

Figure 2.30: A tagged ethernet frame [oEI06, cmp. p. 239]

Traffic classes from 0 to 7 are identified again in both the IEEE 802.1D and the 802.1Q standard. The higher the value the higher the priority of the frame to be transported. E.g. value 7 represents very important traffic which is used to control the network. Priority 0 is the default value which equals to best effort traffic. Value 1 (background transmission) and 2 (unused) represent lower priorities than the best effort serivce.

Figure 2.31 demonstrates a typical ethernet network with VLANs. VLAN1 consists of the ports 6, 7 and 8 on switch 1 and of ports 1, 2 and 3 on switch 2. Ports 1, 2, 3 and 4 of switch 1 and ports 5 and 6 of switch 2 are assigned to VLAN2. To be able to distinguish between the different VLANs the additional VLAN-header is added to the ethernet frame structure. Thus frames on the trunk-line are tagged and can be assigned a priority as described in the paragraph above. The bandwidth between the VLANs is to be shared on a single trunk-line in this case.

Traffic that flows through a single layer 2 device such as a switch or a bridge can be classified on all fields an ethernet frame contains. In the case of figure 2.31 this could e.g. be traffic flowing from Node1 to Node2 of traffic from Node6 to Node7. But this is not standardized. Therefore this approach is largely vendor specific. In this document two approaches are described: ebtables, which is the layer 2 equivalent for iptables on the Linux operating system, and the capabilities of a Cisco IOS running on a Catalyst Switch.

The ebtables implementation consits of two parts. Filtering capabilities are implemented inside the Linux kernel. This feature has been added to the standard Linux kernel since version 2.6. Patches for Linux 2.4 also exist. Parameters for the categorization are controlled through a user-space program called “ebtables”. Categorized

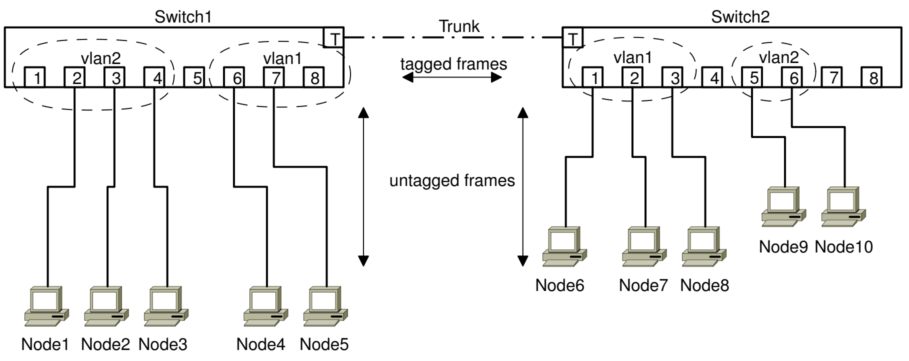


Figure 2.31: An ethernet network with VLANs

frames are distinguished from each other through marks which are interpreted by the Linux kernel. Prioritization is then controlled with another user-space program called “tc” which expands to “traffic control”. Listings 2.1 and 2.2 are based on the example given in [D. 05] and show a reference of how to set up classification of frames based on the MAC-address for prioritization with the “tc” commando.

```
ebtables -A chain -d dst-mac -j mark --set-mark mark-no --mark-target target
```

Listing 2.1: ebtables for ingress traffic

```
ebtables -A chain -s src-mac -j mark --set-mark mark-no --mark-target target
```

Listing 2.2: ebtables for egress traffic

One rule per direction is required to be able to prioritize both ingress and egress traffic. An example of how to use these commands is given in chapter 3.

Capabilities of a Cisco Catalyst Switch Cisco Catalyst switches support the 802.1Q standard [Cis05, cmp. p. 1]. This standard tags frames as described above in the “Inbound Classification” section. [Cis07c, cmp. p. 6] describes how the priority which is to be placed into the VLAN tag of received packets is defined. This value has to be considered during forward decisions. Listing 2.3 gives a short reference on how to define priority on a switchport.

```
switchport priority extend {cos value | trust}
```

Listing 2.3: Set priority of data traffic on a Cisco switch

“cos” is a keyword in this case and value is a number from 0 to 7 conforming the 802.1P priority levels. “trust” is a Cisco specific enhancement which allows the switch to take over the priority of an attached device (e.g. a PC or a VOIP phone).

2.10.1.3 Treatment of classified traffic

[Bei07] states that smart queuing techniques make it possible to increase performance. Less important protocols or applications are handled with lower priority, which guarantees a better service quality for the more important users. The publication mentions three ways and their differences to accomplish this:

- **Queuing** happens only if the interface is busy. In other words - it only happens when it is not possible immediately to transmit a packet over the output interface. The simplest form of queuing is a simple FIFO queue where packets that have been waiting the longest are transmitted first. Packets are then dropped when the queue is full. This case has to be handled by upper layer protocols (e.g. retransmission). Usually more sophisticated queuing mechanisms with queues meeting complex service requirements are implemented. After categorizing a packet it is placed into the appropriate queue. If an interface is ready to transmit the queuing algorithm decides which packet to send first.

- **Traffic shaping** referes to an approach that counts all identified traffic for an interface. If the counted traffic exceeds a user configurable limit (e.g. a bandwidth threshold), additional packets are put into a queue and delayed. Thus bandwidth gets limited to the configured amount.

- **Traffic policing** is similar to traffic shaping except in the point how traffic that exceeds the limits is handled. Usually this traffic gets dropped. Another way could be to modify the priority field (TOS) in the IP header.

[JJ01, cmp. p. 3ff] mentions that the 802.1d standard does not determine how queuing has to be implemented in a switch. The publication describes three different approaches and their effects:

- **First Come First Serve** is an alternative name for the FIFO approach. There is no prioritization and frames get stored in the order they arrive at the switch.

- **Priority Queueing** is similar to an ordinary FIFO in the fact that service within a priority is first come, first served. Higher priority frames move ahead of lower-priority ones. In case of priority queueing with non-preemptive service, higher priority frames do not preempt lower-priority frames that are already in service.

- **Fair Queueing** is a queueing scheme that is mainly used to fulfill different latency and bandwidth requirements for different applications which share the network. It is possible to guarantee a fixed amount of traffic on potential congestion points. Remaining bandwidth is used for other traffic then.

An example for fair queuing is an algorithm called “deficit round robin”. Traffic class queues get processed in a strictly cyclic order. Each queue is assigned a “queue credit”. This parameter determines the number of bits that may be sent in a given service cycle. The “credit counter” is a parameter that determines the number of remaining bits in a service cycle. Fragments are only sent if they are less or equal the size of the credit counter. If the fragment is smaller it gets sent and the credit counter is decreased by the size of that fragment. In the case the fragment is bigger, the queue service terminates for the current cycle and serves the next queue in the cycle. At the begin of a service cycle the credit counter is increased by the amount of the queue credit. Another algorithm, the so called “stochastic fairness queueing” algorithm, is used to gain fairness of many FIFO queues, each handling a connection.[KR05] mentions that a hash-algorithm is used to determine in which a connection has to be placed. This results in random queue assignment for packets which shall be transmitted.

[KR05] mentions two implementations of algorithms for traffic limiting and shaping that have been realized for the popular Linux operating system. Complex QoS scenarios are realized with algorithms that allow organization of traffic in classes. Therefore it is important for the implementation of a real-time network to classify traffic and then to assign these classes criterions that affect the real-time behavior of connections.

Hierarchical Token Bucket (HTB) describes a system for organizing available bandwidth in a hierarchical way. The hierarchical organization is shown in figure 2.32.

A token bucket is an analogy to a bucket with waterdrops (tokens) flowing into it. Tokens are produced permanently. In case the bucket is full, tokens are abolished. Three scenarios can be imagined:

- if fragments arrive as fast as the bucket “produces” tokens, they can be sent immediately
- if fragments arrive at a faster rate they must wait until enough tokens have been produced
- in case enough fragments have been produced, fragments may be sent as fast as possible until all tokens have been used up (burst).

Hierarchical organization of bandwidth allocation aims to gain two targets: bandwidth should be alloted and guaranteed to classes which have been predefined. On the other hand unassigned bandwidth sould be available for all other classes. This is depicted in figure 2.32. Here traffic is shared between two users (e.g. they are identified by their MAC-address). User B is guaranteed a bandwidth of at least 50MBit/s. User A further separates his bandwidth again into two categories. User A can transmit real-time traffic at guaranteed 10MBit/s. If no real-time traffic is to be sent, this bandwidth can be used for data connections. This also applies the other way round. Unused data traffic can be used for real-time traffic.

The hierarchical token bucket algorithm is part of the vanilla Linux kernel since version 2.4.20. After compilation the module is called “sch_htb.o” (or “sch_htb.ko” in Linux 2.6).

Hierarchical Fair Service Curve (HFSC) is an algorithm that has been implemented with focus on the transmission of multimedia data. For voice and video data it is important to transport data in real-time. Bandwidth and latency are managed as separate parameters. So the quality of a service can be imagined as a curve for which parameters have to be defined.

Further the HFSC algorithm differentiates between real-time and a link-sharing criterion. The real-time criterion is responsible to meet absolute guarantees for latency. The link-sharing criterion on the other hand is responsible for fair distribution of bandwidth between different links. In some cases it is important to break the link-sharing criterion in the interest to fulfill the real-time criterion which requires to adhere to the predefined latency. The differences are shown in figure 2.33 and 2.34. The piecewise

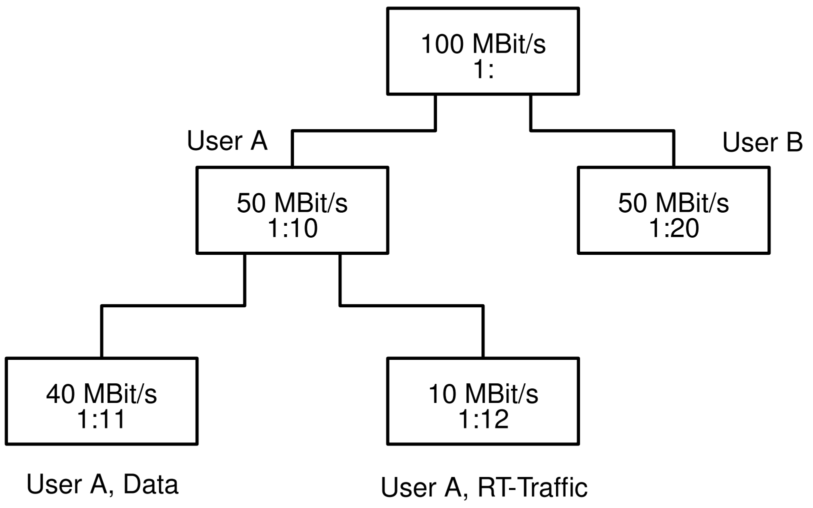


Figure 2.32: Hierarchical Token Bucket Packet Scheduling [KR05, cmp. p. 30]

definition of a service curve requires more parameters to be defined, but as a result, the latency for packet transmission can be minimized. Steeper service curves become more shallow afterwards to meet link-sharing requirements.

As with the HTB algorithm, this algorithm is part of the vanilla Linux kernel since version 2.4.20. The module is called "sch_hfsc.o" or "sch_hfsc.ko" respectively in Linux version 2.6.

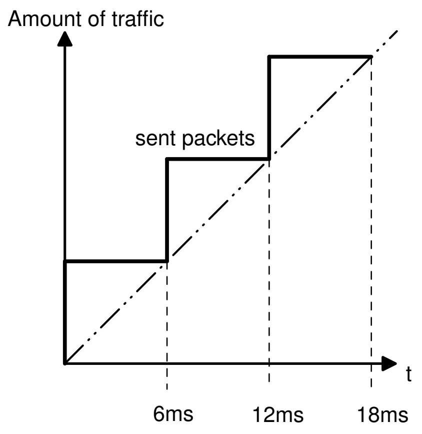


Figure 2.33: Linear service curve [KR05, cmp. p. 34]

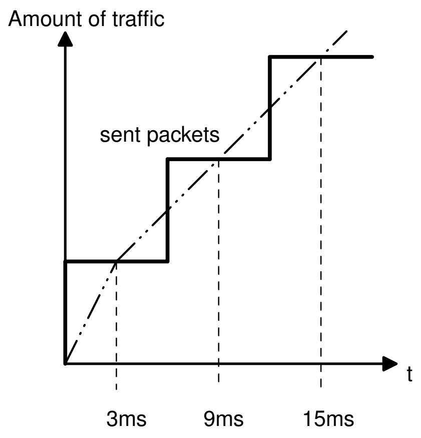


Figure 2.34: Piecewise service curve [KR05, cmp. p. 35]

2.10.1.4 Conclusion

The Quality of Service approach allows unmodified standard protocols to be used. This is a big advantage over the TDMA approach which cannot be implemented existing protocols that are standardized. Further [Kos05] mentions that in [Jas02, cmp. p.] is explained how to use ethernet with Class of Service (CoS) for real-time applications.

Hard real-time can be reached in dependence on the used topology and requirements to the technical process. It is possible to gain transaction times in the range of a few milliseconds if the right topology has been chosen and the load has been configured accordingly.

2.10.2 The TDMA approach

Time Division Multiple Access has already been explained briefly in 2.4.3.8. This chapter shows that this concept is hard real-time capable and how it has been implemented recently in the *RTnet* project. This project has been chosen as reference project because it offers a mostly POSIX socket compatible interface, is under active development, available as source code and well documented.

With TDMA in ethernet networks it is guaranteed that only one node at a time accesses the whole network. Therefore no collisions can occur. Congestion of network traffic is also avoided [J. 04, cmp. p. 2].

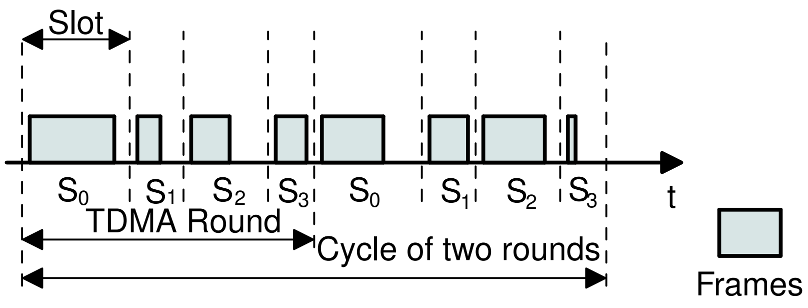


Figure 2.35: TDMA cycles

Figure 2.35 is based on figure 3 of [J. 05b, cmp. p. 4] and shows a TDMA setup with static slot allocation. Thus the period of each slot is constant over all TDMA cycles. How much of the available time in a slot is used by a station depends on the amount of traffic it has to convey over the network.

2.10.2.1 Synchronization in TDMA networks

Usually the TDMA approach also requires the use of a time synchronization scheme. This can happen via self made protocols as the “rtnet” project does, but there also exist standardized solutions. In this regard the Precission Time Protocol (PTP) should be mentioned. It is standardized in IEEE 1588. [RDH07, cmp. p. 2] mentions that synchronization in the range of sub-microseconds is possible. This is far more precise than the wide spread Network Time Protocol (NTP) which is used to synchronize clocks of computer systems. With NTP-systems the peak error may be up to 100ms whereas with PTP systems it is $100\mu s$. Since the PTP is used as a dedicated protocol for synchronization only it can be seen as an outbound signalling protocol.

As mentioned above, some projects as the “rtnet” implementation of a TDMA network follow a different way. Clock synchronization for the participating nodes in the same segment is realized in a pluggable MAC layer which is called “RTmac”. In this case timing information is exchanged during a TDMA cycle which can be compared to an inbound signalling scheme. A client starts to synchronize its clock with a calibration request to the master. The master replies with a message that contains the request arrival and reply departure times, both as precise as the system allows [J. 05b, cmp. p. 4]. The slave is then able to calculate the round-trip delay. An estimation for the medium travel time is carried out.

$$\begin{aligned} t_{travel\ [s]} &= \frac{1}{2} * \frac{1}{n_{\ [1]}} \sum_{i=1}^{n\ [1]} (transmission\ time\ slave \Rightarrow master \\ &\qquad + transmission\ time\ master \Rightarrow slave) = \\ &= \frac{1}{2} * \frac{1}{n_{\ [1]}} \sum_{i=1}^{n\ [1]} (T_{slave\ msg\ received\ [s]} - T_{master\ msg\ sent\ [s]} \\ &\qquad + T_{master\ msg\ received\ [s]} - T_{master\ msg\ sent\ [s]}) \end{aligned} \tag{2.17}$$

With a lowercase t referring to a time period and uppercase T identifying a specific point in time, formula 2.17 averages round trip times of n tries. To get the time for transfering a message from this slave to the master the round trip time is divided by 2.

To reduce the potential scheduling jitters an offset period t_{offset} is calculated (formula 2.18). With T_{sched} being the time when a data transmission cycle is intended to begin, the point in time when a specific slot starts can be calculated: T_{slot}. The offset period T_{offset} in turn allows to improve precision of slot starting times (see T_{slot} in formula 2.19):

$$t_{offset\ [s]} = T_{master\ msg\ sent\ [s]} + t_{travel\ [s]} - T_{master\ msg\ received\ [s]} \tag{2.18}$$

$$T_{slot\ [s]} = T_{sched\ [s]} + t_{slot\ [s]} - t_{offset\ [s]} \tag{2.19}$$

Multiple packets can be queued on a slot. Therefore it was necessary to define priorities with which traffic gets transported. The granularity in this case is per packet. If two packets have been scheduled on the same time slot the packet with highest priority (priority 0) gets sent first. *RTnet* currently offers 31 real-time levels and one level for non real-time traffic.

With the Time Triggered Protocol (TTP) a similar approach has been implemented. The difference to *rtnet* is, that synchronization between the nodes is completely implemented in hardware [Pop00, cmp. p. 16]. The network is abstracted in form of a TTP-controller which mainly consists of a message base interface (MBI) which is often implemented as a dual ported RAM. Thus this memory area can be accessed through the RT-Kernel, running on the node, and the TTP-controller. Messages are "produced" in course of the running process and the TTP-controller knows then when a message can be released onto the bus.

2.10.2.2 Varities of TDMA

Centralized slot allocation works with a master/slave relation ship between the nodes. The master is often referred to as "Managing Node". It determines the time when each node is allowed to access the medium [J. 05b, cmp. p. 4]. Existing implementations can be distinguished based on which point in time a node is allowed by the master to access the network:

- Polling through the managing node: [WB05, cmp. p. 304] explains that Ethernet Powerlink depends on a single master querying all nodes in a network for data transmissions. Thus no collisions can occur and the underlying CSMA/CD media access needs not to be disabled. See 4.2 for further explanations.
- Assignments of time slots through the managing node: for this case the *RTnet* project uses a mechanism called "RTcfg" which is implemented in an own network protocol that integrates into the TDMA communication cycle. It is used to configure both generic and service specific (concerning the RTmac media access) parameters, as well as monitoring services. The protocol does not depend on specific transmission protocol such as IPv4. It only requires support for broadcast transmissions. [J. 05b, cmp. p. 5] mentions four tasks of RTcfg to handle:
  - Handling new nodes which join the real-time network at any time. Discipline configuration data has to be distributed in this case (see below).
  - Monitoring of active nodes (e.g. exchange of physical and logical addresses, comparable to what the ARP does)
  - The start-up procedure has to be synchronized. The way this happens is defined by *RTcfg*
  - Arbitraty configuration data is exchanged without protocols such as TFTP/FTP and their underlying TCP/IP protocols.

  The parameter sets for all clients in the managed network are stored in a central configuration server (the master). The registering sequence of a client happens in three steps:

  - First, the client receives a single packet that contains initial parameters and data to set up a RTmac discipline. The client recognizes that the message is directed to its physical or logical address.
  - The second step is to announce the presence of the client to any other network node. In this step address information is exchanged. This can be compared to the ARP process in standard ethernet networks. Configuration data can be fragmented into several frames or packets. Stage 2 is finished after the server (master) has received an acknowledge message from the last missing client node.
  - The last step is optional and can be seen as a rendezvous point for both server and clients to wait for all participants to complete processing the configuration data.

  During operation, clients are monitored with a heart-beat signal. In case of a time-out a client can be declared dead, which leads to a broadcast message, informing all other node about the absence of this node. Nodes which receive this message remove any address of the broken client in their local tables.

Distributed allocation is being introduced in [Pop00, cmp. p. 15] on the base of the TTP. The document describes that each node in such a network is assigned a fixed slot in a communication cycle. [H. 98, cmp. p. 5] explains how slots are allocated with this approach. Each nodes contains a message descriptor list (MEDL) in its TTP-controller. This list in turn contains the information which node is allowed to send a message of a specific format at a particular point in time.

In a TTP network information is exchanged through two different frame types: normal (N) frames and initialization (I) frames. Frames types are distinguished by their first bit of their header. A TTP based system is initialized and failed nodes are reintegrated with I-frames. The data field of an I-frame contains the state of the TTP-controller. N-frames are used to convey data over the real-time network.

[Her95, cmp. p. 2] mentions TTP in the MARS project as a field of application. The architecture of the MARS project is explained in [Joh93, cmp. p. 2]. It has been developed at the Technical University of Vienna aiming at management of highly critical distributed control applications. Processing nodes must show a fail-silent behaviour and messages are exchanged through a network with a time triggered structure.

2.10.2.3 Conclusion

One of the downsides of TDMA is, that no standard protocols can be used. It requires the networking stack of used operating systems to be modified by all means. The point in time, when a packet or a frame enters the network is essential because both collisions (in half-duplex networks) and traffic congestion (in both half- and full-duplex networks) has to be avoided. Further it may happen that bandwidth, which could have been used for best-effort traffic is wasted due to a bad scheduling scheme [Kos05, cmp. p. 15]. This can partially be avoided in case dynamic slot allocation is used.

The advantage of TDMA media access is that determinism can be gained easily. Compared to the QoS approach there are fewer factors that influence the real-time behavior of the whole system.

2.10.3 The Token-Passing approach

[Kos05, cmp. p. 15] describes that Token-Passing is related to the TDMA approach in the point, that a transmission happens in time slots. Synchronization of nodes in such a network is not necessary as media-access is regulated through a so called “Token”. This token is a data packet that is handed from node to node in a cyclic fashion. The owner of the token is allowed to place data onto the bus.

[Tzi99] mentions an academic approach to real-time ethernet which is based on token-passing media-access technology. As with *RTnet* it is an academical project which aims at using standard ethernet hardware for implementing a real-time network. In other words it is a software-only solution which can be understood as an additional layer above the CSMA/CD media-access scheme. It also offers a industry-standard conforming socket interface. Concrete implementations exist for FreeBSD, Linux and DOS operating systems.

Real-Time Ethernet (RETHER) connections are unidirectional and have to be established before data can be transmitted. After the establishment standard **send()** and **recv()** system calls can be used. Originally the project has been designed to work in single-segment networks. Extensions lead to the support for switched ethernet networks. Thus it is possible to operate a multi-segment network. Nodes between two or more segments act as gateways. These gateways act as sender on one segment and as a receiver on the other because of the unidirectional nature of connections.

2.10.3.1 Conclusion

The Token-Passing approach has not been chosen by any of the wide spread industrial ethernet solutions. The RETHER project has not been modified since many years. The latest implementation has been developed for FreeBSD 2.1. Therefore it is only mentioned here as a supplement to the QoS and the TDMA approach.

Chapter 3

Validation of real-time timing-behavior

3.1 Introduction

This chapter describes the concrete implementation of a real-time ethernet network. The experiment was used to compare two different approaches to real-time behavior:

- Ethernet-Hardware with tdma media-access (figure 3.1)
- Ethernet and QoS (figure 3.2)

The latter has been the prefered way to implement real-time networks [Kut02, cmp. p. 74].
The main purpose of the experiment was to show how to apply the time/utility functions which have been stated in 2.3.1.1.

3.1.1 The arrangement

Three computers have been connected through an ethernet switch with ethernet cables (see figure 3.3; only the master and one slave node is shown for simplicity - the other slave node is connected in the same way as the shown one). One of these computers acts both as communication master and sink. The other computers act as source and send messages towards the sink which in turn changes the state of the parallel data-port pins based on the messages it received from these client nodes. Required parallport pins are connected to the persistence oscilloscope through a two-wire cable. Thus port-changes can be recorded. Gathered data is then transmitted to the management node through a RS232 cable for further investigation and statistical analysis.

3.2 Evaluation of the real-time communication system

In this document the evaluation of real-time communication systems is considered to be accomplished in three major steps:

- traffic generation

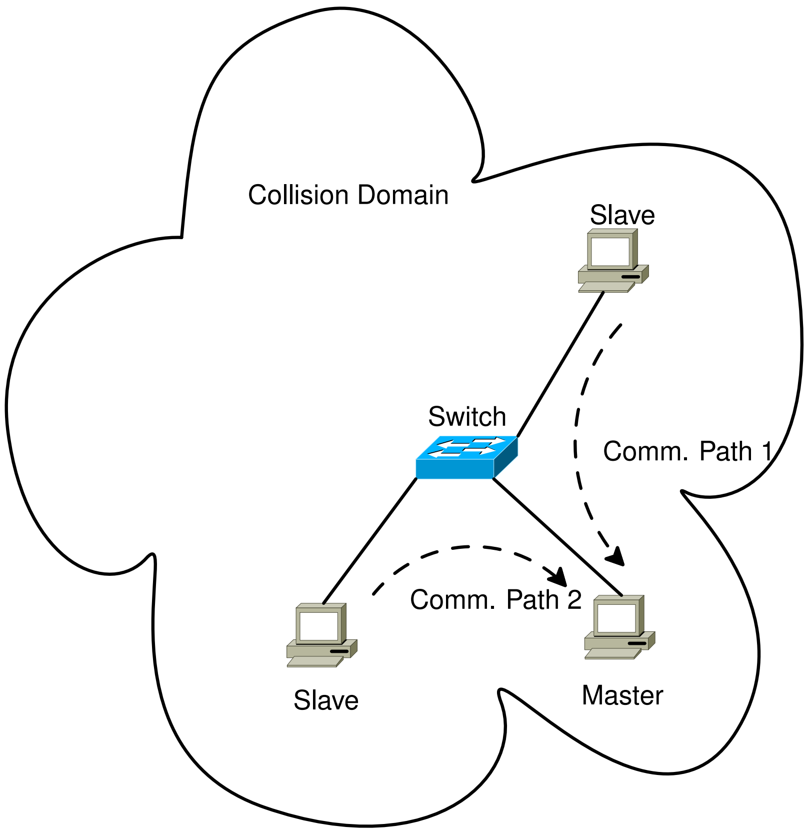


Figure 3.1: Set-up of the real-time TDMA experiment

- traffic accounting
- analysis of gathered data

3.2.1 Traffic generation

For the evaluation of real-time data transmission, traffic should be generated. The following approaches have been considered:

- generating packets from user-space in a non real-time operating system
- generating packets from a non real-time, single tasking operating system with busy waiting
- generating packets form either user- or kernel-space with a real-time operating system

Off-the-shelf operating systems have limited support for generating traffic at a constant rate in a user-space process. This can be reasoned in schedulers which interrupt long lasting processes. In preemptive operating systems processes get interrupted due to preserve processing time for other processes and threads [Jos05, cmp. p. 33]. Thus the packet generation by user-space processes would result in frequently packet bursts

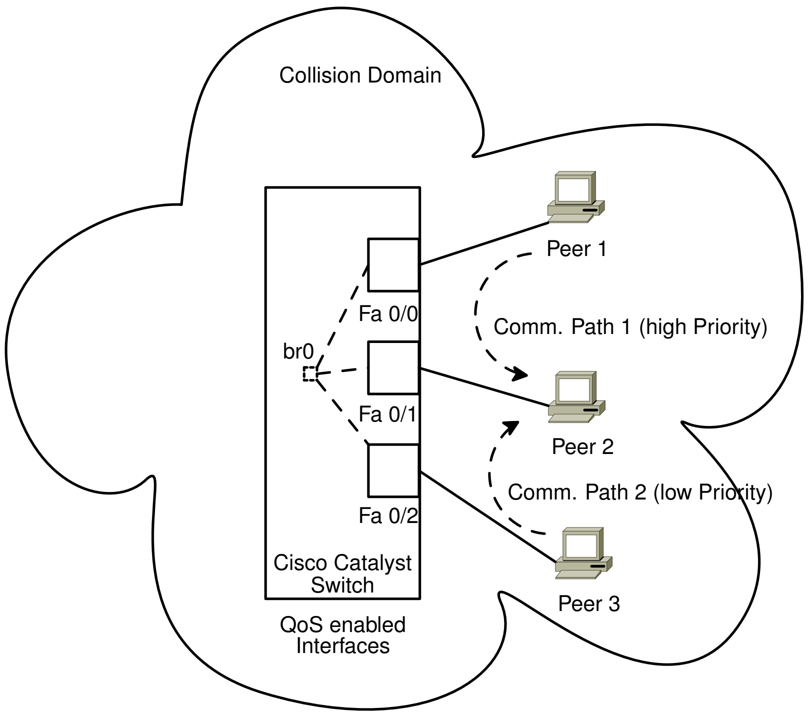


Figure 3.2: Set-up of the QoS experiment

which are interrupted and triggered by the process scheduler. In some cases this is desireable as it behaves like a real coincidentally occurring data transmission. This document focuses on packet generation at a constant rate for all real-time approaches in user-space because of better comparability.

The correlation between packets sent on the source and received on the sink would require an additional mechanism which assures that packets are accounted correctly. This issue can be traced back to the unreliable nature of the UDP protocol [Hei02, cmp. p. 332] with which higher layer protocols are required to initiate a retransmission of lost packets.

The parallel port can be used for measurements of timing-behavior because of its low latency [J. 03, cmp. p. 1]. This requires direct access to the hardware which is usually achieved with low level port input and output calls [cap06, cmp.]. In the Linux operating system this is intended for kernel internal use. Additionally it is only available to specific hardware such as the wide-spread PC x86 architecture [Fre95, outb(2)].

If using these system calls from user-space is undesirable, a user-space parallel port device driver has been implemented. It allows modification of the port pins as unprivileged user via `ioctl()` (see [Fre95, ioctl(2)]) calls which in turn do the low level port access [Tim00]. To investigate the timing-behaviour a device with a common time-base for its inputs is desireable. Thus a persistence oscilloscope has been used. This also allows later examination of gathered data.

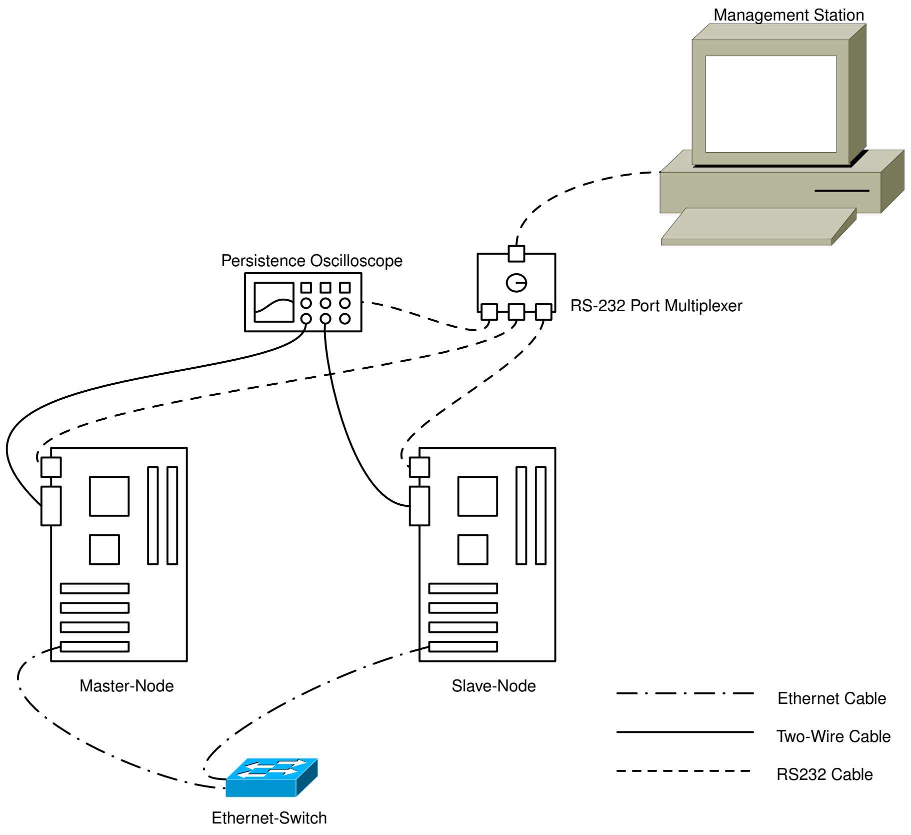


Figure 3.3: The arrangement for the experiment

Another approach would be to measure latency based on time stamps which are placed into sent data. Measuring latency on two devices would require to synchronize them because latency is calculated as a difference and points in time are absolute time stamps. Synchronization can be disregarded when measuring the round trip time. This approach has been chosen in [Sch06, cmp.]. In this case measurements regarding the transmission and reception of data are conducted on the same device. Timepoints are then based on the same timebase, and as such it is shook out. This correlation is shown in formula 3.1.

$$\begin{aligned} t_{RTT\ [s]} &= T_{back\ [s]} - T_{base\ [s]} = (T_{base\ [s]} + t_{forth\ [s]} + t_{back\ [s]}) - T_{base\ [s]} \\ &= t_{forth\ [s]} + t_{back\ [s]} \end{aligned} \tag{3.1}$$

Using the parallel port as a display for events has its limitations. It is difficult to narrow down the results of measured data. If, for example, the parallel port state has simply been bound to interrupts of a network interface card, with a change of the signal gage it is not possible based on the changing signal level to determine what caused this event. The interrupt could eventually be triggered by either a received packet or an application which initiated a data transmission. This circumstance has then to be investigated by the interrupt service routine [J. 05a, cmp. p. 269]. A filter which changes the ports state only on a specific event could be implemented in software. This adds latency

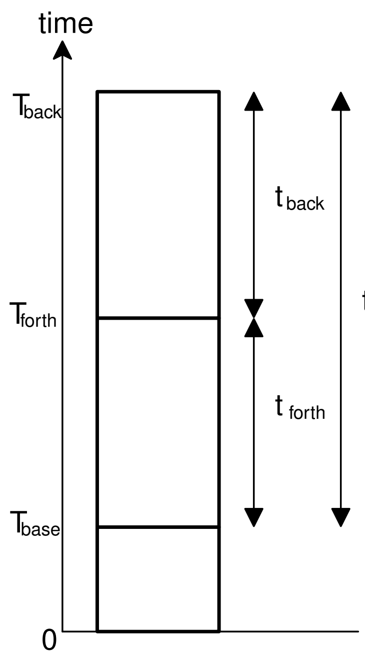


Figure 3.4: Calculating the round-trip time

(e.g. from the operating systems scheduler if it has been implemented in user-space) to the results. These parameters have been explained in 2.9.1.

3.2.2 Traffic accounting

In this document traffic should be measured using a persistence oscilloscope. Thus it is necessary to provide an interface between the nodes which participate in communication and the oscilloscope. As described in 3.2.1 the parallel port can be used to signal certain events in the communication structure.

Two approaches for determining the points in time of events have been considered:

3.2.2.1 Implementation of an interrupt service routine

Interrupt service routines[1] are called upon the event of a hardware interrupt [J. 05a, cmp. p. 258]. Interrupt handlers run at interrupt time. In e.g. Linux this results in further restrictions which forbid anything that would sleep, lock a semaphore or allocate memory except in one specific way (using the `GFP_ATOMIC` macro).

The interrupt service routine does not handle the interrupt itself but changes the state of the parallel port which can then be measured using the oscilloscope.

[1]ISR, sometimes also referred to as “interrupt handler”

3.2.2.2 Implementation of a client-server architecture

The second attempt comprises all parts of the real-time network as described in section 2.7 and consists of two programs: a server and a client application. The client application sends UDP packets towards the server. With each sent UDP packet the state of a parallel port pin of the client computer changes. On the server side each received packet containing predefined data triggers the state of the parallel port at the server computer to change its state. Thus an oscilloscope which has a common time-base for at least two of its inputs can be used to determine the time period (= latency) between these two events.

In other words this approach enables to measure the latency between the notification and the processing of a specific event in an automation network.

3.2.3 Analysis of gathered data

After gathering the data it has to be conveyed from the measurement device to the management station. Modern persistence oscilloscopes often offer serial (RS232, USB) or parallel (IEEE 1284) interfaces for this purpose. On the computer side a management application takes over the receiving task.

Gathered data has to be prepared into a format which statistical applications can import. Imported data is then fitted to a special distribution model. This allowes to make statistical estimations [Lot01, cmp. pp. 313] with a predictable probability of correctness [Lot01, cmp. pp. 598]. The intention to make these statistical estimations is to predict behavior of the communication system with changed parameters.

3.3 Preliminaries

3.3.1 Used Hardware

3.3.1.1 Computer hardware

Developed software has been tested mainly on off the shelf hardware. Only x86 based processors have been used. This decision has been met to prove that it is possible to gain real-time capabilities even on very cheap and wide spread hardware. The following systems have been set up:

- System 1 (Master): AMD K6-2 with 300 MHz and 128 MBs of RAM
- System 2 (Slave): Intel Celeron with 433 MHz and 64 MBs of RAM
- System 3 (Slave): AMD Geode with 133 MHz and 32 MBs of RAM

3.3.1.2 Networking hardware

- Network interfaces cards (NICs): Realtek 8139C and Realtek 8139D based interface cards of different vendors for the PCI bus

- Switch: Surecom EP-808X-R: 8-Port 100/10M N-Way Mini Switch (RTnet experiment)
- Switch: Cisco 2950 Series “IOS (tm) C2950 Software (C2950-I6Q4L2-M), Version 12.1(22)EA5, RELEASE SOFTWARE (fc1)” (QoS experiment)

3.3.1.3 Supplementary tools

- Oscilloscope: Tektronix TDS 220 persistence oscilloscope equipped with an extension module with serial (RS232) and parallel (IEEE1284) interfaces for data exchange with the management station

3.3.2 Software used in the experiment

The focus has been held on open-source software since it is possible to see how a particular technology has been implemented. Further it is possible to make modifications such as the insertion of trace points (e.g. simple printf() statements).

3.3.2.1 Operating System

Mainly the Debian GNU/Linux Etch (4.0, testing branch) distribution has been used as base as well for the real-time operating system, as for the development platform, as for the deployment platform. The difference lies in the modifications which have been undertaken:

- Development platform: no major modifications, only the (Linux) kernel has been compiled for personal needs
- Testing platform: same software constellation as with the target (the system where the real-time environment is to be used). A real-time kernel based on the RTAI HAL patches (see 3.3.2.2) has been compiled and installed. Testing has been performed in a virtual machine (see 3.3.2.3) as far as possible.
- Deployment platform: same as the development platform

3.3.2.2 Real-Time extensions for Linux

During implementation of the real-time distribution the choice between the RTAI and Xenomai Linux extension had to be met.

The Xenomai Real-Time framework for Linux offers support for real-time user-space processes [S. 07, cmp. p. 2]. It is supported as backend by the RTnet project. As of the writing of this document the project is under active development. The main reason why Xenomai has not been chosen for this thesis was the lack of documentation.

RTAI on the other hand is the older project. Therefore more documentation is available in general. Some versions of RTAI and RTnet do not correlate well together. E.g. the combination of RTAI 3.4 and RTnet 0.9.8 required some header files to be modified. This experiment is based on RTnet 0.9.8 and RTAI 3.5 which correlated together without any modifications. According the information given in the `Changelog`[2] file RTAI 3.5 mainly differes from RTAI 3.4 in its implementation of the scheduler and interprocess communication mechanisms. Real-time processes can be invoked from user-space using the “lxrt” kernel module [P. 04, cmp.].

3.3.2.3 Virtualization software

For virtualization the processor emulator QEMU which is also open-source software has been used. The purpose for virtualization in this correlation was to test if developed software works together with both the real-time kernel and the real-time libraries which offer an interface to real-time kernel functions. With the emulator being a user-space process its execution has the restriction to be suspended by the host’s process scheduler. As such examining the timing-behavior was not possible in the virtual environment.

The QEMU Accelerator is an extension to QEMU which is basically a user-space process. The accelerator is mainly a kernel driver (e.g. a kernel module with Linux) which allowes the emulator to cooperate very fast with the operating-system kernel (memory access, CPU access, etc.) [Fab07, cmp.].

3.3.3 Correlation of development and deployment

Software which was intended to run on the real-time platform was developed on the host system. All the tools which are needed for development require unnecessary ressources on the target platform. Once software has been compiled for the target platform it was tested on a virtual machine. After passing the tests it was deployed to the real-time platform.

3.3.4 Steps toward a real-time network

A linux system is prepared to be able to be deployed as a real-time platform. The distribution to be deployed is based on Debian 4.0. The deployment platform is also based on Debian GNU Linux 4.0. All binaries which are contained in the Debian packages are compiled for Intel i386 processors. Self-compiled binaries (which include the real-time kernels) are compiled for Pentium MMX processors, since this is the greatest common divisor in the used systems.

3.3.5 Compilation of a patched kernel

First of all we have to obtain the unmodified (vanilla) linux kernel-source. In this case an Austrian mirror has been chosen (Listing 3.1).

```
phylos:/usr/src% wget ``http://www.kernel-inode-at.lkams.kernel.org/pub/linux/kernel/v2.6/linux
    -2.6.17.tar.bz2''
--09:32:20--  http://www.kernel-inode-at.lkams.kernel.org/pub/linux/kernel/v2.6/linux-2.6.17.tar.
    bz2
           => `linux-2.6.17.tar.bz2'
Resolving www.kernel-inode-at.lkams.kernel.org... 81.223.20.167
Connecting to www.kernel-inode-at.lkams.kernel.org|81.223.20.167|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42,731,688 (41M) [application/octet-stream]
```

[2]http://cvs.gna.org/cvsweb/vulcano/ChangeLog?rev=1.173;content-type=text%2Fplain;cvsroot=rtai

```

...
```

Listing 3.1: Obtaining the vanilla kernel-source

After that the decompression of the source-code some symbolic links have to be set up (Listing 3.2)

```
phylos:/usr/src% tar -xvjf linux-2.6.17.tar.bz2
linux-2.6.17/
linux-2.6.17/.gitignore
linux-2.6.17/COPYING
linux-2.6.17/CREDITS
linux-2.6.17/Documentation/
linux-2.6.17/Documentation/00-INDEX
...

phylos:/usr/src% ln -s linux-2.6.17{,-rtai-pentium-mmx}
```

Listing 3.2: Obtaining the vanilla kernel-source

Now the source-code of RTAI can be obtained and installed (Listing 3.3).

```
phylos:/usr/local/src% tar -xvjf rtai-3.4-cv.tar.bz2
rtai-3.4-cv/
rtai-3.4-cv/addons/
rtai-3.4-cv/addons/comedi/
rtai-3.4-cv/addons/comedi/README
rtai-3.4-cv/addons/comedi/rtai_comedi.h
rtai-3.4-cv/addons/comedi/GNUmakefile.am
...
```

Listing 3.3: Installing the source-code of rtai

After that the kernel can be patched for the use with RTAI (Listing 3.4)

```
phylos:/usr/src/linux% cat /usr/local/src/rtai-3.4-cv/base/arch/i386/patches/hal-linux-2.6.17-i386
        -1.3-08.patch | patch -p1
patching file drivers/pci/msi.c
patching file include/linux/hardirq.h
patching file include/linux/ipipe.h
patching file include/linux/linkage.h
patching file include/linux/preempt.h
...
```

Listing 3.4: Patching the linux-kernel

Now the kernel can be built. In this case some debian specific tools [Ron07, cmp.] are used. In the end there will be a debian package that can easily be installed and removed on the target platform (Listing 3.5)

```
phylos:/usr/src/linux% fakeroot make-kpkg --append-to-version -rp-rtai-pentium-mmx --revision 2
        kernel_image modules_image
```

Listing 3.5: Compiling the kernel with a debian tool "make-kpkg"

This results in a debian package called "linux-image-2.6.17-rp-rtai-pentium-mmx_2_i386.deb". It contains the linux kernel and the loadable modules. The distributed "Makefile" has been altered to compile binaries for the i586 (Pentium) platform:

```
--- makefile    2007-03-21 08:26:01.000000000 +0100
+++ makefile.rp 2007-03-21 08:26:20.000000000 +0100
@@ -49,6 +49,7 @@

 reconfig:: config-script

+config.status: host_alias:=i586-pc-linux-gnu
 config.status: .rtai_config
                @test -r config.status && recf=yes || recf=no ; \
                eval `grep ^CONFIG_RTAI_INSTALLDIR $<`; \
```

Listing 3.6: Installing the source-code of rtai

The build directory for RTAI is then created (Listing 3.7).

```
phylos:/usr/local/src% mkdir rtai-3.4-cv-build
```

Listing 3.7: Installing the source-code of rtai

In the build directory we can start the configuration process (Listing 3.8)

```
phylos:/usr/local/src/rtai-3.4-cv-build% make -f ../rtai-3.4-cv/makefile.rp menuconfig
```

Listing 3.8: Configuring RTAI

Standard settings have been used. The configuration has only been modified, for the binaries to contain debugging symbols (Listing 3.9).

```
CONFIG_RTAI_KMOD_DEBUG=y
CONFIG_RTAI_USER_DEBUG=y
```

Listing 3.9: Installing the source-code of rtai

After that the make command can be invoked:

```
phylos:/usr/local/src/rtai-3.4-cv-build% make -f ../rtai-3.4-cv/makefile
make all-recursive
make[1]: Entering directory `/usr/local/src/rtai-3.4-cv-build'
Making all in base
make[2]: Entering directory `/usr/local/src/rtai-3.4-cv-build/base'
Making all in include
...
phylos:/usr/local/src/rtai-3.4-cv-build% make install
```

Listing 3.10: Compiling RTAI

The invokation of "make install" has been aborted at the creation of device entries in the /dev directory. This would require root rights and is not necessary here since it is the development platform.

3.3.6 Compilaton of RTnet

RTnet has been compiled to work together with RTAI. After downloading the package it has been compiled (listing 3.11).

```
phylos:/usr/local/src% wget ``http://www.rts.uni-hannover.de/rtnet/download/rtnet-0.9.8.tar.bz2''
--23:51:03--␣␣http://www.rts.uni-hannover.de/rtnet/download/rtnet-0.9.8.tar.bz2
␣␣␣␣␣␣␣␣␣␣␣=>␣`rtnet-0.9.8.tar.bz2'
Resolving␣www.rts.uni-hannover.de...␣130.75.137.14
Connecting␣to␣www.rts.uni-hannover.de|130.75.137.14|:80...␣connected.
HTTP␣request␣sent,␣awaiting␣response...␣200␣OK
Length:␣897,736␣(877K)␣[application/x-bzip]

100%[==================================================================================>]␣897,736␣
     ␣␣␣␣␣121.68K/s␣␣␣␣ETA␣00:00

23:51:11␣(124.57␣KB/s)␣-␣`rtnet-0.9.8.tar.bz2'␣saved␣[897736/897736]

phylos:/usr/local/src%␣tar␣-tjf␣rtnet-0.9.8.tar.bz2
rtnet-0.9.8/
rtnet-0.9.8/NEWS
rtnet-0.9.8/TODO
rtnet-0.9.8/aclocal.m4
rtnet-0.9.8/stack/
rtnet-0.9.8/stack/ipv4/
...
phylos:/usr/local/src%␣tar␣-xjf␣rtnet-0.9.8.tar.bz2
phylos:/usr/local/src%␣ln␣-sf␣rtnet-0.9.8␣rtnet
phylos:/usr/local/src%␣cd␣rtnet
```

Listing 3.11: Compilation of RTnet

The makefile had to be adapted again to meet the host-system requirements (Pentium processor; i586 architecture) (figure 3.12).

```
phylos:/usr/local/src/rtnet% diff -Nau makefile{,.rp}
--- makefile      2007-03-21 09:16:48.000000000 +0100
+++ makefile.rp 2007-03-21 09:16:59.000000000 +0100
@@ -48,6 +48,7 @@

 reconfig:: config-script

+config.status: host_alias=i586-pc-linux-gnu
 config.status: .rtnet_config
        @test -r config.status && recf=yes || recf=no ; \
        eval `grep ^CONFIG_RTNET_INSTALLDIR $<`; \
```

Listing 3.12: Adapting the makefile

After that the installation could be finished (listing 3.13).

```
phylos:/usr/local/src/rtnet% make -f makefile.rp menuconfig
make[1]: Entering directory `/usr/local/src/rtnet-0.9.8/scripts/kconfig'
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer  -DLOCALE  -DCURSES_LOC="<ncurses.h>" -I/
      usr/local/src/rtnet/scripts/kconfig -c /usr/local/src/rtnet/scripts/kconfig/lxdialog/
      checklist.c -o lxdialog/checklist.o
...
phylos:/usr/local/src/rtnet% make
phylos:/usr/local/src/rtnet% make install
```

Listing 3.13: Finishing the installation of RTnet

3.3.7 Deploying the libraries

```
rtk62:/usr/local/rtai/lib# echo "include /etc/ld.so.conf.d/*.conf >> /etc/ld.so.conf
rtk62:/usr/local/rtai/lib# pwd >> /etc/ld.so.conf.d/rtai.conf
rtk62:/usr/local/rtai/lib# ll /etc/ld.so.cache             [19:42:51 @ 07-03-15]
-rw-r--r-- 1 root root 9373 2007-03-15 19:42 /etc/ld.so.cache
rtk62:/usr/local/rtai/lib# ldconfig                        [19:42:56 @ 07-03-15]
rtk62:/usr/local/rtai/lib# ll /etc/ld.so.cache             [19:42:58 @ 07-03-15]
-rw-r--r-- 1 root root 9693 2007-03-15 19:42 /etc/ld.so.cache
rtk62:/usr/local/rtai/lib#                                 [19:42:59 @ 07-03-15]
```

Listing 3.14: Deploying the RTAI libraries

3.3.8 Deployment of the real-time linux-distribution

Listing 3.15 shows the partitioning scheme for the real-time linux distribution. The swap partition has been created for non real-time kernels with which paging has been enabled. In the RTAI kernel paging has been disabled.

```
phylos:/% sudo fdisk -l /dev/sdg

Disk /dev/sdg: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdg1   *           1           4       32098+  83  Linux
/dev/sdg2               5        5005    40170532+   5  Extended
/dev/sdg5               5          37      265041   82  Linux swap / Solaris
/dev/sdg6              38         675     5124703+  83  Linux
/dev/sdg7             676        5005    34780693+  83  Linux
```

Listing 3.15: Partition scheme for the system harddisk

The second logical partition (/dev/sdg5) is big enough (5GBs) to carry the real-time linux image. The harddisk is connected to the deployment platform via an USB $\Longleftrightarrow$ IDE adapter. In the next step the filesystem is created on this partition (Listing 3.16).

```
phylos:/% su -
Password:
phylos:~# mkfs -t ext3 -L rtnet-linux /dev/sdg6

      [19:30:58 @ 07-01-22]
```

```
mke2fs 1.40-WIP (14-Nov-2006)
Filesystem label=rtnet-linux
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
641280 inodes, 1281175 blocks
64058 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1312817152
40 block groups
32768 blocks per group, 32768 fragments per group
16032 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 38 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Listing 3.16: Creating the filesystem

After that the system files can be copied with tar (Listing 3.17).

```
phylos:~# mount /dev/sdg6 /mnt/temp
phylos:~# ll /mnt/pontis/images
total 88672
drwxrwxrwx  4 rpoisel users    16384 2007-01-13 11:56 .
drwxrwxrwx 16 rpoisel users    16384 1970-01-01 01:00 ..
-rwxrwxrwx  1 rpoisel users  3223671 2007-01-13 01:12 ant_linux.tar.gz
drwxrwxrwx  4 rpoisel users    16384 2006-11-08 19:21 cdroms
drwxrwxrwx  2 rpoisel users    16384 2006-06-21 00:29 harddisks
-rwxrwxrwx  1 rpoisel users 87502779 2007-01-14 16:02 rt_etch_base.tar.bz2
phylos:~# cd /mnt/pontis/images
phylos:/mnt/pontis/images# tar -tjf rt_etch_base.tar.bz2
./
./initrd.img.old
./sys/
./dev/
./dev/midi1
./dev/ram13
./dev/smpte0
./dev/urandom

...

phylos:/mnt/pontis/images# tar -xjf rt_etch_base.tar.bz2 -C /mnt/temp
```

Listing 3.17: Copying the system files

The next step is to install the boot-manager “Grub” (Listing 3.18).

```
phylos:/mnt/pontis/images# sudo umount /mnt/temp
phylos:/mnt/pontis/images# grub

    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

        [ Minimal BASH-like line editing is supported.  For
          the  first  word, TAB lists possible command
          completions.  Anywhere else TAB lists the possible
          completions of a device/filename. ]

grub> root (hd3,5)
grub> find /boot/
 Possible files are: initrd.img-2.6.18-3-486 config-2.6.18-3-486 config-2.6.19.1-rp-xenomai-
       pentium-mmx System.map-2.6.18-3-486 System.map-2.6.19.1-rp-xenomai-pentium-mmx vmlinuz
-2.6.18-3-486 vmlinuz-2.6.19.1-rp-xenomai-pentium-mmx grub
grub> setup (hd3)
 Checking if ``/boot/grub/stage1''␣exists...␣yes
␣Checking␣if␣``/boot/grub/stage2'' exists... yes
 Checking if ``/boot/grub/e2fs_stage1_5''␣exists...␣yes
␣Running␣``embed /boot/grub/e2fs_stage1_5 (hd3)''... 16 sectors are embedded.
succeeded
 Running ``install␣/boot/grub/stage1␣(hd3)␣(hd3)1+16␣p␣(hd3,5)/boot/grub/stage2␣/boot/grub/menu.
       lst''...␣succeeded
Done.

grub>␣quit
```

Listing 3.18: Installing the boot-manager

The “find” command has been used to see if the selected root partition (“root” command) was correct. After exeucting “setup” the MBR is manipulated so that the harddisk is bootable. The boot-loader and the running Linux system has been configured to be accessable via the serial console (Listing 3.19).

```
phylos:/mnt/pontis/images#
phylos:/% picocom -b 19200 /dev/tts/0
picocom v1.4

port is         : /dev/tts/0
flowcontrol     : none
baudrate is     : 19200
parity is       : none
databits are    : 8
escape is       : C-a
noinit is       : no
noreset is      : no
nolock is       : no
send_cmd is     : ascii_xfr -s -v -l10
receive_cmd is  : rz -vv

Terminal ready

    GNU GRUB  version 0.97  (610K lower / 158720K upper memory)

 +-------------------------------------------------------------------------+
 | Debian GNU/Linux, kernel 2.6.19.1-rp-xenomai-pentium-mmx                |
 | Debian GNU/Linux, kernel 2.6.19.1-rp-xenomai-pentium-mmx (single-user>  |
 | Debian GNU/Linux, kernel 2.6.18-3-486                                   |
 | Debian GNU/Linux, kernel 2.6.18-3-486 (single-user mode)                |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 +-------------------------------------------------------------------------+
      Use the ^ and v keys to select which entry is highlighted.
      Press enter to boot the selected OS, 'e' to edit the
      commands before booting, or 'c' for a command-line.
```

Listing 3.19: Accessing the serial console

After booting the target system, some adaptions have to be done (Listing 3.20).

```
rtk62:~# fdisk -l /dev/hda

Disk /dev/hda: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1           4       32098+  83  Linux
/dev/hda2               5        5005    40170532+   5  Extended
/dev/hda5               5          37      265041   82  Linux swap / Solaris
/dev/hda6              38         675     5124703+  83  Linux
/dev/hda7             676        5005    34780693+  83  Linux
rtk62:~# mkswap /dev/hda5
Setting up swapspace version 1, size = 271396 kB
no label, UUID=f2d3ffa1-8de2-46b6-ab6e-bd0f78ddff5d
rtk62:~# vi /etc/fstab

# UNCONFIGURED FSTAB FOR BASE SYSTEM
/dev/hda6           /             ext3          noatime 0 0
/dev/hda5           none          swap          sw      0 0

rtk62:~# useradd -c ``Rainer␣Poisel''␣-g␣100␣-m␣-s␣/bin/bash␣rpoisel
rtk62:~#␣passwd␣rpoisel
Enter␣new␣UNIX␣password:
Retype␣new␣UNIX␣password:
passwd:␣password␣updated␣successfully
```

Listing 3.20: Final installation steps

The existing swap partition is initialized (“mkswap” command) and populated to the system (stanza in the /etc/fstab file). The last step is to create a non-privileged user for compilation purposes and everyday work.

3.3.9 Deployment of the real-time application framework

One of the computers, running a real-time Linux distribution has been assigned the IP *10.0.0.10*. After setting up the network compiled files, which have been bundled using the *tar* and the *gzip* commandos, can be copied over a standard ethernet/tcp-ip network using the *scp* command (see listing 3.21). On the target platform the archive is then unpacked.

```
phylos:/mnt/win_d/svn/repos/docs/07ws/tm031051_rpoisel/diplomarbeit/source/rt_comm% scp rt_comm.
        tgz root@10.0.0.10:/usr/local/diploma_thesis
root@10.0.0.10's password:
rt_comm.tgz                                                             100%    32KB   31.8KB/s
        00:00
phylos:/mnt/win_d/svn/repos/docs/07ws/tm031051_rpoisel/diplomarbeit/source/rt_comm% ssh -l root
        10.0.0.10
root@10.0.0.10's password:
Last login: Mon Apr  2 21:48:14 2007 from 192.168.6.116
Linux rtceleron 2.6.19.7-rp-rtai-pentium-mmx #1 PREEMPT Wed Mar 21 08:05:44 CET 2007 i686

rtceleron:~# cd /usr/local/diploma_thesis
rtceleron:/usr/local/diploma_thesis# tar -xvzf rt_comm.tgz
rtnet_comm_source
rtnet_comm_sink
qos_comm_source
qos_comm_sink
rtceleron:/usr/local/diploma_thesis# ^D
Connection to 10.0.0.10 closed.
```

Listing 3.21: Using scp to copy the dist-file

3.3.10 Starting RTnet

Computers running RTnet are interfaced using a serial console cable. Thus it is possible to control PCs after decoupling them from the non real-time ethernet network. Since the serial Linux console offers only one session at a time the terminal multiplexer "screen"[3] is used. This procedure has the advantage that a application occupying a terminal (with e.g. traces) can be switched into the background effectively. Listing 3.22 shows how RTnet is invoked on the target system which acts as the TDMA master. Startup scripts for both RTAI and RTnet are provided in the source code section (A.1). Startup of RTAI has been included into the Debian GNU/Linux System V init script scheme of the target machines, so it gets executed on every system boot [Deb06, cmp.]. On the slaves similar commands have been executed.

```
rtk62:~# screen -S rtnet
rtk62:~# /etc/init.d/rtnet start

...

DHCPRELEASE on eth0 to 192.168.6.254 port 67
done.

*** RTnet 0.9.8 - built on Mar 27 2007 21:52:14 ***

RTnet: initialising real-time networking
rt_8139too Fast Ethernet driver 0.9.24-rt0.6
PCI: Found IRQ 11 for device 0000:00:14.0
RTnet: registered rteth0
initializing loopback...
RTnet: registered rtlo
RTcfg: init real-time configuration distribution protocol
RTmac: init realtime media access control
RTmac/TDMA: init time division multiple access control mechanism
Waiting for all slaves...
rtk62:~#
```

Listing 3.22: Starting rtnet

[3]http://www.gnu.org/software/screen/

3.4 Description of the real-time network-testing framework

3.4.1 Organization of the source code

Source code which is mentioned in this section can be found in the Appendix (A.1.1). The real-time framework is organized in a modular way. Source and sink are implemented as separate programs: rt_comm_source.c (Appendix A.5) and rt_comm_sink.c (Appendix A.6). Command-line parsing is accomplished using the `getopt` library in lines 73 to 87 and lines 106 to 162 in rt_comm_source.c (Appendix A.5). System calls regarding the timing behaviour of the application are the ones the RTAI lxrt real-time framework offers [P. 05, cmp.]. Integration of system calls and transition aspects have been considered as described in [P. 04, cmp.].

Networking functions are called through function pointers. This allowes the development of backends with a defined interface. This interface is defined in the common header file rt_comm_backend_ifc.h (Appendix A.7). A backend has therefore to implement the `init_backend()` function in which the function pointers are initialized.
The `init_backend()` function is called before the network functions are being used. This is shown in line 93 of rt_comm_source.c (Appendix A.5) and line 89 in rt_comm_sink.c (Appendix A.6).

With this approach new backends can be implemented with a C-File containing the implementations of the function pointers and the `init_backend()` function. After compilation the object file has to be linked to the resulting binary statically. The program logic only relies on the contents of the `backend_desc` structure which is initialized by the backend itself. Thus the backend can be exchanged without touching the rest of the program logic. This has the advantage of better comparability since the focus in which backends are used is identical among them.

The chosen approach results in a executable file for each backend. Better modularization could have been gained with the use of dynamic libraries (e.g. shared objects on the Linux platform and dynamic link libraries on Microsoft Windows). These libraries allow access to functions which are determined at run time [Dav03, cmp. chapter "Shared Libraries"]. Thus only one binary for all backends would be sufficient. Functions of the backends are then loaded out of shared libraries at run-time.

3.4.2 Access to the parallel port

Access to the parallel port is accomplished with macros which have been defined in parport.h (Appendix A.4). This header is also intended to activate and deactivate access to the parallel port. `USE_PARPORT` set to 1 activates this feature, 0 (zero) deactivates it at compile time. The port base address is hardcoded to 0x378 and since applications are intended to be executed as user-space programs a call to `iopl(2)` is necessary (see the `INIT_PARPORT` macro) to gain the privilege level to do so. This requires the application to be executed with root rights. Changing the port state is done with calls to `outb(2)`. The `TOGGLE_PIN` macro is called with values from 0 to 7 mapping to the data-pins 1 to 8 of the parallel port.

3.4.3 The Source

The source (rt_comm_source.c, Appendix A.5) has been programmed to allow sending of UDP-packets at a specific rate. The rate at which packets get transmitted is determined by the “lDelay” value which can be modified using the command-line parameter “–tx-delay”. Lines 181 and 225 manage the activation of the RTAI hard real-time scheduler which is necessary for some system calls such as `rt_dev_connect()` or `rt_send()`.

3.4.4 The Sink

The sink (rt_comm_sink.c, Appendix A.6) listens at the specified port for UDP-packets (line 150 and 157). The content of these UDP packets is then investigated if it contains the quit message (lines 165 to 169) or if the state of a specific pin of the parallel port has to be changed (lines 171 to 179). As with the source, calls to the RTAI scheduler are made to gain real-time timing behavior (lines 147 and 184).

3.5 RTnet - Latency Measurements

This experiment aims to acquire latency in a deterministic network. As a result it should be possible to determine the probability to gain specific latency constraints.

3.5.1 Using TDMA networking functions

RTnet offers a similar interface to networking as the POSIX standard does [J. 05b, cmp. p. 3]. Thus it is possible to use the same data types as with the POSIX functions. The file rt_comm_rtnet.c shows how function pointers have been initialized to use rtnet for data exchange. With this module all function pointers act as wrappers for RTnet networking functions.

3.5.2 Gathering latency data

After starting all participating nodes (one master and two slaves) as described in 3.3.10 programs accomplishing communication can be started.

```
rtk62:/usr/local/diploma_thesis# ./rtnet_comm_sink --port 1234
Starting hard real-time receive process. All your base are belong to us!
Quit message received, shutting down therefore. For great justice.

Statistics:
===========
PIN1: 300
PIN2: 11474
PIN3: 0
PIN4: 0
PIN5: 0
PIN6: 0
PIN7: 0
PIN8: 0
1 Quit message received.
```

Listing 3.23: Invoking the application on the master node

In listing 3.23 the packet sink waits for incoming UDP packets until a message containing the string “42” arrives. Then a statistic is printed to see if all packets arrived at their destination. On one of the client machines the source is started (listing 3.24) with a given number of messages (300) to transmit.

```
rtceleron:/usr/local/diploma_thesis# ./rtnet_comm_source --server-port 1234 --server-ip 10.0.0.1 \
--message PIN1 --toggle-pin 2 --num-messages 300 --num-failure-messages 20 --tx-delay 3000000
```

Listing 3.24: Invoking the application on the second slave node

The second client machine is used to cause additional load during the deterministic transmission (listing 3.25). Thus a larger number of messages to transmit is provided (1000000) to occupy media as much and as long as possible.

```
rtgeode:/usr/local/diploma_thesis# ./rtnet_comm_source --server-port 1234 --server-ip 10.0.0.1 \
--message PIN2 --toggle-pin 1 --num-messages 1000000 --num-failure-messages 20 --tx-delay 3000000
```

Listing 3.25: Invoking the application on the first slave node

Figure 3.5 and 3.6 demonstrate how latency values have been gathered using the “cursor” function of the persistence oscilloscope. The signal level of one of the parallel port datapins changes with each sent or received packet. These datapins are connected to the persistence oscilloscope: signal 1 to the communication source and signal 2 to the sink. The communication source (signal 1) sends a specific number of packets towards the communication sink (signal 2). Therefore each edge of the signal displays either a packet which has been sent by the communication source (signal 1) or a packet which has been received at the sink (signal 2). As such the time between transmission and the reception of a packet (also known as *latency*) can be stated as the period between two edges (e.g. the period between cursor 1 and cursor 2). In this case at least 50 latency values have been taken per cycle time. In the next step these values can be statistically evaluated.

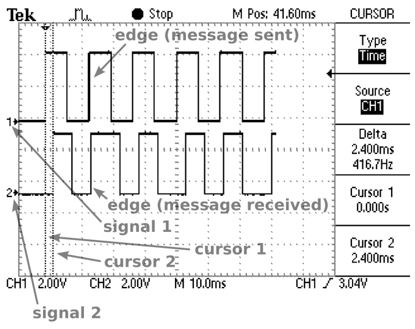


Figure 3.5: Latency measured during a $3ms$ cycle

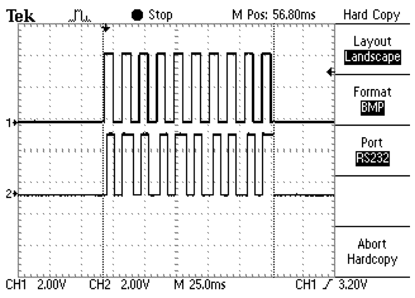


Figure 3.6: 20 sent and received messages in hard real-time mode

3.5.3 Measurement results

Statistical analysis has been performed using the statistical software project “R”[4]. The accomplishment of analysis for 3000μs cycle time is presented in listings 3.26 and 3.28. These two files have been concatenated together to supply the function for other cycle times too. Latency times for other cycle times have been acquired with tally sheets as shown in line 3 in listing 3.27 (line 3).

[4]http://www.r-project.org

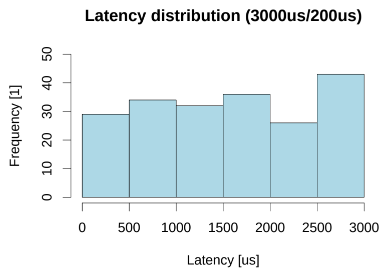


Figure 3.7: Distribution of latencies with RTnet at a cycle time of 3000μs

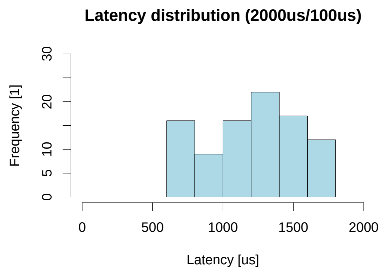


Figure 3.8: Distribution of latencies with RTnet at a cycle time of 2000μs

```
### 3000us cycle time, 200us tdma offset
lCycleTime<-"3000"
lLatency <- scan("../data/latency.dat")
printhistpdf(lLatency,paste("rtnet_latency_",lCycleTime,".pdf",sep=""),3000,50,"3000","200")
```

Listing 3.26: rtnet_latency_3000.R

```
### 2000us cycle time, 100us tdma offset
lCycleTime<-"2000"
lLatency <- c(rep(600,times=8),rep(800,times=8),rep(1000,times=9),rep(1200,times=16),rep(1400,
        times=22),rep(1600,times=17),rep(1800,times=12))
printhistpdf(lLatency,paste("rtnet_latency_",lCycleTime,".pdf",sep=""),2000,30,"2000","100")
```

Listing 3.27: rtnet_latency_2000.R

```
printhistpdf=function(pDistribution,pFilename,pUpXlim,pUpYlim,pCycleTime,pTDMAOffset,pBars=6,
        pLowXlim=0,pLowYlim=0)
{
        x11()
        par(cex=2.0)
        hist(pDistribution,main=paste("Latency distribution (",pCycleTime,"us/",pTDMAOffset,"us)",
                sep=""),col="Light Blue",br=pBars,plot=TRUE,xlim=c(pLowXlim,pUpXlim),ylim=c(pLowYlim,
                pUpYlim),xlab="Latency [us]",ylab="Frequency [1]")
        printpdf(pFilename)
}

printpdf=function(pFilename)
{
        dev.copy(device=x11)
        dev.print(pdf,pFilename,width=10.0,height=7.5)
        dev.off(dev.prev())
        dev.off()
}
```

Listing 3.28: functions.R

Further investigations have been carried out to determine if shorter cycle times lead to distributions with more frequent latencies similar to the configured cycle time. The distributions are shown in figures 3.7 to 3.10. The lowest configured cycle time of 400μs led to latency times of up to 800μs. As such with a cycle time of 400μs this system is only soft real-time capable. With a cycle time of $1ms$ all messages have been received on time. None of the measured latency times was longer than $1ms$.

The difference between figure 3.7 and 3.8 looks like someone might expect it - With a cycle time of $2ms$ the density of latencies is higher at the value of the cycle time. With a communication cycle of $2ms$ packets have less time to arrive at their destination than with $3ms$. The distribution at a cycle time of $1ms$ shows that there is no such relation. Latencies are distributed uniformly again over the cycle time as was the case with $3ms$.

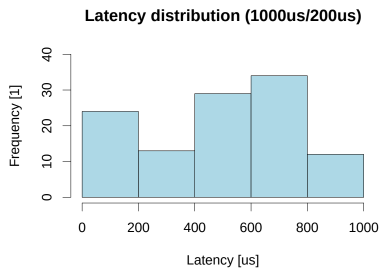


Figure 3.9: Distribution of latencies with RTnet at a cycle time of $1000\mu s$

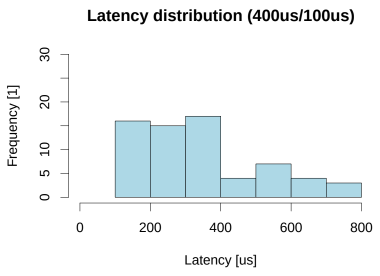


Figure 3.10: Distribution of latencies with RTnet at a cycle time of $400\mu s$

3.5.3.1 Conclusion

Regarding the definition for accurate timing which has been stated in section 2.3.1.1, the RTnet implementation reached its goals. Measured latency was never longer than $1ms$ with a cycle time of $1ms$. As such assuming a maximum latency of $1ms$, the timing-behavior of data transmission is predictable in this network. A latency of $1ms$ conforms to class “E” for latency of the IOANA Realtime Classification (IRC) [LM06, cmp. p. 2].

Further investigations could be done to predict the behavior of the system in dependency of ambient conditions. [Ric05] describes how to fit measured data to statistical distributions. This allows to calculate the accuracy of hard real-time capabilities. Predictions could be made regarding the propability of a packet to arrive at its destination on time.

Another approach could be to determine the relation between specific parameters and the timing behavior of the system. [Lot01, cmp. pp. 689 – 722] explains regression curves. This method is a mathematical description of the relation between random variables. Based on measured data it can be applied to predict the timing behavior of the communication system with changed ambient conditions.

3.6 QoS - Latency Measurements

Figure 3.2 depicts how the systems where set up to exchange prioritized traffic. Both classification and handling of classified traffic were carried out on a Cisco 2950 Series switch. Since the server (the communication sink) has been connected to the Cisco switch using the trunk port, it had to support the IEEE 802.1Q standard too.

Both [Tho05, cmp. chapter 9.1] and [Cis07a, cmp.] explain that in this correlation queuing disciplines only determine the way how data is sent. Further the destination node is not possible to control the amount of data which is to be sent to it.

The Cisco 802.1Q QoS implementation The configuration of VLANs has been carried out according to the steps mentioned in [Cis07b, cmp.]. This is shown in listing 3.29.

```
Switch# vlan database
Switch(vlan)# vlan 20 name real-time
Switch(vlan)# exit
APPLY completed.
Exiting....
Switch#
```

Listing 3.29: VLAN configuration on a Cisco 2950 Series Switch

Interfaces can be assigned to an existing VLAN. Listing 3.30 shows this part of the *running configuration* of the used Cisco switch. Interface “FastEthernet0/4” has been used to enable an internet connection for nodes in the network. It was used as an uplink-port. Interface “FastEthernet0/3” has been configured as a trunk port. The server (or communication sink) has been connected to this interface. Frames leaving the switch through this port still have the extended 802.1Q ethernet frame format.

```
interface FastEthernet0/1
switchport access vlan 20
switchport mode access
switchport priority extend cos 7
spanning-tree portfast
!
interface FastEthernet0/2
switchport access vlan 20
switchport mode access
switchport priority extend cos 3
spanning-tree portfast
!
interface FastEthernet0/3
switchport access vlan 20
switchport mode trunk
spanning-tree portfast
!
interface FastEthernet0/4
switchport access vlan 20
switchport mode access
spanning-tree portfast
```

Listing 3.30: Assigning interfaces to VLANs

With this experiment the standard image “SI” of the switch operating-system “IOS” has been used. It has limited support for classification of ethernet traffic [Cis07a, cmp.]. CoS values are assigned based on the switchport where ethernet frames are received. The configuration statements are shown in listing 3.30. After issuing the commands the configuration can be verified 3.31. The output of this status command does not print configured trunk ports.

```
Switch#sh vlan
VLAN Name                                 Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                              active    Fa0/6, Fa0/8, Fa0/9, Fa0/10,
                                                    Fa0/11, Fa0/12
20   real-time                            active    Fa0/1, Fa0/2, Fa0/4
...
```

Listing 3.31: Verifying the VLAN configuration

IOS of the Cisco switch implements a weighted round-robin scheduling. This approach divides traffic of an ethernet interface into four queues which can be prioritized. [Cis07a, cmp.] states that the weight value determines how many packets are transmitted for every other weight of a different queue. On Line 1 in listing 3.32 all queues have been assigned the same weight. As such each packet is handled with the same priority. Line 2 and 3 assign CoS values 7 and 3 to different WRR queues. This is necessary to make sure that traffic coming from these two clients is handled separately.

```
Switch(config)#wrr-queue bandwidth 255 255 255 255
Switch(config)#wrr-queue cos-map 1 7
Switch(config)#wrr-queue cos-map 2 3
```

Listing 3.32: Prioritization of classified traffic on a Cisco 2950 Series Switch

The complete configuration parameters of the Cisco Catalyst 2950 switch are shown in listing A.12 in the appendix.

The Linux 802.1Q QoS implementation In the Linux operating system the 802.1Q standard is implemented as a kernel-module. It is called “8021q.ko” on Linux 2.6 and ”8021q.o” on Linux 2.4 respectively. After loading the module using `modprobe` or `insmod` it is possible for the Linux system to handle ethernet frames that have been tagged using the 802.1P and 802.1Q header respectively.

Listing 3.33 shows the commands which have been executed to enable VLAN support for the running Linux kernel using the `vconfig` tool.

```
rtk62:~# modprobe 8021q
802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
All bugs added by David S. Miller <davem@redhat.com>
rtk62:~# vconfig add eth1 20
Added VLAN with VID == 20 to IF -:eth1:-
rtk62:~# cat /proc/net/vlan/config
VLAN Dev name    | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
eth1.20  | 20  | eth1
rtk62:~# cat /proc/net/vlan/eth1.20
eth1.20  VID: 20  REORDER_HDR: 1  dev->priv_flags: 1
         total frames received           26
          total bytes received         2109
      Broadcast/Multicast Rcvd            8

      total frames transmitted            0
       total bytes transmitted            0
            total headroom inc            0
           total encap on xmit            0
Device: eth1
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0 7:0
EGRESSS priority Mappings:
rtk62:~# ifconfig -a
...
eth1      Link encap:Ethernet  HWaddr 00:02:44:45:E5:A2
          inet6 addr: fe80::202:44ff:fe45:e5a2/64 Scope:Link
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:37745 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14514 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8001984 (7.6 MiB)  TX bytes:1406908 (1.3 MiB)
          Interrupt:11 Base address:0x8e00

eth1.20   Link encap:Ethernet  HWaddr 00:02:44:45:E5:A2
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:115 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7715 (7.5 KiB)  TX bytes:0 (0.0 b)
...
```

Listing 3.33: Enabling 802.1Q on an interface with Linux

After addition of the VLAN a new interface “eth1.20” was made available. On this interface ethernet frames from which the VLAN header information has been removed are received. Thus it is assigned an IP address (*10.0.0.1* in this case). The original interface “eth1” still receives tagged ethernet frames. It is not assigned an IP address and put into promiscuous mode to accept all frames. Listing 3.34 shows the configuration commands.

```
rtk62:~# ifconfig eth1 0 promisc
rtk62:~# ifconfig eth1.20 10.0.0.1 netmask 255.0.0.0 up
```

Listing 3.34: Configuration of additional VLAN-interfaces

3.6.1 Using standard networking functions

Listing A.9 in the appendix shows that POSIX networking functions have been wrapped so that they are usable as function pointers. Further difference to the RTnet module shown in A.8 is, that `rt_make_soft_real_time()` and `rt_make_hard_real_time()` functions are not called (function bodies are empty) because POSIX networking functions are implemented as standard system calls which may not occur when hard real-time is activated.

The `rt_sleep()` function call had to be replaced with `rt_busy_sleep()` to work correctly. This influenced the period between packet transmission, but not latency as packets have to be processed on a dedicated computer which is not in the busy waiting state. It caused more cpu-time to be used by the real-time processes during waiting phases on the communication source.

3.6.2 Gathering latency data

```
rtk62:/usr/local/diploma_thesis# ./qos_comm_sink --port 1234
Starting hard real-time receive process. All your base are belong to us!
Quit message received, shutting down therefore. For great justice.

Statistics:
===========
PIN1: 300
PIN2: 2000000
PIN3: 0
PIN4: 0
PIN5: 0
PIN6: 0
PIN7: 0
PIN8: 0
1 Quit message received.
```

Listing 3.35: Invoking the application on the master node

```
rtceleron:/usr/local/diploma_thesis# ./qos_comm_source --server-port 1234 --server-ip 10.0.0.1 \
--message PIN1 --toggle-pin 2 --num-messages 300 --num-failure-messages 20 --tx-delay 1000000000
```

Listing 3.36: Invoking the application on the first slave node

The second client machine is used to cause additional load during the transmission (listing 3.37). Thus a larger number of messages to transmit is provided (1000000) to occupy media as much and as long as possible.

```
rtgeode:/usr/local/diploma_thesis# ./qos_comm_source --server-port 1234 --server-ip 10.0.0.1 \
--message PIN2 --toggle-pin 1 --num-messages 1000000 --num-failure-messages 20 --tx-delay
    1000000000
```

Listing 3.37: Invoking the application on the second slave node

Figure 3.11 and 3.12 show how latency has been measured in the QoS network. The source transmits 300 packets towards the sink which change the state of the first databit pin on both the source and the sink. Latency is then recorded through the parallel ports which are connected to a persistence oscilloscope.

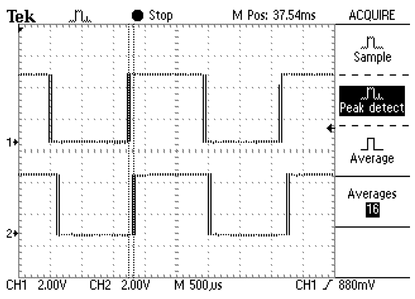


Figure 3.11: Latency with one slave

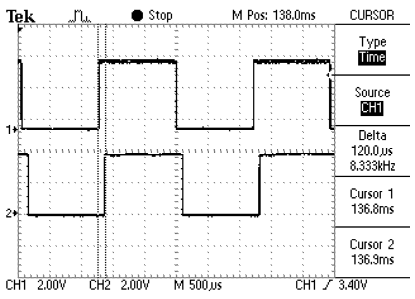
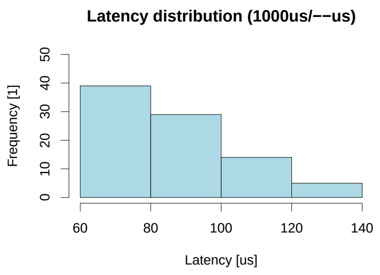


Figure 3.12: Latency with two slaves



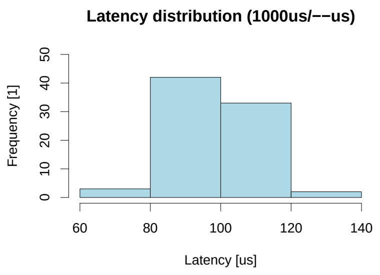


Figure 3.13: Latency distribution with one slave

Figure 3.14: Latency distribution with two slaves

3.6.3 Measurement results

Histograms show that latency values are lower than with the experiments in 3.5.3. The histograms in 3.13 and 3.14 display that latency increases with the number of nodes occupying the medium at the same time. The distribution shown in figure 3.14 shows more density of latency values in upper levels. This could be caused by the medium which has to be shared between the one more node as in figure 3.13. This is the same behavior which has been expected in 3.5.3. Further investigations could prove this relation.

3.6.3.1 Conclusion

This experiment showed that a QoS network with three nodes which has been implemented using the standard POSIX networking functions fulfills the requirements for accurate timing in a hard real-time network. Here measured latency was lower than with the RTnet experiment.

In an industrial environment much more clients occupy the medium at the same time. Therefore it would be necessary to prove that this concept still works under these circumstances. Another approach would be to simulate the timing behavior in a network with a mutlitude of nodes accessing the same medium.

Chapter 4

Real-Time Ethernet applied - Industrial solutions

In 2.10.2 *RTnet* has been introduced. This prototype project has not been standardized yet. Therefore it has just academical relevance. A table in [WB05, p. 313] shows a comparison of widespread standardized real-time capable networks.

4.1 Technology overview

[CS06, cmp. p. 1] mentions that the IEC has accepted 10 suggestions at the time of its writing (April 2006) for ethernet based industrial communication protocols (see table 4.1).

SPECIFICATION NAME	IEC STANDARD
EPA	IEC / PAS 62409
EtherCAT	IEC / PAS 62407
EtherNet/IP	IEC / PAS 62413
ETHERNET Powerlink	IEC / PAS 62408
MODBUS-RTPS	IEC / PAS 62030
P-NET on IP	IEC / PAS 62412
PROFINET IO	IEC / PAS 62411
SERCOS III	IEC / PAS 62410
TCnet	IEC / PAS 62406
Vnet/IP	IEC / PAS 62405

Table 4.1: Pre-standards of ethernet based automation protocols

Further Schwab and Lüder mention that in each of the pre-standards ethernet is adapted by software- and/or hardware-extensions to meet requirements (mainly real-time behavior and determinism) for automation technology:

- protocols which are implemented on top of layer 3 of the OSI-reference model (the stack-model on the left of figure 4.1)

- protocols which replace layers 3 and 4 of the OSI-reference model (the stack-model in the middle of figure 4.1)
- protocols which modify layers 2 to 4 of the OSI-reference model (the stack-model on the right of figure 4.1)

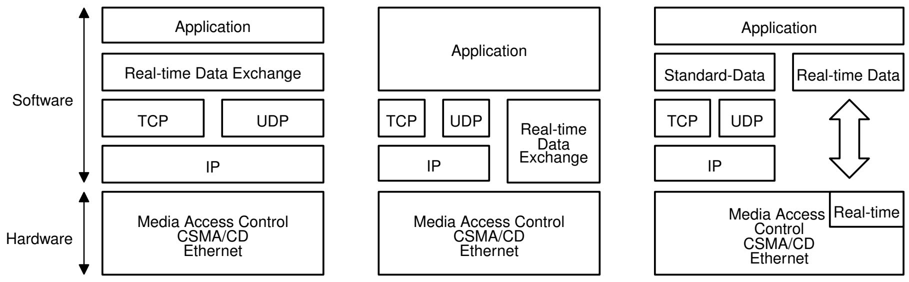


Figure 4.1: Stack modifications in real-time ethernet networks [CS06, cmp. p. 1]

This document focuses on industrial ethernet solutions that are most relevant [CS06, cmp. p. 1] for the European automation market.

4.2 ETHERNET Powerlink

ETHERNET Powerlink extends the operating systems layer 3 and 4 implementation of the OSI-stack with an additional middleware for isochronous, cyclic data exchange. This corresponds to the graphic in the middle of figure 4.1. Asynchronous data is exchanged through an additional layer between layer 2 and 3 of the OSI-model [CS06, cmp. p. 2]. These changes in the standard networking-stack implementations still allow the usage of standard ethernet components [cTA06, cmp. p. 6].

ETHERNET Powerlink can be operated in two modes: open and protected mode [WB05, cmp. p. 304]. The latter relies on a special protocol suite and does not allow the usage of standard protocols such as TCP, UDP or IP in general. Open mode is less real-time capable than protected mode, but enables the usage of standard protocols.

Access to the media is centrally controlled by the “Managing Node” (MN) [WB05, cmp. p. 303pp.]. Thus ETHERNET Powerlink is organized as a monomaster system with centralized slot allocation. [cTA06, cmp. p. 8] explains how a base-cycle in protected mode is organized. And [WB05, cmp. p. 304] gives details about each phase:

- start phase: a broadcast message (Start-of-Cycle, SoC) synchronizes all participating nodes.
- isochronous phase: the communication master requests every participating node with a unicast message (PollRequest, PReq) to transmit their gathered data; each node which has been requested broadcasts its data in a PollResponse (PRes); after querying each node the managing node broadcasts an End-of-Cycle (EoC) message.

- asynchronous phase: is used for ad-hoc data to be placed onto the bus; during this phase standard protocols such as TCP, UDP and IP can be transmitted over real-time media; nodes which want to send data during the asynchronous phase have to signal this circumstance in a flag of the PollResponse message. Upon reception of the AsyncInvite message nodes start to transmit their ad-hoc data.

[cTA06, cmp. p. 9] further mentions a third operation mode – multiplex mode. In this case isochronous data slots may be used by nodes only every nth communication cycle. Thus bandwidth can be preserved and better latency can be achieved.

4.2.1 Conclusion

As with RTnet, ETHERNET Powerlink follows the TDMA approach regarding media access. The underlying CSMA/CD media access is still active. No collisions will occur because the master enables only a single node to occupy the media at a time [cTA06, cmp. p. 8]. During the "open" operational mode it is possible to use standard network protocols such as TCP, UDP and IP.

4.3 Ethernet/IP

[ODV06, cmp. p. 2] states that "EtherNet/IP is a member of a family of networks that implements the Common Industrial Protocol (CIP)". CIP has been developed as a network independent protocol. Other networks beneath the CIP layer may be DeviceNet for example [VS06, cmp. p. 2]. Secure data transmission such as program up- and download or the transmission of configuration data is implemented using the TCP . Control and time sensitive data is conveyed using the UDP .

[CS06, cmp. p. 2] mentions that EtherNet/IP is implemented above the TCP/IP and UDP/IP layers of an operating system respectively. Thus it is possible to offer an EtherNet/IP implementation as an additional software component which does not modify elements of the target operating system. The role of networking layers in this regard is depicted in figure 4.1 on the left.

This implementation is only soft real-time capable. This can be led back to the usage of standard ethernet with factors such as full queues and/or collisions as described in and 2.4.3.2.

4.4 EtherCAT

EtherCAT technology has been inventented and users incorporated to the EtherCAT Technology Group (ETG) [WB05, cmp. p. 306].

The publication further mentions that EtherCAT participants are connected together in a logical bus with duplex lines. Physical arrangements may be the line or tree topology. A single master is responsible for synchronization and message exchange.

The EtherCAT protocol is optimized for efficient bandwidth utilization [Gro06, cmp. p. 9]. Messages originate from the master and are handed from one node to the other. Thus only one header for all nodes in an EtherCAT domain is appened to transmitted data within a communication cycle. The usable data rate can be up to 90% of channel bandwidth. Slave nodes require special hardware - the Fieldbus Memory Management Unit (FMMU) which receives and interprets the messages. If a slave node has been addressed in the message the corresponding part of the telegram is copied into the main memory of the slave to be handled by its operating software. The content of messages conveyed on the EtherCAT network get modified when they get handed through a node [Ros06, cmp. p. 1].

For the interpretation of telegrams at least on the slave side, this technology requires stack modifications as shown on the most right of figure 4.1. For the master-node software-only implementations exist (e.g. http://www.etherlab.org by the German company IgH).

4.5 PROFINET

PROFINET is a part of standards IEC 61158 and IEC 61784 [e.V06, cmp. p. 1]. Since PROFINET V2, ethernet has been chosen as the only data link protocol. While PROFINET V2 has been implemented in software, PROFINET V3's real-time capabilities are based on hardware extensions [CS06, cmp. p. 2].

Transmission of data with PROFINET can be divided into three subgroups: non time-critical traffic, real-time traffic and isochronous real-time traffic [e.V06, cmp. p. 3]. PROFINET V3 like EtherCAT requires stack modifications as shown in the right graphic of figure 4.1. PROFINET V2 can be categorized into the protocols which are completely implemented in software, as such it corresponds to the middle graphic in figure 4.1 [CS06, cmp. p. 1].

Non time-critical traffic (TCP/IP and UDP/IP traffic) Communication in this mode occurs with protocols conforming to international standards such as IEEE 802.3 (ethernet), TCP/IP and UDP/IP respectively. TCP/IP ensures correct order or transmitted data. This feature is also called “flow-control”. As such, this mode is mainly used to transfer configuration data [e.V06, cmp. p. 1].

Real-time traffic (RT) is used to convey time-critical process-data in a production environment . In this mode IP-addressing is only used partially [e.V06, cmp. p. 1]. RT-communication happens parallel to transmission of non time-critical data. The following scenarios are considered:

- RT-communication within the same network-segment: only ethernet is used. This can be determined by ethertype “0x8892” of received messages. Cycle times of less than 10 milliseconds can be achieved with this approach [WB05, cmp. p. 309].

- RT-communication between different network-segments: routing-information is required to determine the target network. Here time-sensitive data is conveyed over the UDP protocol (“RT over UDP”).
- RT-communication with multiple participants: cyclic data exchange occurs with ethernet multicast in this case.

Isochronous real-time traffic mode (IRT) is intended to convey time-sensitive data with requirements for deterministic timing-behaviour such as data for motion control applications. No routing is possible and as such data transmission can only occur within the same network segment.

Bus cycles are devided into the so called “red interval” for isochronous data transmission and into the “green interval” for RT, TCP/IP and UDP/IP transmission respectively. The PTCP-Protocol is used for synchronization of participants and communication elements such as switches [WB05, cmp. p. 310]. Further special ASICs are required for transmission in IRT mode. The changeover from red to the green interval is controled by hardware [e.V06, cmp. p. 4]. Bandwidth has to be divided into a IRT part and a part for TCP/IP, UDP/IP and RT communication.

4.5.1 Integration of existing fieldbusses

PROFINET supplies a modell for integration of existing PROFIBUS, INTERBUS and DeviceNet fieldbus installations. The intersection from PROFINET to fieldbus networks occurs with the help of proxies. This device represents fieldbus devices as ethernet devices for the PROFINET network and PROFINET devices as fieldbusdevices on the fieldbus [e.V06, cmp. p. 22].

4.6 MODBUS/TCP

“MODBUS is an application layer messaging protocol positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of busses or networks.” [MI06a, p. 2]. MODBUS-RTPS is to be standardized in the international standard IEC PAS 62030.

Currently the following busses or networks are supported as underlying communication infrastructure:

- TCP/IP over Ethernet
- Asynchronous serial transmission over different media types (EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fiber, radio, etc.)
- token passing networks such as MODBUS PLUS

To gain independcy of the underlying communication technology the MODBUS protocol defines a simple Protocol Data Unit (PDU) which is media independent. This unit is then encapsulated into an Application Data Unit (ADU) which contains e.g.

physical addressing information [MI06b, cmp. p. 4]. [CS06, cmp. p. 2] states that MODBUS-RTPS, which is a different name for MODBUS/TCP, is based on ethernet and standard TCP/IP and implemented on top of layer 4 of the OSI model. Data is exchange occurs connection oriented and as such integrity of data is ensured during transmission.

MODBUS is implemented as a client/server communication infrastructure [MI06b, cmp. p. 2]. The client/server model is based on four types of messages:

- MODBUS Reqeust: sent by the client towards the server
- MODBUS Indication: request message received by the server
- MODBUS Response: message sent by the server
- MODBUS Confirmation: response message received by the client

As a level 7 protocol MODBUS-RTPS is intended to provide an abstract data model which can be used to transmit the process image (see [MI06a, cmp. p. 3pp]. There is no synchronization of clients and the server. Thus it is not possible to overcome the limitations of classical ethernet (e.g. collisions, full queues in communication elements). MODBUS-RTPS conforms to the left picture in figure 4.1.

4.7 Sercos III

Sercos III uses the TDMA approach on ethernet networks [CS06, cmp. p. 2]. Transmission can be devided into two transmission modes: real-time and non real-time communication.

4.7.1 Topology

Sercos III is based on a ring topology. Because of duplex capabilities of ethernet technology the ring is two-way. Data paths are therefore redundant. Using the line topology is also possible but the advantage of redundancy is not available then [Lut06, cmp. p. 1].

4.7.2 Real-Time communication

The ethertype for frames transmitted over the real-time channel is “0x88CD”. The non real-time service channel is used for transmission of parameters and diagnostic data.

At the beginning of a communication cycle the master sends a broadcast message (master-synchronizationtelegram, “MST”) to synchronize clocks of participants. At the initialization phase time slots have been associated with clients. During these time slots clients send their data (actual value) towards the communication master in amplifier telegrams “AT”. After that the server transmits its set points towards the clients in a master data telegram “MDT” [WB05, cmp. p. 311].

Whereas non real-time data is transmitted in standard ethernet frames, real-time data is conveyed in adapted ethernet frames [WB05, cmp. p. 312].

4.8 Prospect of industrial solutions

4.8.1 Organization in interest groups

All mentioned approaches in the industrial environment are organized in so called interest groups which in turn are registered clubs. Members of these clubs are often enabled to participate when technology decisions have to be met. Further access to documents regarding the technology is restricted to members too.

One of the largest interest groups is PROFIBUS International. It has 1300 members (e.g. manufacturing companies, integrators, end users, institutes, etc.) worldwide [e.V06, cmp. p. 27]. Ethernet Powerlink follows with 400 members [cTA06, cmp. p. 5] and 150.000 nodes which are installed in 25.000 different factory machines. The EtherCAT Technology Group comprises 300 members [Ros06, cmp. p. 5] and the Sercos interest group has approximately 60 members [Lut06, cmp. p. 1] and 1.5 million nodes which implement this protocol.

4.8.2 Compatibility and Enhancements

Some protocols can be seen as a successor of previous fieldbus protocols. Modbus/TCP is an implementation of the Modbus protocol upon a TCP/IP stack [MI06a, cmp. p. 2]. The application layer of Ethernet Powerlink is based on the CANOpen protocol [cTA06, cmp. p. 10]. PROFINET offers interfaces to existing fieldbus technologies such as PROFIBUS, AS Interface and Interbus [e.V06, cmp. p. 2].

Using existing protocols for the application layer has the advantage of interoperability and exchangeability of components from different vendors [cTA06, cmp. p. 10].

4.8.3 Fields of application

[WB05, cmp. p. 312–313] mentions fields of applications for industrual real-time ethernet implementations. Ethernet Powerlink can be used in scenarios which require standard ethernet hardware and Sercos III is intended to be used in motion control scenarios where lowest latency times are a requirement. As of the writing of [WB05] in 2005 PROFINET and Sercos III were only announced standards.

Appendix A

Source Code

A.1 Managing RTAI and RTnet

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: RTAI
# Description:       Linux Real-Time extensions start script
#
### END INIT INFO

# Author: Rainer Poisel <tm031051@fh-stpoelten.ac.at>

# Do NOT "set -e"

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="Sets my settings"
NAME=rtai
SCRIPTNAME=/etc/init.d/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure that this file is present.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started

        # create devices
        mknod -m 666 /dev/rtai-shm c 10 254 >/dev/null 2>&1
        for n in `seq 0 9`; do mknod -m 666 /dev/rtf$n c 150 $n >/dev/null 2>&1; done

        # load real-time modules
        insmod /usr/local/rtai/modules/rtai_hal.ko >/dev/null 2>&1
        insmod /usr/local/rtai/modules/rtai_lxrt.ko >/dev/null 2>&1
        insmod /usr/local/rtai/modules/rtai_sem.ko >/dev/null 2>&1
        insmod /usr/local/rtai/modules/rtai_rtdm.ko >/dev/null 2>&1

        return 0
}

#
# Function that stops the daemon/service
#
do_stop()
{
        # Return
        #   0 if daemon has been stopped
        #   1 if daemon was already stopped
        #   2 if daemon could not be stopped
        #   other if a failure occurred

        rmmod rtai_rtdm >/dev/null 2>&1
        rmmod rtai_sem >/dev/null 2>&1
        rmmod rtai_lxrt >/dev/null 2>&1
        rmmod rtai_hal >/dev/null 2>&1

        return 0
}

#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
        #
        # If the daemon can reload its configuration without
        # restarting (for example, when it is sent a SIGHUP),
        # then implement that here.
        #
        echo -n ""
}

case "$1" in
  start)
        [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
        do_start
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  stop)
        [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
        do_stop
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  #reload|force-reload)
        #
        # If do_reload() is not implemented then leave this commented out
        # and leave 'force-reload' as an alias for 'restart'.
        #
        #log_daemon_msg "Reloading $DESC" "$NAME"
        #do_reload
        #log_end_msg $?
        #;;
  restart|force-reload)
        #
        # If the "reload" option is implemented then remove the
        # 'force-reload' alias
        #
        log_daemon_msg "Restarting $DESC" "$NAME"
        do_stop
        case "$?" in
          0|1)
                do_start
                case "$?" in
                        0) log_end_msg 0 ;;
                        1) log_end_msg 1 ;; # Old process is still running
                        *) log_end_msg 1 ;; # Failed to start
                esac
                ;;
          *)
                # Failed to stop
                log_end_msg 1
                ;;
        esac
        ;;
  *)
        #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
esac

:
```

Listing A.1: rtai script

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:        0 1 6
# Short-Description: RTnet
# Description:         Real-Time ethernet start script
#
### END INIT INFO

# Author: Rainer Poisel <tm031051@fh-stpoelten.ac.at>

# Do NOT "set -e"

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="Sets my settings"
NAME=rtnet
SCRIPTNAME=/etc/init.d/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure that this file is present.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started

        # Add code here, if necessary, that waits for the process to be ready
        # to handle requests from services started subsequently which depend
        # on this one.  As a last resort, sleep for some time.

        # turn off networking
        /etc/init.d/networking stop

        # remove non real-time modules
        rmmod 8139too >/dev/null 2>&1
        rmmod mii >/dev/null 2>&1

        # create device nodes
        mknod /dev/rtnet c 10 240

        # load real-time modules
        /usr/local/rtnet/sbin/rtnet start
}

#
# Function that stops the daemon/service
#
do_stop()
{
        # Return
        #   0 if daemon has been stopped
        #   1 if daemon was already stopped
        #   2 if daemon could not be stopped
        #   other if a failure occurred

        # unload real-time modules
        /usr/local/rtnet/sbin/rtnet stop

        # remove device nodes
        rm /dev/rtnet >/dev/null 2>&1

        # load non real-time modules
        modprobe mii
        modprobe 8139too

}

#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
        #
        # If the daemon can reload its configuration without
        # restarting (for example, when it is sent a SIGHUP),
        # then implement that here.
        #
        echo -n ""
}

case "$1" in
  start)
        [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
        do_start
		case "$?" in
			0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
			2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
		esac
		;;
  stop)
		[ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
		do_stop
		case "$?" in
			0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
			2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
		esac
		;;
  #reload|force-reload)
		#
		# If do_reload() is not implemented then leave this commented out
		# and leave 'force-reload' as an alias for 'restart'.
		#
		#log_daemon_msg "Reloading $DESC" "$NAME"
		#do_reload
		#log_end_msg $?
		#;;
  restart|force-reload)
		#
		# If the "reload" option is implemented then remove the
		# 'force-reload' alias
		#
		log_daemon_msg "Restarting $DESC" "$NAME"
		do_stop
		case "$?" in
		  0|1)
			do_start
			case "$?" in
				0) log_end_msg 0 ;;
				1) log_end_msg 1 ;; # Old process is still running
				*) log_end_msg 1 ;; # Failed to start
			esac
			;;
		  *)
			# Failed to stop
			log_end_msg 1
			;;
		esac
		;;
  *)
		#echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2
		echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
		exit 3
		;;
esac

:
```

Listing A.2: rtnet script

A.1.1 The real-time data transmission Testing framework

A.1.1.1 The build script

```
# File:         Makefile
# Author:       Rainer Poisel <tm031051@fh-stpoelten.ac.at>
# Description:  Makefile for the RT Communication source and sink
# Created:      March 31, 2007
# Hints:

CC=cc
TAR=tar
STRIP=strip
RTAI_CONFIG_PREFIX=/usr/local/rtai/bin
CFLAGS=-c -g -Wall -I/usr/src/linux `$(RTAI_CONFIG_PREFIX)/rtai-config --lxrt-cflags` \
                        -D_REENTRANT -Iinclude
LDFLAGS=-lpthread `$(RTAI_CONFIG_PREFIX)/rtai-config --lxrt-ldflags`
TARFLAGS=-cpzvf
ADDLIBS=
DIST=rt_comm.tgz
SRCDIST=rt_comm_src.tgz
RTNET_SOURCE_BIN=rtnet_comm_source
RTNET_SOURCE_OBJ=rt_comm_source.o rt_comm_error.o rt_comm_rtnet.o
RTNET_SINK_BIN=rtnet_comm_sink
RTNET_SINK_OBJ=rt_comm_sink.o rt_comm_error.o rt_comm_rtnet.o
RTNET_PARPORT_BIN=rtnet_parport
RTNET_PARPORT_OBJ=rtnet_parport.o rt_comm_error.o rt_comm_rtnet.o
QOS_SOURCE_BIN=qos_comm_source
QOS_SOURCE_OBJ=rt_comm_source.o rt_comm_error.o rt_comm_qos.o
QOS_SINK_BIN=qos_comm_sink
QOS_SINK_OBJ=rt_comm_sink.o rt_comm_error.o rt_comm_qos.o

all: rtnet qos
rtnet: CFLAGS+=-I/usr/local/rtnet/include
rtnet: $(RTNET_SOURCE_BIN) $(RTNET_SINK_BIN) $(RTNET_PARPORT_BIN)
qos: $(QOS_SOURCE_BIN) $(QOS_SINK_BIN)
debug: CFLAGS+=-DDEBUG -g
debug: all
static: LDFLAGS+=-static
static: ADDLIBS+=/usr/local/rtai/lib/liblxrt.a /usr/lib/libpthread.a /usr/lib/libpthread_nonshared.a
static: all
        $(STRIP) $(RTNET_PARPORT_BIN) $(RTNET_SOURCE_BIN) $(RTNET_SINK_BIN) $(QOS_SINK_BIN) $(QOS_SINK_BIN)
dist: all
        $(TAR) $(TARFLAGS) $(DIST) $(RTNET_PARPORT_BIN) $(RTNET_SOURCE_BIN) $(RTNET_SINK_BIN) $(QOS_SOURCE_BIN) $(QOS_SINK_BIN)
srcdist:
        -$(TAR) $(TARFLAGS) $(SRCDIST) *.c *.h include/*

%.o: %.c include/parport.h include/rt_comm_backend_ifc.h
        gcc $(CFLAGS) $< -o $@

$(RTNET_PARPORT_BIN): $(RTNET_PARPORT_OBJ)
        gcc $(LDFLAGS) -o $(RTNET_PARPORT_BIN) $(RTNET_PARPORT_OBJ) $(ADDLIBS)

$(RTNET_SOURCE_BIN): $(RTNET_SOURCE_OBJ)
        gcc $(LDFLAGS) -o $(RTNET_SOURCE_BIN) $(RTNET_SOURCE_OBJ) $(ADDLIBS)

$(RTNET_SINK_BIN): $(RTNET_SINK_OBJ)
        gcc $(LDFLAGS) -o $(RTNET_SINK_BIN) $(RTNET_SINK_OBJ) $(ADDLIBS)

$(QOS_SOURCE_BIN): $(QOS_SOURCE_OBJ)
        gcc $(LDFLAGS) -o $(QOS_SOURCE_BIN) $(QOS_SOURCE_OBJ) $(ADDLIBS)

$(QOS_SINK_BIN): $(QOS_SINK_OBJ)
        gcc $(LDFLAGS) -o $(QOS_SINK_BIN) $(QOS_SINK_OBJ) $(ADDLIBS)

.PHONY: clean

clean:
        -rm *.o $(RTNET_PARPORT_BIN) $(RTNET_SOURCE_BIN) $(RTNET_SINK_BIN) $(QOS_SOURCE_BIN) $(QOS_SINK_BIN) $(DIST) $(SRCDIST)
```

Listing A.3: Makefile

A.1.1.2 The program logic

```
#ifndef __PARPORT_H__
#define __PARPORT_H__ 1

/**     \brief  Parallel Port Debugging
```

```
 *      \author Rainer Poisel (tm031051)
 *      \date   March 23, 2007
 *
 *      Macros for displaying events
 *      on the parallel port of a PC
 */

#include <rtai_lxrt.h> /* iopl() */
#include <asm/io.h>
#define USE_PARPORT 1 /* set to 0 for not usint the parport */
#define PARPORT_BASE 0x378 /* /dev/printers/0 */
#define NUM_PINS 8
#define PIN_OFFSET 1 /* Pin 1 to 8 */
#define PRIVILEGE_LEVEL 3 /* Ring 3 */

/** \def INIT_PARPORT()
 *   Initializes parallel port for user-space usage
 */
#if USE_PARPORT == 1
#define INIT_PARPORT() \
        static unsigned lVal = 0; \
                                iopl(PRIVILEGE_LEVEL); \
                                outb(0, PARPORT_BASE + 2);
#else
#define INIT_PARPORT()
#endif

/** \def TOGGLE_PIN(pin)
 *   Toggles a parallel data-port pin
 */
#if USE_PARPORT == 1
#define TOGGLE_PIN(pin) \
                                lVal = inb(PARPORT_BASE); \
                                lVal ^= (1 << (pin % NUM_PINS)); \
                                outb(lVal, PARPORT_BASE);
#else
#define TOGGLE_PIN(pin)
#endif

#endif
```

Listing A.4: parport.h

```
/**     \brief  RT Communication source
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      The source for real-time UDP communication
 *      Invocation example: ./rt_comm_source --server-port 1234 --server-ip 10.0.0.1 --message
      PIN1 --toggle-pin 2 --num-messages 20 --num-failure-messages 10 --tx-delay 5000000
 */

/* standard C-API */
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <string.h>

/* Linux specific */
#include <sched.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>

/* UDP specifics */
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>

/* RTAI and RTnet specifics */
#include <rtai_lxrt.h>

/* own includes */
#include "rt_comm_backend_ifc.h"
#include "rt_comm_error.h"
#include "parport.h"

/* ------- Defines ------- */
#define MAX_MSG_SIZE 512
#define DEFAULT_STATIC_STRING_LENGTH 255
#define RT_TASK_ID 292
#define DEFAULT_MESSAGE "PIN1"
#define DEFAULT_TOGGLE_PIN 1
#define DEFAULT_PORT 1234
#define DEFAULT_NUM_RT_MESSAGES 10000L
#define DEFAULT_NUM_SEND_FAILURES 1000000L
#define DEFAULT_TX_DELAY 50000000 /* 5000000ns = 5ms */
```

```

/* ------ typedefs ------ */

/* ------ Prototypes ------ */

/**     \fn      void usage(char const* pProgramName);
 *      \brief   print how to use this program
 *      \param   pProgramName the name of this binary
 *      \return  nothing
 */
static void usage(char const* pProgramName);

/* ------ Implementation ------ */
int main(int pArgc,char* pArgv[])
{
        RT_TASK* lClientTask = NULL;
        RTIME lDelay = DEFAULT_TX_DELAY;
        char lProgramName[DEFAULT_STATIC_STRING_LENGTH];
        int lSocketFD = 0;
        int lReturn = 0;
        long long lMaxCnt = DEFAULT_NUM_RT_MESSAGES, lCnt = 0,lMaxFailCnt =
              DEFAULT_NUM_SEND_FAILURES,
                        lFailCnt = 0;
        static struct sockaddr_in lLocalAddr,lRemoteAddr;
        char lMessage[DEFAULT_STATIC_STRING_LENGTH];
        short lParportPin = DEFAULT_TOGGLE_PIN;
        struct backend_desc lBackend;

        /* command-line parsing variables */
        int lOptionIndex = 0;
        struct option lLongOptions[] =
        {
                {"local-ip",required_argument,0,'i'},
                {"local-port",required_argument,0,'p'},
                {"server-ip",required_argument,0,'j'},
                {"server-port",required_argument,0,'q'},
                {"message",required_argument,0,'m'},
                {"toggle-pin",required_argument,0,'t'},
                {"num-messages",required_argument,0,'n'},
                {"num-failure-messages",required_argument,0,'f'},
                {"tx-delay",required_argument,0,'d'},
                {"help",no_argument,0,'h'},
                {0,0,0,0}
        };

        /* initialize parallel port debugging */
        INIT_PARPORT()

        /* initialize variables with functions if necessary */
        init_backend(&lBackend);
        memset(&lLocalAddr,0,sizeof(struct sockaddr_in));
        lLocalAddr.sin_family = AF_INET;
        lLocalAddr.sin_addr.s_addr = INADDR_ANY;
        /* local port is chosen by the operating system (linux) */
        memset(&lRemoteAddr,0,sizeof(struct sockaddr_in));
        lRemoteAddr.sin_family = AF_INET;
        lRemoteAddr.sin_addr.s_addr = INADDR_ANY;
        lRemoteAddr.sin_port = htons(DEFAULT_PORT);

        strncpy(lProgramName,pArgv[0],DEFAULT_STATIC_STRING_LENGTH);
        strncpy(lMessage,DEFAULT_MESSAGE,DEFAULT_STATIC_STRING_LENGTH);

        /* doing the command-line parsing */
        while(1)
        {
                lReturn = getopt_long(pArgc,pArgv,"i:p:j:q:m:t:f:n:d:h",lLongOptions,&lOptionIndex
                        );
                if(lReturn == -1) /* all options parsed */
                        break;

                switch(lReturn)
                {
                        case 'p':
                                /* port to bind to */
                                lLocalAddr.sin_port = htons(strtol(optarg,NULL,10));
                                break;
                        case 'q':
                                /* server port */
                                lRemoteAddr.sin_port = htons(strtol(optarg,NULL,10));
                                break;
                        case 'i':
                                /* ip address to bind to */
                                lLocalAddr.sin_addr.s_addr = inet_addr(optarg);
                                break;
                        case 'j':
                                /* server ip address */
                                lRemoteAddr.sin_addr.s_addr = inet_addr(optarg);
                                break;
                        case 'm':
                                /* message to send */
```

```
                    strncpy(lMessage, optarg, DEFAULT_STATIC_STRING_LENGTH);
                    break;
                case 't':
                    /* parallel data-port to toggle */
                    lParportPin = strtol(optarg,NULL,10);
                    break;
                case 'n':
                    /* max number of messages to transmit */
                    lMaxCnt = strtol(optarg,NULL,10);
                    break;
                case 'f':
                    /* maximum transmission errors */
                    lMaxFailCnt = strtol(optarg,NULL,10);
                    break;
                case 'd':
                    /* delay between transmission of packets */
                    lDelay = strtol(optarg,NULL,10);
                    break;
                case 'h':
                    /* help string */
                    usage(lProgramName);
                    return EXIT_SUCCESS;
                    break;
                case '?':
                    /* unknown argument; error message is displayed by getopt */
                    return EXIT_FAILURE;
                default:
                    abort();
            }
        }

        /* allocate memory only from RAM, no paging */
        mlockall(MCL_CURRENT|MCL_FUTURE);

        /* create the socket */
        lSocketFD = lBackend.socket(AF_INET,SOCK_DGRAM,0);
        if(lSocketFD < 0)
        {
            fprintf(stderr,"Error opening the socket.\n");
            return EXIT_SUCCESS;
        }

        /* priority = 1; stack_size = 0; max_msg_size = 0 */
        lClientTask = rt_task_init((RT_TASK_ID),1,0,0);
        if(lClientTask == NULL)
            bail("Error during initialization of the master task.\n");

        /* switch into hard real-time mode */
        lBackend.request_hard_realtime();

        /* bind the socket to a local address */
        lReturn = lBackend.bind(lSocketFD,(struct sockaddr *)&lLocalAddr,sizeof(struct sockaddr_in
            ));
        if(lReturn < 0)
            bail("rt_dev_bind() returned an error.\n");

        /* rt_dev_connect() specifies the destination address for the socket */
        lBackend.connect(lSocketFD,(struct sockaddr*) &lRemoteAddr,sizeof(struct sockaddr_in));

        while(1)
        {
            /* send the message */
            if(lBackend.send(lSocketFD,lMessage,strnlen(lMessage,DEFAULT_STATIC_STRING_LENGTH)
                +1,0) > 0)
            {
                TOGGLE_PIN(lParportPin - 1)

                if(lCnt < (lMaxCnt - 1))
                {
                    ++lCnt;
                }
                else
                {
                    /* get out of this loop since there is no serial communication
                     * possible in hard real-time mode
                     */
                    break;
                }
            }
            else
            {
                if(lFailCnt < (lMaxFailCnt - 1))
                {
                    ++lFailCnt;
                }
                else
                {
                    break;
                }
            }
```

```
                rt_busy_sleep(nano2count(lDelay)); /* pass control to the RTAI scheduler */
        }

        /* switch to soft real-time */
        lBackend.release_hard_realtime();

        /* close socket */
        lBackend.close(lSocketFD);

        /* delete the task */
        rt_task_delete(lClientTask);

        fprintf(stdout,"\nStatistics:\n"
                       "==========\n");
        fprintf(stdout,  "Successfully transmitted messages: %lld\n"
                         "Failures:                          %lld\n",
                                                                        lCnt+1,lFailCnt);

        return EXIT_SUCCESS;
}

static void usage(char const* pProgramName)
{
        fprintf(stdout,"%s [--local-ip ip] [--local-port port] [--server-port port]\n"
                "[--server-ip ip] [--message message] [--toggle-pin pin]\n"
                "          [--num-messages num] [--num-failure-messages num]\n"
                "          [--tx-delay delay]\n\n"
                "    port    ... port to send messages or to bind (locally) to\n"
                "    ip      ... ip-address to send messages or to bind (locally) to\n"
                "    message ... message to send\n"
                "    num     ... number of messages\n"
                "    pin     ... parallel data-port pin to toggle on message transmission\n"
                "    delay   ... delay between transmission of packets\n\n",
                pProgramName);
}
```

Listing A.5: rt_comm_source.c

```
/**     \brief  RT Communication sink
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      The sink (server) for real-time UDP communication
 *      Invocation example: ./rt_comm_sink --port 1234 --quit-message 42 --message1 PIN1
 */

/* standard C-API */
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <string.h>

/* Linux specific */
#include <sched.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>

/* UDP specifics */
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>

/* RTAI specifics */
#include <rtai_lxrt.h>

/* own includes */
#include "rt_comm_backend_ifc.h"
#include "rt_comm_error.h"
#include "parport.h"

/* ------ Defines ------ */
#define MAX_MSG_SIZE 8192
#define DEFAULT_PORT 1234
#define DEFAULT_STATIC_STRING_LENGTH 255
#define RT_TASK_ID 292 + 1
#define QUIT_STRING "You have no chance to survive make your time."
#define PIN_MESSAGE "PIN"

/* ------ typedefs ------ */

/* ------ Prototypes ------ */

/**     \fn     void usage(char const* pProgramName);
 *      \brief  print how to use this program
 *      \param  pProgramName the name of this binary
 *      \return nothing
```

```c
 */
static void usage(char const* pProgramName);

/* ------ Implementation ------ */
int main(int pArgc,char* pArgv[])
{
        RT_TASK* lServerTask = NULL;
        char lMsg[MAX_MSG_SIZE];
        char lProgramName[DEFAULT_STATIC_STRING_LENGTH],
                     lQuitMessage[DEFAULT_STATIC_STRING_LENGTH];
        int lSocketFD = 0;
        int lReturn = 0,lCnt = 0;
        static struct sockaddr_in lLocalAddr;
        long long lNumClientMessages[NUM_PINS]; /* number of received messages */
        char lPinMessages[NUM_PINS][DEFAULT_STATIC_STRING_LENGTH];
        struct backend_desc lBackend;

        /* command-line parsing variables */
        int lOptionIndex = 0;
        struct option lLongOptions[] =
        {
                {"port",required_argument,0,'p'},
                {"quit-message",required_argument,0,'q'},
                {"help",no_argument,0,'h'},
                {0,0,0,0}
        };

        /* initialize parallel port debugging */
        INIT_PARPORT()

        for(lCnt = 0; lCnt < NUM_PINS; ++lCnt)
        {
                lNumClientMessages[lCnt] = 0;
                snprintf(lPinMessages[lCnt],DEFAULT_STATIC_STRING_LENGTH,"%s%d",
                                PIN_MESSAGE,lCnt + PIN_OFFSET);
        }

        /* initialize variables with functions if necessary */
        init_backend(&lBackend); /* init function pointers of the backend */
        memset(&lLocalAddr,0,sizeof(struct sockaddr_in));
        lLocalAddr.sin_family = AF_INET;
        lLocalAddr.sin_addr.s_addr = INADDR_ANY;
        lLocalAddr.sin_port = htons(DEFAULT_PORT);

        strcpy(lProgramName,pArgv[0]);
        strncpy(lQuitMessage,QUIT_STRING,DEFAULT_STATIC_STRING_LENGTH);

        /* doing the command-line parsing */
        while(1)
        {
                lReturn = getopt_long(pArgc,pArgv,"p:q:h",lLongOptions,&lOptionIndex);
                if(lReturn == -1) /* all options parsed */
                        break;

                switch(lReturn)
                {
                        case 'p':
                                /* port to listen on */
                                lLocalAddr.sin_port = htons(strtol(optarg,NULL,10));
                                break;
                        case 'q':
                                strncpy(lQuitMessage,optarg,DEFAULT_STATIC_STRING_LENGTH);
                                /* message to set pin2 of the parallel data-port */
                                break;
                        case 'h':
                                /* display help */
                                usage(lProgramName);
                                return EXIT_SUCCESS;
                        case '?':
                                /* unknown argument; error message is displayed by getopt */
                                return EXIT_FAILURE;
                        default:
                                abort();
                }
        }

        /* allocate memory only from RAM, no paging */
        mlockall(MCL_CURRENT|MCL_FUTURE);

        /* create the socket */
        lSocketFD = lBackend.socket(AF_INET,SOCK_DGRAM,0);
        if(lSocketFD < 0)
        {
                fprintf(stderr,"Error opening the socket.\n");
                return EXIT_SUCCESS;
        }

        /* priority = 1; stack_size = 0; max_msg_size = 0 */
        lServerTask = rt_task_init((RT_TASK_ID),1,0,0);
        if(lServerTask == NULL)
```

```
                bail("Error during initialization of the master task.\n");

        fprintf(stdout,"Starting hard real-time receive process. All your base are belong to us!\n");
        fflush(stdout);

        /* switch into hard real-time mode */
        lBackend.request_hard_realtime();

        /* bind the socket to a local address */
        lReturn = lBackend.bind(lSocketFD,(struct sockaddr *)&lLocalAddr,sizeof(struct sockaddr_in));
        if(lReturn < 0)
                bail("bind() returned an error.\n");

        while(1)
        {
                /* block until a packet is received */
                lReturn = lBackend.recv(lSocketFD,lMsg,sizeof(lMsg),0);

                /* quit on error */
                if(lReturn < 0)
                {
                        break;
                }
                /* quit if requested */
                else if((strncmp(lMsg,lQuitMessage,DEFAULT_STATIC_STRING_LENGTH) == 0) ||
                                (strncmp(lMsg,"42",DEFAULT_STATIC_STRING_LENGTH) == 0)) /* emergency exit feature */
                {
                        break;
                }
                /* set pins if requested */
                for(lCnt = 0; lCnt < NUM_PINS; ++lCnt)
                {
                        if(strncmp(lMsg,lPinMessages[lCnt],DEFAULT_STATIC_STRING_LENGTH) == 0)
                        {
                                ++lNumClientMessages[lCnt];
                                TOGGLE_PIN(lCnt);
                                break;
                        }
                }
                /* ignore all other messages */
        }

        /* switch to soft real-time mode to be able to place system-calls */
        lBackend.release_hard_realtime();

        /* close the socket */
        lBackend.close(lSocketFD);

        /* quit the real-time task */
        rt_task_delete(lServerTask);

        fprintf(stdout,"Quit message received, shutting down therefore. For great justice.\n");

        fprintf(stdout,"\nStatistics:\n"
                                                                                "===========\n");
        for(lCnt = 0; lCnt < NUM_PINS; ++lCnt)
        {
                fprintf(stdout,         "%s: %lld\n",lPinMessages[lCnt],lNumClientMessages[lCnt]);
        }
        fprintf(stdout,"1 Quit message received.\n\n");

        return EXIT_SUCCESS;
}

static void usage(char const* pProgramName)
{
        fprintf(stdout,"%s [--port port] [--quit-message message]\n\n"
                   "    port          ... port to listen on\n"
                   "    quit-message  ... message that causes the program to quit\n\n",
                   pProgramName);
}
```

Listing A.6: rt_comm_sink.c

A.1.1.3 The backends and their interface

```
#ifndef __RT_COMM_BACKEND_IFC_H__
#define __RT_COMM_BACKEND_IFC_H__ 1

/**     \brief  RT Communication Interfaces
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      Interfaces for communication backends
 */

/* include files for data types */
#include <sys/types.h>

/**     \struct backend_structure
 *                      \brief  contains function pointers to the backend-functions
 */
struct backend_desc
{
        /* sink part */
        ssize_t (*recv)(int pSockFD,void* pBuf,size_t pLen,int pFlags);

        /* source part */
        int (*connect)(int pSockFD,const struct sockaddr* pServerAddr,socklen_t pAddrLen);
        ssize_t (*send)(int pSockFD,const void* pBuf,size_t pLen,int pFlags);

        /* common part */
        int (*socket)(int pDomain,int pType,int pProtocol);
        int (*close)(int pSockFD);
        int (*bind)(int pSockFD,const struct sockaddr* pMyAddr,socklen_t pAddrLen);
        void (*request_hard_realtime)(void);
        void (*release_hard_realtime)(void);
};

/**     \fn     int init_backend();
 *      \brief  initializes the backend
 *      \param  pBackend pointer to a backend structure
 *      \return 0 on success; -1 in case of an error
 */
extern int init_backend(struct backend_desc* pBackend);

#endif
```

Listing A.7: rt_comm_backend_ifc.h

```
/**     \brief  RT Communication RTnet module
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      The RTnet Backend
 */

#include <stdlib.h>
#include <stdio.h>

/* backend specifics */
#include <rtnet.h>

/* own include files */
#include "rt_comm_backend_ifc.h"

/* ------ Defines ------ */

/* ------ Prototypes ------ */
static void make_hard_realtime(void);
static void make_soft_realtime(void);
static int rt_socket(int pDomain,int pType,int pProtocol);
static int rt_close(int pSockFD);
static int rt_bind(int pSockFD,const struct sockaddr* pMyAddr,socklen_t pAddrLen);
static ssize_t rt_recv_func(int pSockFD,void* pBuf,size_t pLen,int pFlags);
static ssize_t rt_send_func(int pSockFD,const void* pBuf,size_t pLen,int pFlags);
static int rt_connect(int pSockFD,const struct sockaddr* pServerAddr,socklen_t pAddrLen);

/* ------ typedefs ------ */

/* ------ Implementation ------ */
int init_backend(struct backend_desc* pBackend)
{
        /* assign function pointers */
        pBackend->request_hard_realtime = &make_hard_realtime;
        pBackend->release_hard_realtime = &make_soft_realtime;
        pBackend->socket = &rt_socket;
        pBackend->close = &rt_close;
        pBackend->bind = &rt_bind;
        pBackend->recv = &rt_recv_func;
        pBackend->connect = &rt_connect;
        pBackend->send = &rt_send_func;

        return 0; /* success */
}

static void make_hard_realtime(void)
{
        rt_make_hard_real_time();
}
```

```

static void make_soft_realtime(void)
{
        rt_make_soft_real_time();
}

static int rt_socket(int pDomain,int pType,int pProtocol)
{
        return rt_dev_socket(pDomain,pType,pProtocol);
}

static int rt_close(int pSockFD)
{
        return rt_dev_close(pSockFD);
}

static int rt_bind(int pSockFD,const struct sockaddr* pMyAddr,socklen_t pAddrLen)
{
        return rt_dev_bind(pSockFD,pMyAddr,pAddrLen);
}

static ssize_t rt_recv_func(int pSockFD,void* pBuf,size_t pLen,int pFlags)
{
        return rt_dev_recv(pSockFD,pBuf,pLen,pFlags);
}

static int rt_connect(int pSockFD,const struct sockaddr* pServerAddr,socklen_t pAddrLen)
{
        return rt_dev_connect(pSockFD,pServerAddr,pAddrLen);
}

static ssize_t rt_send_func(int pSockFD,const void* pBuf,size_t pLen,int pFlags)
{
        return rt_dev_send(pSockFD,pBuf,pLen,pFlags);
}
```

Listing A.8: rt_comm_rtnet.c

```
/**     \brief  RT Communication QoS module
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      The QoS Backend
 */

#include <stdlib.h>
#include <stdio.h>

/* backend specifics */
#include <sys/socket.h>
#include <unistd.h>

/* own include files */
#include "rt_comm_backend_ifc.h"

/* ------ Defines ------ */

/* ------ Prototypes ------ */
static void make_hard_realtime(void);
static void make_soft_realtime(void);
static int qos_socket(int pDomain,int pType,int pProtocol);
static int qos_close(int pSockFD);
static int qos_bind(int pSockFD,const struct sockaddr* pMyAddr,socklen_t pAddrLen);
static ssize_t qos_recv(int pSockFD,void* pBuf,size_t pLen,int pFlags);
static ssize_t qos_send(int pSockFD,const void* pBuf,size_t pLen,int pFlags);
static int qos_connect(int pSockFD,const struct sockaddr* pServerAddr,socklen_t pAddrLen);

/* ------ typedefs ------ */

/* ------ Implementation ------ */
int init_backend(struct backend_desc* pBackend)
{
        /* assign function pointers */
        pBackend->request_hard_realtime = &make_hard_realtime;
        pBackend->release_hard_realtime = &make_soft_realtime;
        pBackend->socket = &qos_socket;
        pBackend->close = &qos_close;
        pBackend->bind = &qos_bind;
        pBackend->recv = &qos_recv;
        pBackend->connect = &qos_connect;
        pBackend->send = &qos_send;

        return 0; /* success */
}

static void make_hard_realtime(void)
{
        /* nothing to do here because normal system calls are used
```

```
         * this is not allowed with RTAI in general
         */

}

static void make_soft_realtime(void)
{
        /* see make_hard_realtime for more information */
}

static int qos_socket(int pDomain,int pType,int pProtocol)
{
        return socket(pDomain,pType,pProtocol);
}

static int qos_close(int pSockFD)
{
        return close(pSockFD);
}

static int qos_bind(int pSockFD,const struct sockaddr* pMyAddr,socklen_t pAddrLen)
{
        return bind(pSockFD,pMyAddr,pAddrLen);
}

static ssize_t qos_recv(int pSockFD,void* pBuf,size_t pLen,int pFlags)
{
        return recv(pSockFD,pBuf,pLen,pFlags);
}

static int qos_connect(int pSockFD,const struct sockaddr* pServerAddr,socklen_t pAddrLen)
{
        return connect(pSockFD,pServerAddr,pAddrLen);
}

static ssize_t qos_send(int pSockFD,const void* pBuf,size_t pLen,int pFlags)
{
        return send(pSockFD,pBuf,pLen,pFlags);
}
```

Listing A.9: rt_comm_qos.c

A.1.1.4 Error management

```
#ifndef __RT_COMM_ERROR_H__
#define __RT_COMM_ERROR_H__ 1

/**     \brief  RT Communication Error Module
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      Functions regarding error management
 */

/**     \fn     void bail(char* pBailString);
 *      \brief  bail and exit process
 *      \param  pBailString string to bail
 *      \return nothing
 */
extern void bail(char* pBailString);

#endif
```

Listing A.10: rt_comm_error.h

```
/**     \brief  RT Communication Error Module
 *      \author Rainer Poisel (tm031051)
 *      \date   March 31, 2007
 *
 *      RT UDP Error module
 */

#include <stdlib.h>
#include <stdio.h>

#include "rt_comm_error.h"

/* ——— Defines ——— */

/* ——— Prototypes ——— */

/* ——— typedefs ——— */

/* ------- Implementation ------- */
void bail(char* pBailString)
{
        fprintf(stderr, pBailString);
        fprintf(stderr, "Will exit now.\n");
        exit(EXIT_FAILURE);
}
```

Listing A.11: rt_comm_error.c

A.2 Configuration files

A.2.1 Switch configuration

```
!
version 12.0
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Switch
!
wrr-queue bandwidth 255 255 255 255
wrr-queue cos-map 1 0 1 7
wrr-queue cos-map 2 2 3
wrr-queue cos-map 3 4 5
wrr-queue cos-map 4 6
!
ip subnet-zero
!
interface FastEthernet0/1
 switchport access vlan 20
 switchport priority extend cos 7
 spanning-tree portfast
!
interface FastEthernet0/2
 switchport access vlan 20
 switchport priority extend cos 3
 spanning-tree portfast
!
interface FastEthernet0/3
 switchport access vlan 20
 switchport mode trunk
 spanning-tree portfast
!
interface FastEthernet0/4
 switchport access vlan 20
 spanning-tree portfast
!
interface VLAN20
 no ip directed-broadcast
 no ip route-cache
 shutdown
!
!
line con 0
 transport input none
 stopbits 1
line vty 5 15
!
end
```

Listing A.12: catalyst_2950.conf

A.2.2 RTnet configuration

```
#!/bin/sh

prefix="/usr/local/rtnet"
exec_prefix="${prefix}"
RTNET_MOD="${exec_prefix}/modules"
RTIFCONFIG="${exec_prefix}/sbin/rtifconfig"
RTCFG="${exec_prefix}/sbin/rtcfg"
TDMACFG="${exec_prefix}/sbin/tdmacfg"

MODULE_EXT=".ko"
RT_DRIVER="rt_8139too"
RT_DRIVER_OPTIONS="cards=0,1,0,0"
IPADDR="10.0.0.1"
NETMASK=""
RT_LOOPBACK="yes"
RTCAP="no"
STAGE_2_SRC=""
STAGE_2_DST=""
STAGE_2_CMDS=""

TDMA_MODE="master"
TDMA_SLAVES="10.0.0.10 10.0.0.11"
TDMA_CYCLE="3000"
TDMA_OFFSET="200"
```

Listing A.13: The master's rtnet.conf

```
#!/bin/sh

prefix="/usr/local/rtnet-rtai-3.5-0.9.8"
exec_prefix="${prefix}"
RTNET_MOD="${exec_prefix}/modules"
RTIFCONFIG="${exec_prefix}/sbin/rtifconfig"
RTCFG="${exec_prefix}/sbin/rtcfg"
TDMACFG="${exec_prefix}/sbin/tdmacfg"

MODULE_EXT=".ko"
RT_DRIVER="rt_8139too"
RT_DRIVER_OPTIONS="cards=1,0,0,0"
IPADDR="10.0.0.10"
NETMASK="255.0.0.0"
RT_LOOPBACK="yes"
RTCAP="no"
STAGE_2_SRC=""
STAGE_2_DST=""
STAGE_2_CMDS=""

TDMA_MODE="slave"
TDMA_SLAVES="10.0.0.10 10.0.0.11"
TDMA_CYCLE="3000"
TDMA_OFFSET="200"
```

Listing A.14: The slave's rtnet.conf

Glossary

ADEOS Adaptive Domain Environment for Operating Systems - “The Adaptive Domain Environment for Operatign Systems (Adeos) was designed to offer the capability of sharing the hardware between multiple operating systems. This can in turn be used to create a real-time domain which has priority on all other domains.” [Yag01, p. 1]

API “An application programming interface (API) is a source code interface that a computer system or program library provides to support requests for services to be made of it by a computer program.” [Wik07a, cmp.]

ARP With the IP protocol it is only possible to address nodes on layer 3 of the ISO/OSI network model. Layer 3 is not aware of the hardware-address of participating nodes. It is therefore not possible to communicate on basis of physical addresses. The Address Resolution Protocol (ARP) has been developed for this purpose. It converts logical IP-addresses into network specific hardware-addresses [Hei02, cmp. p. 228].

ASIC “An application-specific integrated circuit (ASIC) is an integrated circuit (IC) customised for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. In contrast, the 7400 series and 4000 series integrated circuits are logic building blocks that can be wired together for use in many different applications.” [Wik07b]

ASN.1 Abstract Syntax Notation 1 - Is the encoding scheme for Management-Information-Base-Variables which have been defined by the ISO. It is used to describe complex data structures in a manufacturer independent way. [Hei02, cmp. p. 744-745]

burst “Operation of a data network in which data transmission is interrupted at intervals.” [Wik07c]

busy waiting Permanently requesting if a specific status has been reached. Polling as a field of application requests communication participants to transmit their data in a cyclic fashion. [WB05, cmp. p. 263]

CAN Controller Area Network - A fieldbus system which has mainly been developed for interconnection of devices in cars [WB05, cmp. p. 278].

CAP Computer Aided Planning

collision domain “A single, half duplex mode CSMA/CD network. If two of more Media Access Control (MAC) sublayers are within the sme collision domain and both transmit at the same time, a collision will occur. MAC sublayers separated by a repeater are in the same collision domain. MAC sublayers separated by a bridge are within different collision domains” [oEI05, p. 22]

context switching “Context Switching is the process of switching from one thread of execution to another. This involves saving the state of the CPU’s registers and loading a new state, flushing caches, and changing the current virtual memory map. Context switches on most architectures are a relatively expensive operation and as such they are avoided as much as possible.” [Jos05, p. 9]

CoS “Class of Service is a method which classifies traffic into specific classes which are offered different grade of service through the network. ”[Jas02, p. 16]

EMC “Electromagnetic Compatibility is the branch of electrical sciences which studies the unintentional generation, propagation and reception of electromagnetic energy with reference to the unwanted effects that such an energy may induce. To this purpose, the goal of EMC is the correct operation, in the same electromagnetic environment, of different equipment which involve electromagnetic phenomena in their operation.” [Wik07e]

FCFS First Come First Served - Elements of this data structure get processed in the order they arrive. [Jas02, cmp. p. 89]

FIFO Abbreviation for a First In First Out queue. Elements that are first stored in this data structure are first fetched out of the queue. This can be seen in contrast to a stack where elements that got first stored in the queue are last fetched out of the queue.

FTP File Transfer Protocol - allowes file exchange between participants regardless of architecture and used operating system. [Hei02, cmp. p. 339]

HAL The Hardware abstraction layer is used to represent different hardware in a similar fashion to the operating system which controls it. Thus it is easy to adapt operating systems which support a HAL to different hardware. [Tie03, cmp. p. 77]

HTTP The HyperText Transport Protocol is mainly used for exchange of hypertext-documents between WWW-clients and WWW-servers. [Hei02, cmp. p. 667]

I/O Input/Output

IAK The processor responds with Interrupt Acknowledge (IAK) upon an Interrupt Request (IRQ) to display the device to place its interrupt vector number on the data bus. The processor then effectively simulates an INT instruction using the supplied vector index. [Abb06, cmp. p. 144]

IEC "The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes international standards for all electrical, electronic and related technologies. These serve as a basis for national standardization and as references when drafting international tenders and contracts. Through its members, the IEC promotes international cooperation on all questions of electrotechnical standardization and related matters, such as the assessment of conformity to standards, in the fields of electricity, electronics and related technologies." [IEC07]

IEEE "The Institute of Electrical and Electronics Engineers or IEEE (pronounced as eye-triple-e) is an international non-profit, professional organization for the advancement of technology related to electricity. It has the most members of any technical professional organization in the world, with more than 360,000 members in around 175 countries." [Wik07h]

inbound signalling The opposite of "outbound signalling". Signalling data and data itself are transmitted in the same communication channels.

IRQ "A device requiring service asserts an Interrupt Request (IRQ) line." [Abb06, p. 144] Therefore this signal displays an event to the processor.

ISO "The International Standards Organization specifies requirements for products, services, processes, materials and systems. ISO standards are designed to be implemented worldwide." [Sec06, cmp.]

ISR "The instruction stream that services the event (an interrupt) is called an Interrupt Service Routine." [Abb06, p.141]

MBR "A Master Boot Record (MBR), or partition sector, is the 512-byte boot sector that is the first sector (Sector 0) of a partitioned data storage device such as a hard disk ... It is sometimes used for bootstrapping operating systems, containing a machine code program; sometimes used for holding part of a disk's partition table2; and sometimes used for uniquely identifying individual disk media, with a 32-bit data signature; although on some machines it is entirely unused." [Wik07k]

multiplexer Multiplexer allow the transmission of more than one signal over one communication line at the same time. There exist different approaches: frequency, time, code and wavelength division multiplex. The main reason for multiplexing is to reduce the number of required communication lines. [Loc02, cmp. p. 220]

OSI The Open Systems Interconnection model is built of 7 layers. The functionality of each layer is determined in a unique manufacturer independent communication model. Each layer provides specific functionality for its upper layer functionality. Communication between layers happens over so called "primitives". [Hei02, cmp. p. 20]

outbound signalling Signalling is performed outside the actual communication stream. Applied to telephony that means that signalling data is transmitted outside the frequency band (300Hz ... 3400Hz) of speech. [EH92, cmp. p. 388]

PCI	The Peripheral Component Interconnect is implemented as a 32 or 64Bit broad address-/databus. There can be multiple masters with central arbitration. Usually PCI is used in Intel based PCs, Power-PCs and Alpha workstations by DEC. [Dem00, cmp. p. 560-561]
PLC	"Usually a Programmable Logic Controller is used to realize a discrete control unit. It requests the state of inputs in a cyclic fashion associates them and sets outputs accordingly to the program logic." [WB05, p. 8]
PPC	Production Planning and Production Control
$process$	"A program during its state of execution" [Her04, p. 21]. If a program gets executed, its program code gets loaded into the computers main memory and started. The running program is then called process.
$Q-Q\ plot$	"A Quantile-Quantile plot is a scatter plot comparing the fitted and empirical distributions in terms of the dimensional values of the variable (i.e., empirical quantiles). It is a graphical technique for determining if a data set come from a known population. In this plot on the y-axis we have empirical quantiles and on the x-axis we have the ones got by the theoretical model." [Ric05, p. 4]
QoS	"Quality of Service describes a guaranteed service, which is offered by a network to an application, based on a contract. Quantitative characteristics (QoS-parameters) which describe the grade of service are negotiated between the network and the application. "[Jas02, p. 13]
RFC	"In internetworking and computer network engineering, Request for Comments (RFC) documents are a series of memoranda encompassing new research, innovations, and methodologies applicable to Internet technologies." [Wik07n]
$round\ trip\ time$	"The interval between the sending of a packet and the reciept of its acknowledgement." [KP91, cmp. p. 1]
RPC	Remote Procedure Calls are used to distribute computing power to several computers. Further it is used to perform specific action on remote computers. [Ray01, cmp. p. 924]
RX	abbreviation for receive
$SMTP$	The Simple Mail Transfer Protocol is used for exchanging E-Mails between computers. It relies on the TCP as transport protocol and uses the well-known port 25. [Hei02, cmp. p. 471]
TCP	The Transport Control Protocol is used in packetswitching networks for virtual connection-oriented and sequenced data transmission. As such the integrity of transmitted data is assured. It is built as layer 4 protocol upon the internet protocol (IP). [Hei02, cmp. p. 265]

TLB “A Translation Lookaside Buffer (TLB) is a cache in a CPU that is used to improve the speed of virtual address translation. A TLB has a fixed number of entries containing parts of the page table which translate virtual addresses into physical addresses.” [Wik07q]

TOS The Type Of Service field can be used for inbound signalling. It displays the priority of IP packets. [Hei02, cmp. p. 97]

TTP “The Time Triggered Protocol belongs to the class of the time-triggered protocols, where the temporal control signals are solely derived from the progression of time. TTP was originally developed for high-dependability hard real-time applications, where timely error detection and fault-tolerance must be provided.” [H. 98, p. 1]

TX abbreviation for transmit

UDP The User Datagram Protocol is a layer 4 protocol which supports multiple sessions for the transmission of datagramms. Like the TCP protocol it relies on the internet protocols on layer 3 of the ISO/OSI stack. As a best-effort protocol there is no end-to-end control, sequencing and acknowledgement for received datagramms. [Hei02, cmp. p. 689]

vanilla Linux This are the official kernel sources released on http://www.kernel.org/. [Ver07, cmp.]

VLAN “. . . is a group of network devices and services that is not restricted to a physical segment or switch . . . VLANs logically segment switched networks based on an organization’s functions, project teams, or applications rather than on a physical or geographical basis.” [Ron04, p. 304]

VME The “Versa Module Europa” or “VERSAbus-E” is “a scalable backplane bus interface . . . Three main types of cards reside on the bus. The Controller, which supervises bus activity. A Master which Reads/Writes data to a Slave board, and a Slave interface which simply allows data to be accessed via a Read or Write from a Master.” [Dav07, cmp.]

VOIP Voice over IP is used to convey telephone calls over packet switched networks. Classical data networks had to be adapted for the transmission of speech data. Digital data is compressed and requires therfore fewer bandwidth than analog telephony (factor 10). [Hei02, cmp. p. 689]

Bibliography

[Abb06] Doug Abbott, *Linux for Embedded and Real-time Applications*, 2006.

[AKRS94] C. Aras, J. Kurose, D. Reeves, and H. Schulzrinne, *Real-time communication in packet-switched networks*, 1994.

[B. 04] B. Chapman and R. Graziani and E. Horn and A. Johnson and A. Large and T. Rufi and J. DeLeon and R. Duffy and J. Lorenz and A. Tucker, *CCNA 1 and 2 Companion Guide*, Cisco Press, 2004.

[Bei07] I. Beijnum, *Traffic Engineering: Queuing, Traffic Shaping, and Policing*, 2007.

[BS07] Byte and Switch, *Metcalfe: FC 'Beginning to Smell'*, `http://www.byteandswitch.com/document.asp?doc_id=34327`, 2007.

[cap06] captain.at, *Parallel port frequency test of RTAI - Hard real time test*, `http://www.captain.at/programming/rtai/test.php`, 2006.

[Cis05] Cisco Systems, Inc., *Configuring EtherChannel and 802.1Q Trunking Between Catalyst L2 Fixed Configuration Switches and a Router (InterVLAN Routing)*, `http://www.cisco.com/warp/public/473/158.pdf`, 2005.

[Cis07a] Cisco Systems, Inc, *Catalyst 2950 and Catalyst 2955 Switch Software Configuration Guide, 12.1(22)EA2 - Configuring QoS*, `http://www.cisco.com/en/US/products/hw/switches/ps628/products_configuration_guide_chapter09186a00802c31e8.html#wp1094275`, 2007.

[Cis07b] ______, *Configuring VLANs*, `http://www.cisco.com/en/US/products/hw/switches/ps628/products_configuration_guide_chapter09186a00802c305f.html`, 2007.

[Cis07c] Cisco Systems, Inc., *Configuring Voice VLAN*, `http://www.cisco.com/univercd/cc/td/doc/product/lan/cat2970/12220se/2970scg/swvoip.pdf`, 2007.

[CS06] A. Lüder C. Schwab, *Automatisierungsprotokolle*, Industrial Ethernet (Ethernet-basierte Automatisierungsprotokolle), April 2006.

[cTA06] EPSG Office c/o TEMA AG, *Ethernet powerlink one step ahead*, EPSG, 2006.

[D. 05] D. Stahr, *Example 5: Rate shaping*, http://ebtables.sourceforge.net/examples/example5.html, 2005.

[Dav03] David A. Wheeler, *Program Library HOWTO*, http://tldp.org/HOWTO/Program-Library-HOWTO/index.html, 2003.

[Dav07] Leroy Davis, *VME Bus*, http://www.interfacebus.com/Design_Connector_VME.html#a, 2007.

[Deb06] Debian FAQ Authors, *The Debian GNU/Linux FAQ - Chapter 10*, http://www.debian.org/doc/manuals/debian-faq/ch-customizing.en.html#s-booting, 2006.

[Dem00] Klaus Dembowski, *PC Hardware Referenz*, 2000.

[EH92] Wolfgang Lörcher Eberhard Herter, *Nachrichtentechnik*, 1992.

[e.V06] PROFIBUS Nutzerorganisation e.V., *Profinet systembeschreibung*, PROFINET, 2006.

[Fab07] Fabrice Bellard, *QEMU Accelerator User Documentation*, http://fabrice.bellard.free.fr/qemu/kqemu-doc.html, 2007.

[Fai01] Gorry Fairhurst, *Ethernet Bridges and Switches*, http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/bridge.html, 2001.

[Fis04] Bernhard Fischer, *Netzwerktechnik*, 2004.

[Flo05] Florida Center for Instructional Technology College of Education, *Topology*, http://fcit.usf.edu/network/chap5/chap5.htm, 2005.

[Fre95] Free Software Foundation, *Linux Programmer's Manual*, 1995.

[Gra03] Steve Graegert, *Das Ethernet Tutorial*, http://eth0.graegert.com/index.php?section=media1&act=download&path=/media/archive1/Books/EtherBook/&file=etherbook.pdf, 2003.

[Gro06] EtherCAT Technology Group, *Ethercat*, ETG, 2006.

[H. 98] H. Kopetz, *A comparison of CAN and TTP*, 1998.

[Har02] C. Harnisch, *Routing & Switching*, 2002.

[Hei02] Mathias Hein, *TCP/IP*, mitp-Verlag/Bonn, 2002.

[Her91] Hermann Kopetz, *Event-Triggered Versus Time-Triggered Real-Time Systems*, Proceedings of the International Workshop on Operating Systems of the 90s and Beyond (London, UK), Springer-Verlag, 1991, pp. 87–101.

[Her95] Hermann Kopetz and Andreas Krüger and Dietmar Millinger and Anton Schedl, *A Synchronization Strategy for a Time-Triggered Multicluster Real-Time System*, 14th IEEE Symposium on Reliable Distributed Systems (1995).

[Her04] Helmut Herold, *Linux/Unix Systemprogrammierung*, 2004.

[Hig98] Gary N. Higginbottom, *Performance evaluation of communication networks*, 1998.

[IEC07] IEC, *Mission and objectives*, `http://www.iec.ch/about/mission-e.htm`, 2007.

[IET94] IETF, *Integrated Services in the Internet Architecture: an Overview*, `http://tools.ietf.org/html/rfc1633`, 1994.

[IET97] ______, *Resource ReSerVation Protocol (RSVP)*, `http://tools.ietf.org/html/rfc2205`, 1997.

[IET98] ______, *An Architecture for Differentiated Services*, `http://tools.ietf.org/html/rfc2475`, 1998.

[J. 03] J. D. Miller, M. R. Anderson, E. M. Wenzel, B. U. McClain, *Latency Measurement of a real-time virtual acoustic environment rendering system*, `http://humanfactors.arc.nasa.gov/publications/20051222163516_Miller_2003_ICADMAWM.pdf`, 2003.

[J. 04] J. Kiszka, R. Schwebel, *Alternative: RTnet*, `http://www.rts.uni-hannover.de/rtnet/download/ad104705.pdf`, 2004.

[J. 05a] J. Corbet, A. Rubini and G. Kroah-Hartman, *Linux Device Drivers, Third Edition*, `http://lwn.net/Kernel/LDD3/`, 2005.

[J. 05b] J. Kiszka, B. Wagner, Y. Zhang, J. Broenink, *RTnet - A Flexible Hard Real-Time Networking Framework*, `http://www.rts.uni-hannover.de/rtnet/download/RTnet-ETFA05.pdf`, 2005.

[Jak] Jakob Engblom and Andreas Ermedahl and Friedhelm Stappert, *Comparing Different Worst-Case Execution Time Analysis Methods*.

[Jas02] J. Jasperneite, *Leistungsbewertung eines lokalen Netywerkes mit Class-of-Service Unterstützung für die prozessnahe Echtzeitkommunikation*, October 2002.

[Jen07] E. Douglas Jensen, *Time/Utility Functions*, `http://www.real-time.org/timeutilityfunctions.htm`, 2007.

[JH86] P. O'Reilly J. Hammond, *Performance analysis of Local Computer Networks*, 1986.

[JJ01] K. Watson J. Jasperneite, P. Neumann, *Real-time communication in industrial automation with switched ethernet networks*, 2001.

[Joh93] Johannes Reisinger and Andreas Steininger, *The Design of a Fail-Silent Processing Node for the Predictable Hard Real-Time System MARS*, Distributed Systems Engineering Journal, pp. 104-111 (1993).

[Jos05] Josh Aas, *Understanding the Linux 2.6.8.1 CPU Scheduler*, `http://josh.trancesoftware.com/linux/linux_cpu_scheduler.pdf`, 2005.

[Kos05] Marco Kosinski, *QoS-orientierte Kommunikation über Ethernet für verteilte, Linux-basierte Automatisierungsanwendungen*, April 2005.

[KP91] Phil Karn and Craig Partridge, *Improving round-trip time estimates in reliable transport protocols*, ACM Transactions on Computer Systems **9** (1991), no. 4, 364–373.

[KR05] P. McHardy K. Rechert, *Vordrängler - Queueing Disciplines: Traffic Control mit Linux*, 2005.

[Kut02] Friedrich Kutscher, *Ethernet in der Industrieautomation*, Diploma Thesis, 2002.

[LH04] J. Loeser and H. Haertig, *Low-latency hard real-time communication over switched ethernet*, 2004.

[Lim01] Ang Ngang Lim, *What is switching fabric?*, `http://searchstorage.techtarget.com/sDefinition/0,290660,sid5_gci214147,00.html`, 2001.

[LM06] A. Lueder and R. Messerschmidt, *Was ist Echtzeit?*, Industrial Ethernet (Ethernet-basierte Automatisierungsprotokolle), April 2006.

[Loc02] Dietmar Lochmann, *Digitale Nachrichtentechnik*, 2002.

[Lot01] Lothar Papula, *Mathematik für Ingenieure und Naturwissenschaftler, Band 3*, 2001.

[Lut06] P. Lutz, *Sercos iii*, Industrial Ethernet (Ethernet-basierte Automatisierungsprotokolle), 2006.

[MI06a] Modbus-IDA, *Modbus application protocol specification*, `http://www.modbus-ida.org/docs/Modbus_Application_Protocol_V1_1b.pdf`, 2006.

[MI06b] ———, *Modbus messaging on tcp/ip implementation guide*, `http://www.modbus-ida.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`, 2006.

[Mue00] F. Mueller, *Timing analysis for instruction caches*, 2000.

[Mül] Frank Müller, *Efficient Analysis of Temporal Properties for Real-Time Systems - A Formal Framework, Supporting Protocols, and an Implementation.*

[ODV06] ODVA, *Ethernet/ip^{TM} - cip on ethernet technology*, ODVA, 2006.

[oEI98] Institute of Electrical and Electronics Engineers (IEEE), *Token ring access method and Physical Layer specifications*, 1998.

[oEI04] ______, *Media Access Control (MAC) Bridges*, 2004.

[oEI05] ______, *Part3: Carrier sense multiple access with collision detection (CS-MA/CD) access method and physical layer specifications*, 2005.

[oEI06] ______, *Virtual Bridged Local Area Networks*, 2006.

[P. 04] P. Soetens, *HOWTO Port your C++ GNU/Linux application to RTAI/LXRT*, `http://people.mech.kuleuven.be/~psoetens/portingtolxrt.html`, 2003, 2004.

[P. 05] P. Cloutier, P. Mantegazza, S. Papacharalambous, *LXRT services (soft-hard real time in user space).*, `https://www.rtai.org/documentation/magma/html/api/group__lxrt.html`, 2005.

[Pop00] P. Pop, *Scheduling and communication synthesis for distributed real-time systems*, 2000.

[Ray01] John Ray, *Der neue Linux Hacker's Guide*, 2001.

[RDH07] H. Huckeba R. Dlugy-Hegwer, *Designing and Testing IEEE 1588 Timing Networks*, `http://www.symmttm.com/pdf/Gps/PTP_WP2.pdf`, 2007.

[Ric05] Vito Ricci, *Fitting Distributions with R*, `http://cran.r-project.org/doc/contrib/Ricci-distributions-en.pdf`, 2005.

[Ron04] Ron Bodtcher, K Kirkendall, Jim Lorenz, Rick McDonald, *CCNA 3 and 4 Companion Guide*, Cisco Press, 2004.

[Ron07] Frank Ronneburg, *Debian-Kernel-Pakete erzeugen*, `http://debiananwenderhandbuch.de/kernelbauen.html`, 2007.

[Ros06] M. Rostan, *Ethercat - der ethernet-feldbus*, Industrial Ethernet (Ethernet-basierte Automatisierungsprotokolle), 2006.

[S. 07] S. Smolorz, *Echtzeit-Linux mit Xenomai*, `http://www.emlix.com/fileadmin/emlix/dokumente/FA_Xenomai.pdf`, 2007.

[Sch06] B. Schneider, *Latenzzeitmessung in einem Linux-Echtzeit-Framework*, August 2006.

[Sec06] ISO Central Secretariat, *ISO in brief*, 2006.

[SGG05] A. Silberschatz, G. Gagne, and P. Galvin, *Operating System Concepts, 7th edition*, John Wiley and Sons, 2005.

[Sma03] Jeffrey L. Small, *Factors that influence the decision to change to switch-fabric backplane technology*, http://www.embedded-computing.com/pdfs/Fairchild.Win03.pdf, 2003.

[Sta03] William Stallings, *Betriebssysteme*, Pearson Studium, 2003.

[Tho05] Thomas Graf, Greg Maxwell , Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, Pedro Larroy, *Linux Advanced Routing & Traffic Control*, http://lartc.org/, 2005.

[Tie03] Eric Tierling, *Windows Server 2003*, 2003.

[Tim00] Tim Waugh, *User-level device drivers*, http://people.redhat.com/twaugh/parport/html/ppdev.html, 2000.

[Tzi99] Tzi-cker Chiueh, *RETHER: A Software-Only Real-Time Ethernet for PLC Networks*, http://www.usenix.org/publications/library/proceedings/es99/full_papers/chiueh/chiueh.pdf, 1999.

[Ver07] Sven Vermeulen, *Gentoo Linux Kernel Guide*, http://www.gentoo.org/doc/en/gentoo-kernel.xml, 2007.

[Vir07] Virtuelles Software Engineering Kompetenzzentrum, *Begriffsdefinition: Echtzeitfhigkeit, Rechtzeitigkeit, Gleichzeitigkeit, Jitter, Determinismus*, http://www.softwarekompetenz.de/?28612, 2007.

[VS06] D. Vasko V. Schiffer, P. Kucharski, *Cip safety auf ethernet/ip*, Industrial Ethernet (Ethernet-basierte Automatisierungsprotokolle), 2006.

[WB05] H. Woern and U. Brinkschulte, *Echtzeitsysteme*, Springer-Verlag, 2005.

[Wik07a] Wikipedia, *Application programming interface*, http://en.wikipedia.org/wiki/API, 2007.

[Wik07b] ______, *Application specific integrated circuit*, http://en.wikipedia.org/wiki/Application-specific_integrated_circuit, 2007.

[Wik07c] ______, *Burst transmission*, http://en.wikipedia.org/wiki/Burst_transmission, 2007.

[Wik07d] ______, *Carrier sense multiple access with collision avoidance*, http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance, 2007.

[Wik07e] ______, *Electromagnetic compatibility*, http://en.wikipedia.org/wiki/Electromagnetic_Compatibility, 2007.

[Wik07f] ______, *Ethernet*, http://en.wikipedia.org/wiki/Ethernet, 2007.

[Wik07g] ______, *Fieldbus*, http://en.wikipedia.org/wiki/Fieldbus, 2007.

[Wik07h] ______, *Institute of Electrical and Electronics Engineers*, http://en.wikipedia.org/wiki/IEEE, 2007.

[Wik07i] ______, *Kernel*, http://de.wikipedia.org/wiki/Kernel_(computer_science), 2007.

[Wik07j] ______, *Mac address*, http://en.wikipedia.org/wiki/MAC_address, 2007.

[Wik07k] ______, *Master boot record*, http://en.wikipedia.org/wiki/Master_boot_record, 2007.

[Wik07l] ______, *Multiplexverfahren*, http://de.wikipedia.org/wiki/TDMA#Zeitmultiplexverfahren_.28TDM.2C_TDMA.29, 2007.

[Wik07m] ______, *Real-time*, http://en.wikipedia.org/wiki/Real-time, 2007.

[Wik07n] ______, *Request for Comments*, http://en.wikipedia.org/wiki/Request_for_Comments, 2007.

[Wik07o] ______, *Rtai*, http://de.wikipedia.org/wiki/RTAI, 2007.

[Wik07p] ______, *Token ring*, http://en.wikipedia.org/wiki/Token_Ring, 2007.

[Wik07q] ______, *Translation Lookaside Buffer*, http://en.wikipedia.org/wiki/Translation_Lookaside_Buffer, 2007.

[Yag01] Karim Yaghmour, *Building a Real-Time Operating System on top of the Adaptive Domain Environment for Operating Systems*, http://www.opersys.com/ftp/pub/Adeos/rtosoveradeos.pdf, 2001.