Technical report, IDE0705, January 2007

# Combining the Good Things from Vehicle Networks and High-Performance Networks

Master's Thesis in Electrical Engineering

Herbert Ecker, Misikir Armide


School of Information Science, Computer and Electrical Engineering
Halmstad University

# Combining the Good Things from Vehicle Networks and High-Performance Networks

Thesis submitted for the degree of Master of Science in Electrical Engineering

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

January 2007

## Preface

This Master's thesis is the final submission for the degree of the Master of Science at Halmstad University, Sweden.

We would like to thank our supervisor Xing Fan for her support, advice and the way she has motivated us throughout our project. We would also like to express our gratitude to Christopher Allen for his valuable help on the language of this thesis.

Special thanks also go to our families for their support and encouragement during the whole time.

Herbert Ecker & Misikir Armide
Halmstad University, January 2007

# List of figures

## Abstract

The aim of this Master's thesis is to develop a solution for combining speed and performance of switched Ethernet with the real time capability and determinism of sophisticated in- vehicle networks. After thorough research in vehicle network standards, their demands and features, the Flexible Time Division Multiple Access (FTDMA) protocol of FlexRay was chosen to be applied on a switched Ethernet architecture since it can accommodate both hard real time tasks and soft real time tasks. To provide hard real time capability, what this paper focuses on, a media access method was developed by creating static TDMA schedules for each node's sending and receiving port according to a certain traffic assumption. To validate the developed media access algorithm several examples with different traffic assumptions and architectures were generated and investigated based on their sending and receiving utilization. A second method for validating and thus proving the functionality of the algorithm was by simulation. Therefore the Matlab Simulink media library extension TRUE TIME was used to simulate a simple example with 100% sending and receiving utilization for each node.

# Contents

# 1. INTRODUCTION

## 1.1 Context

The basic idea of combining approaches of different networks is to gain possibilities and increase performance. Since the aim of this work is to combine the advantages of high performance networks and in-vehicle networks, the combination of high speed performance and determinism is the goal.

With the rising amount of functionality-, safety-, comfort-, driver assistance-, and efficiency demands in vehicles in conjunction with the complexity of huge numbers of electrical components and their wiring, communication between these components increases enormously. For this reason networking within vehicles is indispensable to keep up to the rapidly rising amount of electronics within automobiles.

There are different applications with different demands in bandwidth, fault tolerance, response time, safety issues, etc, which in turn raises the need to partition the network into several distinct domains depending on their demands and functionality. Each domain uses different network standards, protocols and technologies, depending on the particular demands in the domain.

Since the automobile industries, as well as large numbers of research institutes are constantly bent on developing and improving the different standards, the network protocols used in vehicles are very sophisticated and mature.

A great deal of research has also been done in the real time capability of the Ethernet standard. Although it is generally known that the Ethernet standard IEEE 802.03, with CSMA/CD as media access protocol, is not real time capable, there are several approaches with different methods for media access showing that real time traffic over Ethernet is possible. Especially for industrial networks, where real time capability is essential, research has pointed more and more in the direction of switched Ethernet instead of the use of field busses, due to its simplicity, publicity and performance issues. The main advantage of switched Ethernet in comparison with the conventional version is that collisions of packets or frames are completely eliminated, since data is exclusively delivered to the appointed receiving node. This makes simultaneous communication between different nodes possible, as long as the destination nodes differ from each other. Furthermore the nodes communicate in full duplex, which increases the performance enormously.

The aim of this thesis is to combine the advantages of both existing in-vehicle standards and high performance networks in order to make real time communication possible. This raises two possibilities to fulfil this goal: on the one hand trying to apply the switched Ethernet technology to an existing standard of in-vehicle networks in order to increase the achievable bandwidth limit of field busses, or on the other hand, to meet the increasing bandwidth and real time requirements in data networks, taking one of the

protocols from an in-vehicle network and applying the advantages of this standard to a switched Ethernet topology.

Since, as mentioned before, the standards and protocols of in-vehicle networks are already very sophisticated, this work will deal with the second alternative, to integrate one of these mature standards into a switched Ethernet network.
Therefore thorough research in the areas of in-vehicle communication and switched Ethernet has had to be done.

In the following theoretical background chapters the different in-vehicle standards and the switched Ethernet technology are explained. Afterwards the methodology of the project is discussed, followed by a validation of the developed algorithms by utilization calculation and a simple simulation.

# 2. APPLICATION BACKGROUND

In order to find the most appropriate protocol for the purpose of this project, substantial research in the field of existing in-vehicle network standards had to be done. The following subchapters give an overview on the different classifications within vehicle networks, the demands they have to meet and their specifications with their respective advantages and disadvantages as well as their possible field of application. Another subchapter deals with switched Ethernet and explains the technology in further detail.

## 2.1 Classification of automotive network domains

In respect of the different functions and network speeds, the Society of Automotive Engineers (SAE) has introduced three basic categories of in-vehicle networks; class A, B and C networks. .

To find the most adequate protocol, a classification into the different demands concerning their real time capability is useful as well. In order to handle the arising complexity of in-vehicle networks, the system has to be analyzed with respect to the different application demands, and is logically split into several sub-networks.

### *2.1.1 Non safety critical / SAE Class A and B*

Applications and devices which are not safety critical and neither have real time demands nor hard latency constraints, are predominately located in the body domain of the vehicle. Typical examples for such devices are: electrical window lift, rain sensor, windshield wipers, electrical seats and exterior mirror adjustment, lighting, dashboard, control lamps, door status and lock, windscreen washer system, climate control, windshield heating etc.

Since all these devices have no demand for high bandwidth it is not essential to have a bus system with high capacity. Furthermore devices in this domain have to exchange short data fragments rather often. Although these applications are not time critical, there is nevertheless the need for communication among each other. So is it for example important for the windscreen wiper to get its data from the rain sensor, as well as it should be possible for the responsible ECU to save the data from seat and mirror calibration for different drivers.

This category contains the SAE classes A and B, whereas class A networks operate on low speed with data rates below 10 Kbit/s for not time critical communication within the body domain.

Class B Networks operate on medium speed with data rates from 10 Kbit/s up to 125 Kbit/s and are essentially used for the general information transfer between the ECUs. So data coming from various sensors gets available to several ECUs, and makes

consequently redundancy of sensors dispensable. Class B networks are like class A networks applied in the electronic body, and are not used for transferring data which is essential for the main operations of the vehicle.

### *2.1.2 Safety critical / SAE Class C*

The powertrain and chassis domain in vehicles contribute a main part in safety issues and have therefore very strict real time and fault tolerance demands. The function of the powertrain domain is controlling the engine and transmission of a car. The chassis network domain is responsible for the control of steering, braking, suspension, Automatic Stability Control (ASC), Antiblock Braking System (ABS), Electronic Stability Program (ESP) as well as for X-by wire systems (already used in avionics, currently in development for passenger cars and in near future implemented). X-by wire is a term with respect to Steer-by-Wire and Brake-by-Wire systems where steering systems work without any mechanical connections between e.g. steering wheel and wheels.

The demands in bandwidth, real time requirements and fault tolerance are almost equal for the powertrain and chassis sub-network, though, since chassis functions contribute more to the stability of the vehicle, it is more safety critical.

Both domains require deterministic real time behaviour, low latencies, high predictability, high data rates and performance, a synchronized vehicle-wide clock, mechanisms for fault tolerance and error detection, and a high degree of dependability and scalability. Also the ability for fast data exchanges with other sub-networks and among each other is essential. Also a certain amount of redundancy, depending on the susceptibility of the device caused for example by aging or production errors, has to be taken into account.

The Society of Automotive Engineers has classified networks used for this purpose as class C networks, operating on high speed with data rates between 125 Kbit/s and 1 Mbit/s, used for deterministic and safety critical applications in the powertrain and chassis domain.

### *2.1.3 Active / Passive safety*

Systems in a vehicle for avoiding accidents, like vehicle's tires, brakes, handling and visibility are so called active safety features. Also Adaptive Cruise Control (ACC), where the vehicle's speed is regulated depending on the velocity of the car in front, is an active safety feature. Passive safety features like seat belts, airbags, rollover sensors, etc. on the other side help the driver and passengers to stay alive and uninjured during a crash. As well as safety critical applications also active and passive safety functions have very high demands in the in-vehicle network infrastructure, since they have to react on influences from outside very quickly. In the case they are connected to the in-vehicle network (what is not for sure for each application e.g.: belt pretensioners), high speed real time communication with a minimum latency and absolute dependability is demanded.

### *2.1.4 Telematics / Infotainment*

Telematics has a number of applications within a vehicle, starting with satellite navigation to enable the driver to locate a position, plan a route and navigate a journey, over mobile data communication to e.g. provide mobile internet connection, to vehicle tracking systems. Infotainment services can use these connections to the outer world and provide the vehicles passengers with the entertainment and information they want at any time and anywhere. Typical examples of infotainment / telematics applications are digital TV, email, hands free phone, Internet, navigation, CD, DVD, emergency road service, rear seat entertainment and gaming.
The requirement from the vehicle network to provide the mentioned applications is also real time capability, but has unlike before nothing to do with safety issues, concerning the locomotion and the passenger safety, at all. The tasks in this domain are to fulfil the Quality of Service (QoS) demands of the multimedia data streams, what means that high bandwidth is necessary because of the huge amount of data, though latency time is far not as important as it is for example in the safety critical network domain.
Therefore other safety issues come up, concerning integrity and confidentiality of the data transmitted to and from the vehicle.

## 2.2 How to meet the demands

As described in the previous subchapters, there are very large numbers of different applications whit certain demands inside a modern automobile. These demands range from low bandwidth requirements with real time capability and strict constraints concerning delay and jitter, to very high bandwidth demands with real time capability as well, but smoother requests in delay and jitter. Other applications for example need real time capabilities as well, although with little bandwidth and are able to send their data very frequently.

To meet all demands of the different applications, several technologies and methods are necessary, since one system can hardly satisfy all these requirements alone. Automotive companies, researchers and companies working in related fields have developed multiple, already very mature, protocols which are currently used in the present generation of cars.

One question raised in trying to meet the various application demands is, what kind of traffic is better handled using event-triggered or time-triggered media access method.

### *2.2.1 Event-Triggered vs. Time-Triggered*

Event-Triggered

Messages are transmitted immediately due to occurrence of an event. Hence, the protocol has to assure access to the bus for such messages, and has to use some strategy to avoid collisions when two nodes want to transmit messages simultaneously. For example, CAN assigns a priority to each message, so that the highest priority message gets access to the bus. Due to the unpredictability of events dynamic scheduling is used. The main

advantage of event-triggered communication is in terms of utilization of bandwidth. Since the bus is only used when a node has some ready messages to transmit, efficiency in bandwidth usage is higher. Also scalability is another advantage of event triggered systems. The drawbacks are unpredictable jitter for real time communications and difficulties in detection of node failures.

Time-Triggered

Nodes transmit their messages at predefined time slots, what means it is known in advance (at system design time) which station sends a certain amount of data to an appointed destination at a defined time. This type of protocol is well suited for periodic transmissions. Prediction about the system behaviour is easy, since the frame scheduling is statically defined. This simplifies analysis, design, testing and maintenance. Also error detection in the system gets easier due to the regular message transmission of TDMA. Inefficiency in usage of bandwidth is one drawback of time triggered protocols, scalability is the other one. With addition of one node, the message schedule has to be changed and the other nodes have to be informed about the change. Time-triggered protocols are often used in real time systems, requiring high level of dependability and guaranteed delay.

Due to this, X-by-wire and safety critical applications mainly use time-triggered approaches, since the predefined access method to the bus and bounded response time provided by TDMA meet the required real time demands for such applications.

## 2.3 Existing in-vehicle standards

As discussed above, the various application domains demand different performance criteria, safety needs and QoS. According to their requirements a lot of standards have been defined and used in in-car embedded systems. These protocols have different features associated with their functions. They are also classified according to the SAE classes because of having different data rates. The triggering mechanism used in such protocols can be event triggered or time triggered or a combination of both.

The following chapters discuss some of the major existing standards of in-vehicle protocols in further detail.

### *2.3.1 Controller Area Network (CAN)*

This is the most widely used standard in many networked embedded control systems. It is a serial bus communication developed by Bosch in the mid 1980s. It became an ISO standard on twisted pair of copper in 1994. CAN is classified as SAE class C networks. Low speed CAN, being SAE class B, is used for exchange of data between ECUs for non-safety critical demands. Several applications which require real time communications, like in the power train and chassis domain, use CAN. But due to its fault tolerance facilities, it can not be used in the X-by-wire for safety critical

applications. CAN uses a multi master broadcast communication system. Every station has equal rights to access the bus, whereas the succeeding message is the one with higher priority.

CAN works in an event-triggered manner and uses priority assignment to avoid collisions on the network. As media access method Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) is used.

Bit wise arbitration is used, where two signal levels (0 and 1) are the dominant and recessive bits respectively. When two or more stations are transmitting their frame simultaneously, the frame with the higher priority (lower numerical value) is delivered to the receiver without being destroyed by the collision. When a node is transmitting a recessive bit and when there is a dominant bit on the bus, the node automatically stops transmitting its bits.
A CAN frame has an identifier transmitted within the frame, which is equivalent to its priority. Two versions of CAN are available which differ in the length of their identifier. Standard CAN, CAN 2.0A, uses a 11 bits identifier and Extended CAN, CAN 2.0B, uses a 29 bits identifier. Since there are a sufficient number of identifiers in CAN 2.0A, this version is used for most in-vehicle communications. A station broadcasts a message with its identifier on the bus, and any station which is interested in the frame can process the data by filtering the identifier. The identifier does not only show the priority, also the type of data in the frame.

The CAN network standardizes the physical and data link layer (DLL). Higher layer protocols are needed for efficient operation. These protocols define how the CAN protocol is used in applications by specifying start up procedures, handling fault conditions, content of messages and procedures for packaging application messages into frames. The use of CAN in many applications has led to develop higher layer protocols, such as SAE J1939, which is used in Scania's trucks and buses, CANopen, DeviseNet, and CAN Kingdom.

Some of the major advantages of CAN are

- Message prioritization, which helps higher priority messages to be transmitted efficiently.
- Bit wise arbitration for avoiding collisions.
- Higher priority messages have shorter latency times.
- Easy scalability

The drawbacks of CAN are

- Due to higher priority messages, some other messages may miss their deadline. These messages may be important at the application level even if they have lower priority.

- Due to its fault tolerant facilities, it can not be used for X-by-wire and safety critical applications. CAN has a fault containment facility that helps it to recognize and disconnect faulty nodes.
- Nodes can disturb the whole system by transmitting a long message, which is outside their limit. If an undetected faulty node sends continuously a dominant bit, it may totally control the whole bus

A lot of researches have been conducted to improve the drawbacks of CAN. Most of them were done at higher protocol layers of, e.g. Time Triggered CAN (TTCAN). Also some scheduling algorithms were designed to guarantee the deadlines of messages, but most of them require larger bandwidth to attain their aim.

*2.3.2 Time Triggered Protocol Class C (TTP/C)*

TTP/C is the major part of the Time Triggered Architecture (TTA). TTA and TTP/C were developed at the University of Technology, Vienna, Austria. TTP/C has many features and services related to dependability, group membership algorithm and support for mode changes. It is a purely time-triggered protocol, using TDMA as media access method. Each node transmits its message in a predefined time slot. During one TDMA cycle, each node transmits one frame in its slot. The slot size can vary for different nodes, but the size of a slot given to one node is the same in every cycle. A Cluster cycle is a sequence of a fixed number of TDMA cycles.

TTP/C is a composable protocol. Composability is the property related to the integration of a node to a complete system, while the behaviour of a station does not change during the integration. Therefore each subsystem can be analyzed alone without affecting the integration.
Transmission on a TTP/C network is done by redundant channels, what means in each channel the same message is transmitted.

TTP/C can be used with bus or star topology, whereas the star topology offers more fault tolerance than the bus topology. Using dual star topology solves the single point of failure problem.

Since TTP/C has a high fault tolerance, it is used for X-by-wire applications. An advantage of TTP/C is, that it can control a node when it is transmitting, whether the node is using its specified time and range or not. A bus guardian is used for this purpose. Since messages will not miss their deadlines, suitability for hard real time is also another advantage of TTP/C.

The main drawback of TTP/C is its inflexibility for adding new nodes. Since every node is assigned to its time slot statically, it is not possible to change it dynamically. Hence addition of a new node or a new service requires the whole system to be reconfigured.

TTP/C is designed for safety critical applications in automotive control systems, aircrafts control system, power plants or air traffic control. It can also be used for active/passive safety applications.

### *2.3.3 FlexRay Protocol*

The major automotive companies and companies which are working in related fields, like BMW, Bosch, Daimler-Chrysler, General Motors, Motorola, Philips and Volkswagen, started to develop this protocol which is becoming one of the most important standards for in-vehicle communication. The main aim of their research was to produce a high speed flexible fault tolerant protocol. When they started to develop FlexRay, it was partially based on BMW's ByteFlight, which was created for passive safety demands. However, safety critical applications, like X-by-wire, which demand deterministic communication with strict delay requirements, are of great concern in the embedded control platform in vehicles. For such applications TTP/C was the best option. For this reason FlexRay comes up with the combination of these two protocols to satisfy most demands of in-vehicle communication.

FlexRay combines the major ideas of TTP/C and ByteFlight. These are:

- TDMA for static scheduling, i.e. time-triggered messages are transmitted in their pre-defined time slots.
- FTDMA for dynamic scheduling, i.e. event-triggered messages use minislots according to their priority set at the start up.

The combination was implemented by allowing two different sections to be included in one major cycle, called the communication cycle. These are: a TDMA section which handles the time-triggered transmission, and a FTDMA section for the event-triggered transmissions.
The communication cycle begins with the TDMA part and the FTDMA follows at the end.

The TDMA section contains a fixed number of time slots with equal size in every communication cycle. These time slots are assigned to nodes for their transmission. Each node has access to one or more time slots. The time slots assigned to a node are used if the node has a message ready for transmission; if not it remains idle. The allocation of the slots to the nodes is done statically at system design time. Hence in this section, the prediction of the system is simplified, since at any time the type of message and the sending node are known. This section provides reliable communication with guaranteed jitter and latency through fault tolerant clock synchronization. It is more or less similar to TTP/C, except of two points. First, in TTP/C the size of time slots is different for different nodes. The other point is the allocation of slots to the nodes, since one node is given only one time slot and there is no option of multiple slots. A bus guardian is used like in TTP/C to prevent the "babbling idiot" problem, where a faulty node tries to dominate the bus by sending out of its specification.

The FTDMA section performs dynamic scheduling of messages. If an event triggered message occurs at the node, the minislots of this section are used to get access to the bus. Messages are given a unique identifier which shows also the priority of the message. The lower the number, the higher priority it has. Prioritization of messages is also done statically at system design time. Each node has a slot counter, which helps it to know in which slot to transmit its event triggered message. The counter is set to zero every time at the beginning of the FTDMA section. If a node has a message ready for transmission, it checks whether the slot counter value matches its message identifier or not. If so, it starts to transmit immediately. At the end of every transmission, all node counters are incremented by one. If there is no message being transmitted on the bus, all nodes wait for a short period of time and increment their counter. This waiting time period is much less than the time required for transmitting a frame. Hence, a message with a small identifier value is transmitted at the beginning, solving the problem of high priority messages being delayed waiting for their time slot as in TTP/C. Since it may not be possible to transmit all messages from each node due to the fixed length of the FTDMA section, the slot counters do not reach their maximum value. Due to this, low priority messages may have to wait for the next communication cycles to get access to the bus. Therefore, predictability of the system for dynamic frames is not fully certain. A bus guardian is not used in this section.

For an assumed topology of four nodes connected to a bus using FlexRay, the two communication cycles are shown in the figure below. In both cycles the static segment has six equal sized slots and nine minislots. In the static segment of the first cycle, node 1 owns two timeslots, slot one and slot three. Node 2 also uses two time slots, slot two and slot five respectively. Node 3 uses slot four, and slot six is assigned to node 4. Node 4 is not using its timeslot in the first cycle, whereas all remaining nodes use their slots. Slot six is idle during this cycle. In the second cycle the slot allocation is equal to the first cycle, but the method by which nodes make use of their timeslots differs. Node 1 and node 2 use their slots as before, whereas node 3 does not have any message ready to transmit during this cycle. Node 4 uses its time slot this time.

In the minislot section, the nodes transmit according to their messages' priority. In the first cycle node 2 has the highest priority message, so it starts to transmit during the first mini slot. After finishing, the next high priority message from node 1 is transmitted. Afterwards the remaining mini slots remain idle, since there is no ready message. In the second communication cycle, there is one unused mini slot at the beginning of the FTDMA section. It implies that there is no message with highest priority. After that short duration node 3 occupies the next slots to transmit its frame. After the transmission from node 3 is finished, two mini slots pass with out any traffic. Then node 4 uses the remaining slots to send its frame.

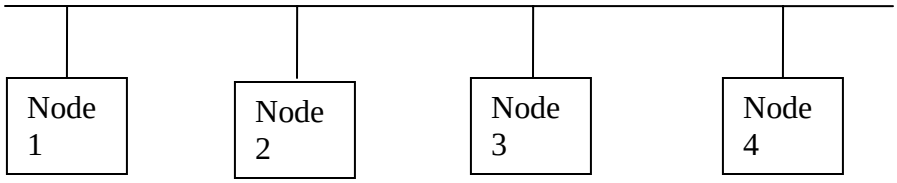



Figure 2.1 Four nodes connected to one FlexRay communication channel

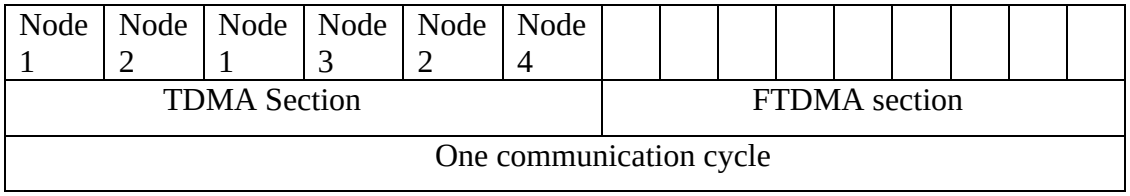
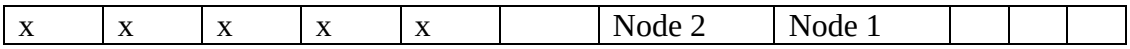



Figure 2.2 Example showing communication cycle of FlexRay with static and dynamic segments, and how the nodes are allocated in the static section.

Fist cycle




Second Cycle

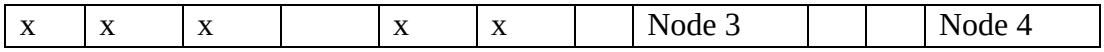



Figure 2.3 Two communication cycles for the above example, showing how messages are transmitted in the static and dynamic segment.

A FlexRay frame contains three parts, header, payload and trailer. The frame ID, length of payload, header Cycle Redundancy Check (CRC) and cycle counter are included in the header part. The frame ID shows the type of the frame and its priority during dynamic scheduling. The length of payload, as its name implies, shows the length of data in the payload. The header CRC checks errors in the header. The cycle counter gives the current value of the counter, and is incremented every time the communication cycle starts. The payload consists of 254 bytes of data the frame transmits. The last part of the frame, the trailer, is a 24 bit CRC value.

Flexibility and predictability are the major advantages of FlexRay, what makes it valid for X-by-wire applications. Another advantage is that single, dual and mixtures of single and dual transmission channels are supported, helping the designer to select a suitable topology according to the redundancy required in the network.

One of the drawbacks of FlexRay is that it is not a composable protocol. Since it uses the principles of minislotting, the message time may not be the same as in the test and

analysis phase. Many researchers pointed out one main disadvantage of FlexRay [24]: since the nodes within a cycle know exactly when they have to send, as well as they know what and when to expect something from other nodes, they can "learn" a wrong TDMA schedule in case a node sends at the wrong time. Another disadvantage is that FlexRay does not provide an acknowledgment message or membership agreement. If these services are required, they have to be implemented in the software.

### *2.3.4 Time Triggered CAN (TTCAN) Protocol*

This protocol uses the standard CAN in its physical and data link layer, in addition it uses a time-triggered approach in the higher layers. It is designed to provide deterministic communication, which is not guaranteed by CAN, by avoiding high latency times. In addition, it utilizes the physical bandwidth of CAN efficiently. The operation is based on a global time synchronisation, initiated by the time master. The time master transmits periodically a "reference message" which starts a new basic cycle. The basic cycle, which is similar to the FlexRay communication cycle, is defined as containing one or more time triggered windows (called exclusive windows) and one event triggered window (called arbitrating window). Arbitrating windowing works like the standard CAN, it gives services to event triggered transmissions according to their priority. The exclusive window takes care of all time-triggered transmissions. Free windows, which are used for future extension, can be included in the basic cycle. When a node requires more windows or when extension of bandwidth is required, these windows are changed to exclusive or arbitrating windows.

TTCAN predefines more than one time masters to avoid single point of failure associated with only one time master, since the whole operation of TTCAN depends on the time master. These time masters – called potential time masters- have their own identifier, which is related to their priority. The bit wise arbitration mechanism in CAN is used to decide which time master should serve the network. If there occurs an error or a missing reference message, all potential time masters recognize it within short time due to a time out. Then they send their reference message and the one with the higher priority becomes the time master. The other potential time masters stop sending their reference messages and they synchronize themselves to the basic cycle.

When using the TTCAN protocol, it is not a must for a node to know all the messages on the bus. A node only needs to know the information required for sending and receiving time-triggered messages and for sending event-triggered messages. This advantage helps to utilize efficiently the memory in the hardware realization. The other merit of TTCAN is, since it is based on CAN, it uses the efficient error detection mechanism of CAN.

The disadvantage of TTCAN is the transmission speed limit, 1Mbit/s, imposed by CAN for higher bandwidth applications. Since TTCAN also uses TDMA, retransmission of messages would affect the whole TDMA schedule. So, lost messages do not get retransmitted. Another drawback is that it is not composable and it does not support membership agreement, bus guardian and reliable acknowledgment.

TTCAN can be used in non-safety critical applications because it is not a fault tolerant protocol.

### *2.3.5 Local Interconnect Network (LIN)*

LIN is a low cost serial communication bus with one master node and multiple slave nodes. LIN is considered as class A network even if it has a speed up to 20Kb/s. It is an open standard which is used in simple control units, like door lock and seat control. It is used in non-safety critical applications. CAN is used as a backbone for LIN interconnection. The master and slave nodes in a LIN cluster are connected with a common bus. Self-synchronization of slave nodes by a sequence of clock signal, which is generated by the master node, is one of the main properties of LIN. The master node uses a schedule table to determine when and which frame is to be transmitted.

LIN offers optimization of energy by making nodes sleeping when it is necessary, for example when the engine is not running.

### *2.3.6 Media Oriented Systems Transport (MOST)*

MOST is used to support infotainment and telematics demands, where huge amounts of data have to be transmitted. Time-triggered and event-triggered transmissions at speeds of 25Mb/s are supported. Audio and video data, GPS navigation and entertainment like radio are typical applications supported by MOST.

MOST uses ring topology with multiple rings for redundancy. Plastic optical fiber is used at the physical layer. For synchronization one time master is used to generate the necessary timing signals. For network management, connection management and power management, one node (for each purpose) can be optionally configured.
Encryption and authentication are not built in MOST.

Although MOST has to handle real time traffic as well, it is of minor importance for this project since it was designed for multimedia tasks and not for hard real time traffic with hard deadline constraints.

*2.3.7 ByteFlight*

BMW was first starting to develop ByteFlight. ByteFlight uses Flexible TDMA as medium access control protocol based on a star topology. It is a high-speed network intended to replace CAN, and provides a high degree of determinism with a transmission rate of 10Mbit/s. There are a number of similarities between the FlexRay and ByteFlight. ByteFlight for example also uses the minislotting concept.

The physical medium used is plastic optical fiber. One dedicated master node is responsible for clock synchronization, whereas any node can be configured to be the master. Typical applications for ByteFlight are air bag systems and sear-belt tensioners. (Passive safety demands)

*2.3.8 TTP/A*

*TTP*/A is the TTP protocol with low speed application, i.e. class A. TTP/A has the same objective as LIN, and provides the same level of communication, but additionally it is fault tolerant because of being part of TTA. Like with LIN also a Master node is for synchronization. But it is not currently used in commercial vehicles.

**2.4 Choice of method**

As described in the previous chapters, the various in-vehicle communication protocols have different methods to satisfy the increasing requirements of automotive demands. One way is prioritization of messages to ensure real time communication like in CAN, or using TDMA to guarantee delay and jitter avoidance in safety critical applications like in TTP/C. FlexRay and Byteflight use FTDMA to assure access to the bus for time and event triggered messages, just to mention few.

Combining these methods with high performance networks, like switched Ethernet, the resulting approach can be of great importance to improve speed and performance on the one hand, and determinism and predictability on the other hand. (What the actual motive of this project is.)

For the aim of this work the TDMA algorithm of FlexRay was chosen to be aligned on switched Ethernet. The reason for this choice among other in-vehicle protocols is the ability of FlexRay to accommodate event and time triggered messages in one communication cycle. Hence, this method should be able to support hard real time (HRT) and soft real time (SRT) traffic in a switched Ethernet environment, with guaranteed delay for the HRT part.

Although this paper mainly focuses on developing methods for the hard real time part, and thus, the time triggered part of the FTDMA protocol, the idea of not choosing a purely TDMA protocol was to have room for further improvements. Another reason for choosing the method of FlexRay was that it is used in some high developed cars, like BMW X5 and 7.

## 2.5 Switched Ethernet

Ethernet is the most commonly used LAN technology nowadays. Originally, it was developed by Xerox Corporation in 1970's using coaxial cable at 3Mbps of data rate. The Carrier Sense Multiple Access / Collision Detection (CSMA/CD) protocol was used to allow multiple users. After its first success, Digital Equipment Corporation and Intel Corporation joined Xerox in 1980 in the development of 10Mbps Ethernet. These three companies also worked on the specification of the Ethernet Version 1.0, which was the basis for the IEEE 802.3. Following its first version, many improvements were made to satisfy the increasing demand of data communications. During all these three decades, it remained the best choice for different LAN sizes due to the following reasons:

- High data rates, starting from 10Mbps, 100Mbps, 1Gbps and 10Gbps.
- Very cost effective.
- Easy installation, maintenance and upgrade.
- Available and dominant in the market
- Supports most of other network protocols.

The IEEE standard, IEEE 802.3, specifies the configuration of Ethernet networks, interaction between different network elements, assuring compatibility of different Ethernet products, access methods to the medium, types of cables used and data rates. The standard describes also the frame format. The basic Ethernet frame contains 8 bytes of preamble for indicating the beginning of the packet and for synchronization purposes, 6 bytes of destination address, which shows a single or a group of receivers, 6 bytes of source address, 4 bytes of length and /or type of the data being transmitted, 46 to 1500 bytes of data and 4 bytes of frame check sequence which contains the CRC value.

Ethernet is a shared medium. Every station has the right to access the medium at any time. There is no priority among them. But at a specified time, there is only one frame on the medium. Hence, there is a probability that two or more stations start to transmit their frames simultaneously, which results in corruption of data due to a collision. The CSMA/CD protocol is used to resolve such problems. CSMA/CD allows nodes to compete for the channel and provides counteractive measures to avoid collisions occurring due to simultaneous transmissions.
This protocol was developed to solve the problems associated with two or more nodes trying to transmit at the same time on the same medium. CSMA/CD implies the following features:

- Carrier sense: each node senses the channel whether there is traffic or not.
- Multiple access: when the medium is free, nodes with ready frames start to transmit.
- Collision Detection: If two or more stations transmit simultaneously, collisions occur. The stations automatically recognize the collision and stop transmitting their frames. They start retransmission after a random length of time, selected by a Back-off Algorithm.

From the start of its development, Ethernet used a shared medium for 10Mbps. Shared Ethernet uses hubs or repeaters for interconnecting the nodes. For this purpose, the star topology is mostly used. The hub being the central node, and the other nodes connected to it. Ethernet hubs repeat data they receive from one of their ports and transmit it to all the other ports. Repeaters help in amplification of the signal to support transmission between far distanced stations without signal degradation. The problems associated with shared Ethernet are:

- The network becomes very slow when there are many users with simultaneous messages.
- Nodes have to compete with each other for shared resources.
- Total bandwidth is limited even if there are many ports.
- Half duplex is limiting connection speed.

These problems forced researchers to develop switched Ethernet, which improves the network performance, since increasing demands in different applications with regard to speed and bandwidth can hardly be satisfied by shared Ethernet.

Ethernet switches avoid the problem of collisions by allowing multiple transmissions at the same time. There is no need of CSMA/CD any more, since every node has access to the medium simultaneously and the switch transmits the frames to the exact destination ports and not to every single port, what is the main advantage of switches compared to hubs. Switches examine every packet and forward it to the appropriate port; they do not repeat the packet and send it to all ports, like hubs do. The self learning behaviour of switches makes them to know the MAC address of the nodes residing in each segment and the interface port they are connected to. They store this information in their tables. When a packet is received at one port, they look up in their tables to which port the packet has to be forwarded.

Since every node has access to the switch without competing with other nodes, the bandwidth is not shared among the other nodes. This gives full bandwidth to each node connected to the switch. Another advantage is the support of full duplex, what allows simultaneous reception and transmission for each node. Switched Ethernet also offers the capability of dividing a large network into several smaller logical groups, improving performance of the small groups.

The two basic technologies of switched Ethernet are cut-through and store-and-forward. The difference between these two technologies is the way the switch forwards the packets. Using cut-through, the switch reads the destination address of the arriving packet and forwards it to that destination without waiting for the whole frame to be received. This architecture is faster because there is no time elapsed for waiting and analyzing the whole frame. If the destination port is busy, the switch will store the packet in its buffer.

Using store-and-forward, the switch receives the whole packet and analyzes it before transmitting it to its destination. One advantage of the store-and-forward architecture is that packets with errors are recognized immediately and they have no chance to

propagate through the network. Nowadays, the speed of both types is becoming comparable so the effect when choosing one over the other is vanishing. Hybrid switches combining both architectures are also available.

When two or more nodes are sending their frames to the same destination, the frame arriving first occupies the receiving port, the others have to queue. A frame can wait in the buffer until the other frames in front of it get transmitted. The delay due to this is not deterministic, since the sequence in which the packages arrive at the port is not predictable. If the buffer is full, packets are discarded. Since there is no mechanism for the sending node to know if its packet was delivered to the right destination or not, there is the need to a way to handle deterministic communications via Ethernet switches without violating their deadlines.

Queuing occurs also at the transmitting port of the switch, when a station has packets to be sent to different destinations. Sometimes a packet at the first place of a queue may block the others in case its destination receiving port is busy. The receiving port for the other packets might be free, but they have to wait until the first packet leaves the queue. Such a situation is called head-of-line (HOL) blocking. The throughput of such an architecture is limited to 58%. But, by using complex queuing disciplines, it can be increased.

## 2.6 FlexRay and Real Time traffic

Since FlexRay uses the Flexible Time Division Multiple Access (FTDMA) algorithm for controlling the access to the medium, a FTDMA cycle needs to be generated for a certain traffic assumption. Therefore it is necessary to distinguish between Hard Real Time (HRT) and Soft Real Time (SRT) traffic and their place in the FTDMA cycle. Furthermore the traffic allocation to the different time slots and nodes has to be done with respect to the different traffic demands each node has, in order to achieve highest efficiency.

### *2.6.1 Real Time traffic*

FlexRay uses in its FTDMA cycle a static part for traffic which has to meet hard real time demands, regarding fault tolerance and deadlines. A flexible part is used for event-triggered soft real time traffic without the strict hard real time constraints. Thus, a differentiation between HRT and SRT has had to be made.

Real time can have many different meanings, but in computer science language it is used in context with computing systems that are able to deal with and use information, considering certain time constraints.

The two different kinds of real time discussed in this paper are soft real time and hard real time. The difference between them is described by the graph below:

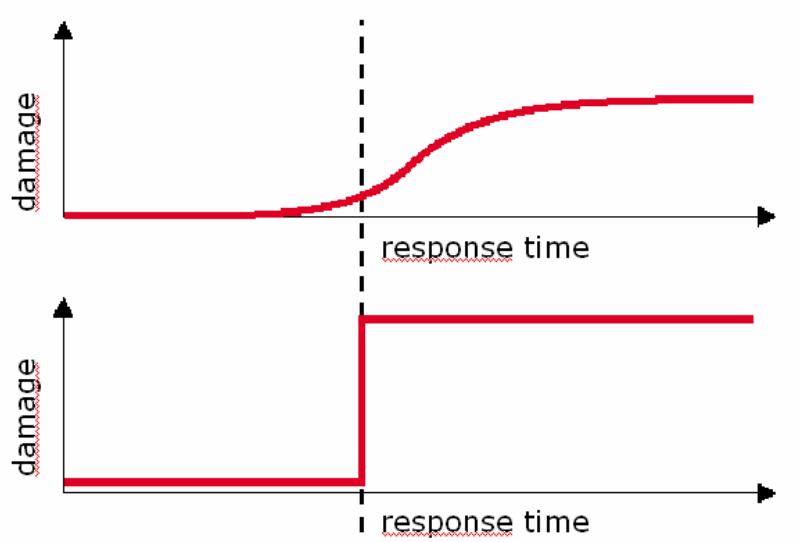



Figure 2.4 Damage caused from soft/hard real time tasks

Soft real time is characterized by the ability to perform a task, which on average, is executed according to the desired schedule. A typical application for soft real time traffic would for example be live video and/or audio streaming. A violation of the constraint rules usually goes at the expense of transmission quality. Unlike hard real time tasks, the damage arising due to constraint violations is not as profound. The system can continue with its operations without sustaining severe damage. Hard real time tasks, on the other hand, are characterized by the guaranteed timing and constraint adherence. Once the deadline is exceeded the damage occurs immediately. An example would be an X-by wire application within a vehicle, where delayed message arrival would lead to disastrous errors, like delayed braking or steering.

But the immediate occurrence of damage for delayed hard real time tasks compared to soft real time tasks, does not necessarily give information on the amount of time elapsing. Hard real time does not automatically mean that the time limit for a task is shorter than for a soft real time task. A HRT deadline can easily be longer than a SRT deadline. The difference is mainly described by the amount of damage caused by delayed messages and the importance of meeting certain constraints.

### *2.6.2 HRT and SRT in FTDMA*

Due to the strict deadline constraints of hard real time tasks, the static TDMA part of the FlexRay protocol is used to deal with that kind of traffic. The TDMA cycle is designed statically, with respect to the amount of traffic, the deadlines and the destinations for each node.

An optimum would of course be, if the case of two or more nodes sending to one destination at the same time never occurred. This would mean that situations where packets have to be queued, either at the transmitter or at the receiver, would never appear.

A worst case scenario on the other hand would be, all nodes constantly trying to send to the same destination simultaneously. This would inevitably lead to transmission delays and packet losses due to overcrowded queues.

An example of both, an optimal solution and a worst case solution, as well as realistic examples are explained this paper.
For soft real time tasks the flexible part of the FlexRay protocol is used, where medium access is handled event-triggered instead of using a static TDMA cycle. This brings the benefit, that not the whole traffic behaviour has to be known in advance, and any traffic which crops up can be transmitted as well within a reasonable amount of time. The minislot principle, used for the event triggered part, is explained in further detail in the previous FlexRay chapter.

# 3. RELATED WORK

Related work and different approaches to apply a TDMA algorithm on switched Ethernet networks are discussed in this section of the paper.

## 3.1 TTCAN over switched Ethernet

In [*12*] a way is presented how to use a Time Triggered Controller Area Network (TTCAN) over mixed Controller Area Network (CAN) / Switched Ethernet. In this approach several CAN networks are connected wire bridges with a switched Ethernet network. The aim is to reach time triggered behaviour on the whole network, respecting the time triggered behaviour of the TTCAN media access method of CAN, also when one part connected to the switched Ethernet is not a CAN, and consequently reach real time behaviour. Unfortunately this approach has not been simulated and evaluated yet.

## 3.2 TD-TWDMA over switched Ethernet

[*18*] presents an approach for a real time protocol for a fibre optical star network, using Time Deterministic Time and Wavelength Division Multiple Access (TD-TWDMA) as media access method. Wave Division Multiplex (WDM) is used to get multiple Gb/s channels, and Time Division Media Access (TDMA) regulates the access to each channel by dividing them into time slot cycles. To each node a certain amount of time slots is allocated for messages which have to be delivered within certain boundaries, or in real time respectively. If a node has no use for these time slots, it can free them and make the medium accessible for best effort messages. A slot allocation algorithm is used to change the TDMA scheme according to the demands of the different nodes.

With this solution dynamic real time demands can be met within very predictable latency. Also dead line guarantees are possible, as well as efficient bandwidth utilisation.

The disadvantage of the fibre optic WDM star network is that optical devices and the medium are quiet expensive.

## 3.3 ProfiNet

ProfiNet [36], [37] is an open standard for industrial Ethernet which supports TCP/IP and IT protocols. It uses TDMA to achieve real time capability in Ethernet networks. The three versions of ProfiNet try to address the demands of real time communication in industrial automation. The first version addresses the non-real time domain. The second one comes up with two communication channels, one for none-real time and one for soft real time (SRT) communication. The latest version tries to replace the SRT solution with isochronous real time (IRT) solution, because of the requirements in motion control applications with very small cycle times. However, for this purpose hardware support is mandatory, and hence it is dependent on real time ASIC (application specific integrated circuit).

### 3.4 FlexRay / CAN

Chapter 2.3.1 and 2.3.3 are dealing with the specifications of the protocols FlexRay and CAN in further detail. The disadvantage with those two standards is that they only reach a performance up to 10 Mbit/s, what is not always sufficient for the constantly increasing bandwidth requirements.

# 4. METHODOLOGY

The basic idea of combining approaches of different networks is to gain additional options and increase performance. Since the aim of this work is to combine the advantages of switched Ethernet and FlexRay, the combination of high speed performance and determinism is the goal.

Referring to the previous background chapters with regard to FlexRay, switched Ethernet and real time traffic, the following chapters explain the Media Access Control (MAC) protocol and picture several practical examples.

## 4.1 MAC protocol

In order to combine the high speed performance of switched Ethernet with the predictability and the determinism of FlexRay, a time-triggered schedule for the Ethernet part needs to be generated. Since switched Ethernet uses full duplex links to connect the nodes, and due to its media access method, a data reception- and transmission time schedule for each node is generated instead of one single schedule, valid for the whole network (as with FlexRay).

Therefore first of all a certain traffic assumption is necessary for the purpose of knowing the traffic behaviour before system- and time schedule design. A traffic assumption is easily made by presuming a certain amount of nodes, sending a defined amount of frames to appointed destinations at dedicated timeslots. The most important values therefore are pure rate, capacity and dead line. The pure rate gives information on how frequent a node is supposed to send data to the certain destination. The capacity states the amount of data, and the dead line value is the maximal interval of timeslots until the packet has to be delivered (further explained, when used in the examples).

Based on the traffic assumption a media access control algorithm can be generated. Media access is controlled by a time schedule, what makes a TDMA cycle necessary. As mentioned before, due to the full duplex connection of each node with the Ethernet switch, a TDMA cycle for reception and transmission for each node is necessary.

The TDMA cycle is designed statically, with respect to the amount of traffic, the deadlines and the destinations for each node. The next chapters only deal with the time-triggered TDMA part of the FlexRay FTDMA cycle, since only that part is hard real time capable and more interesting for this project. Furthermore is synchronisation assumed to be already assured accurately with zero skew tolerance in each example.

There are many different possibilities of generating a TDMA cycle in various ways. Therefore it is important to take the traffic distribution for the different nodes into account and to design an efficient TDMA algorithm. Possibilities of validating a generated TDMA algorithm are either by simulating the whole network or by calculation of the utilization.

## 4.2 Basic examples

The following chapters should give examples on an optimal, a worst case and realistic examples of traffic assumptions and explain the development of the corresponding static TDMA cycle. This chapter deals solely with the static part of FlexRay and thus with hard real time traffic, which is the most important one for the purpose of this thesis.

### *4.2.1 Optimal solution for a static TDMA cycle*

An optimal solution for a traffic assumption and the development of a TDMA cycle would be, as mentioned before, that the case of two or more nodes transmitting data to the same destination simultaneously never occurs.

For simplicity reasons the amount of nodes is limited to four, what makes the explanation of the transmitting and receiving cycles easier. The network architecture, shown in figure 4.1, consists of four nodes connected wire a full duplex link to an Ethernet switch. Each, the receiving port rx and the transmitting port tx, have queues of a certain length. The aim of this best case solution is not to use these queues. This pattern is usually called many-to-many communication and should help the general understanding of the TDMA algorithm design.

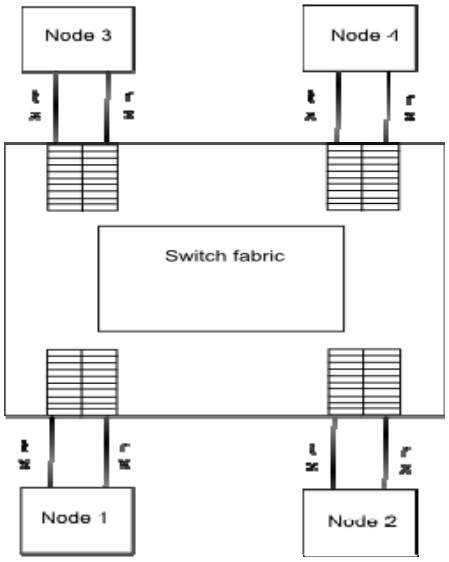



Figure 4.1 Network Architecture

As shown in figure 4.2, in this example it is assumed that each node has to send a certain, not yet specified, amount of data to each other node. The traffic is considered to be hard real time traffic.

| Sending Node | Receiving Node | Traffic classification |
|---|---|---|
| Node 1 | Node 2, Node 3, Node 4 | HRT |
| Node 2 | Node 1, Node 3, Node 4 | HRT |
| Node 3 | Node 1, Node 2, Node 4 | HRT |
| Node 4 | Node 1, Node 2, Node 3 | HRT |

Figure 4.2 Traffic assumptions for an optimal solution

The following figure 4.3 gives an overview of what each node sends to which destination within which timeslot in detail. Thus it appears for example that node 1 sends its data to node 2 in the fist and sixth timeslot, to node 3 in the second and fourth timeslot and so on.

| Transmitter cycle Node 1 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 2 | Node 3 | Node 4 | Node 3 | Node 4 | Node 2 |

| Transmitter cycle Node 2 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 1 | Node 4 | Node 3 | Node 1 | Node 3 | Node 4 |

| Transmitter cycle Node 3 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 4 | Node 1 | Node 2 | Node 4 | Node 2 | Node 1 |

| Transmitter cycle Node 4 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 3 | Node 2 | Node 1 | Node 2 | Node 1 | Node 3 |

Figure 4.3 Transmitter cycles of node 1, 2, 3 and 4 for optimal solution

In order to find an optimal solution without queuing, it is necessary that no node receives data from more than one other than the node within the same timeslot. Examining the second timeslot for example shows that node 1 sends to node 3, node 2 sends to node 4, node 3 sends to node1 and node 4 sends to node 2. Since the nodes have full duplex connectivity to the Ethernet switch, receiving and transmitting data simultaneously does not pose a threat to timely transmission.

As a result of the transmitting cycles of each node, in the next step the receiving cycle, shown in figure 4.4 can be generated.

| Receiving cycle | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| Node 1 <- | Node 2 | Node 3 | Node 4 | Node 2 | Node 4 | Node 3 |
| Node 2 <- | Node 1 | Node 4 | Node 3 | Node 4 | Node 3 | Node 1 |
| Node 3 <- | Node 4 | Node 1 | Node 2 | Node 1 | Node 2 | Node 4 |
| Node 4 <- | Node 3 | Node 2 | Node 1 | Node 3 | Node 1 | Node 2 |

Figure 4.4 Receiving cycle of node 1, 2, 3 and 4

As shown in figure 4.4, each node receives and sends data only from one other node at a time. Taking a look at node 3 for example shows that in the first timeslot it receives data from node 4, in the second timeslot from node 1, in the third timeslot from node 2, in the fourth timeslot from node 1, in the fifth timeslot from node 2 and in the sixth timeslot from node 4. At the same time node 3 sends data in the timeslots one to six to nodes 4, 1, 2, 4, 2 and 1.
Unfortunately in real transfer mode systems it is not always possible to find a perfect solution, since the traffic demands are usually not as homogeneous.

### *4.2.2 Worst case scenario for a static TDMA cycle*

Since traffic distribution in communication networks can be very inhomogeneous, this part of the paper should give an example for a worst case scenario in a TDMA cycle. Using the same traffic assumption as in the last example (Figure 4.2) again, each node is able to send its data to each other node.
To visualise the more detailed traffic information, again the transmitter cycles are shown in figure 4.5 below.

| Transmitter cycle Node 1 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 2 | Node 3 | Node 3 | Node 3 | Node 4 | Node 4 |

| Transmitter cycle Node 2 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 1 | Node 1 | Node 3 | Node 3 | Node 4 | Node 4 |

| Transmitter cycle Node 3 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 2 | Node 1 | Node 1 | Node 2 | Node 4 | Node 4 |

| Transmitter cycle Node 4 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| sending to node -> | Node 2 | Node 1 | Node 3 | Node 3 | Node 1 | Node 2 |

Figure 4.5 Transmitter cycles of nodes 1, 2, 3, and 4 for the worst case scenario

The figure above shows that the traffic distribution is not very homogeneous, since in every time slot one particular node receives data from two or more other nodes. In timeslot one for example node 2 sends its data to node 1 and receives data from node 1, 3 and 4. Since it is only possible for one node to process data from one other node within one timeslot, the packets from the two other nodes are queued and processed in the next two timeslots.

Having a look at timeslots three and four in the figure above shows, that in each of the two timeslots node three receives data from nodes 1, 2 and 4. Consequently the queue builds up until the packets are discarded.

| 1st queuing cycle node 3 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| packages to proceed | | X | XXX | XXXXX | XXXX | XXX |

| 2nd queuing cycle node 3 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| packages to proceed | XX | XX | XXXX | XXXXXX | XXXXX | XXXX |

| 3rd queuing cycle node 3 | | | | | | |
|---|---|---|---|---|---|---|
| time slot -> | 1 | 2 | 3 | 4 | 5 | 6 |
| packages in queue | XXX | XXX | XXXXX | XXXXXXX | XXXXXX | XXXXX |

Figure 4.6 Increasing queue for each TDMA cycle from node 3

Figure 4.6 shows how the amount of data which node three has to process increases within each TDMA cycle. The amount of packages to handle for the node is meant to be at the beginning of each timeslot. Since the sequence of arrival of the packages from the different nodes at the recipient is hardly predictable, packages are simply substituted by an “X” instead of personating them with their transmitter node.

In timeslot two the node receives data from node one, which does not address any problems since it can be processed within the same timeslot and does not have to be queued. Within the next timeslot data from all three other nodes arrive, whereas two of them consequently have to be queued. In timeslot four, node three again receives data from all other nodes and the queue increases to four data packages at the end of timeslot four. Within the last two timeslots of the first TDMA cycle, and the first timeslot in the next cycle, no data is addressed to node three which makes it possible to decrease the packets in the queue.

The fact that in the next timeslot again data from node one is received, and that there is still one queued packet remaining from the first cycle, makes it obvious that the amount of data in the queue increases with each additional TDMA cycle.
As a result packets become:

- discarded from the queue, depending on the used queue handling mechanism, and thus do not arrive at the receiver
- or arrive delayed without meeting the strict hard real time constraints

To avoid more traffic than the network can handle it is important to check the utilization at system design time. An example for a realistic TDMA algorithm is shown in the next chapter

## 4.3 Realistic traffic assumption

### *4.3.1 Architecture*

In order to increase severity and to be able to make traffic assumptions more realistic and complex, the amount of nodes was increased to six for the next architecture. The six nodes are, as in the examples before (figure 4.1), connected as a full duplex link to an Ethernet switch. Each, the receiving port rx and the transmitting port tx, have queues of a certain length.

### *4.3.2 Traffic assumption*

The figures below give an overview on the traffic distribution for the architecture described in the previous chapter. The pure rate gives information on how frequent a node is supposed to send data to the certain destination. The amount of packets which have to be sent within that certain time period is stated in the capacity column. A pure rate value of five and a capacity of three would for example describe three packets which have to be sent to their destination within every five time slots. A deadline value of seven for this example would then mean that the very last point in time for finishing the transmission would be after seven time slots. Thus, the pure rate remains the same, what means that within the (in this case) next five timeslots after the first five, the same amount of data has to be sent again.
The respective pure rates, capacities and dead lines for each assumed data transmission between the different nodes are stated in the figures below. As in the examples before, the assumed traffic for each transmission is hard real time.

| Traffic assumption, node 1 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 3 | 4 | 2 | 6 |
| Node 6 | 10 | 4 | 12 |

| Traffic assumption, node 2 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 4 | 1 | 4 |
| Node 6 | 20 | 6 | 40 |

| Traffic assumption, node 3 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 10 | 2 | 10 |
| Node 2 | 10 | 2 | 10 |
| Node 4 | 10 | 2 | 10 |
| Node 5 | 10 | 2 | 10 |
| Node 6 | 10 | 1 | 11 |

| Traffic assumption, node 4 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 3 | 20 | 6 | 20 |

| Traffic assumption, node 5 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 2 | 20 | 1 | 40 |
| Node 3 | 20 | 3 | 20 |
| Node 6 | 10 | 2 | 10 |

| Traffic assumption, node 6 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 1 | 20 | 11 | 20 |

Figure 4.7 Traffic assumption for a realistic example

The sequence in which the receiving nodes are listed in the respective tables does not give any information on how the packets are received or queued. They are just listed in numerical order, due to overview issues.

*4.3.3 TDMA cycle generation*

Based on the traffic assumptions for each node from the previous chapter, the TDMA cycle can be generated. One main aspect therefore is the relation between the amount of nodes and the amount of timeslots per cycle. Since a node does not send data to itself but only to maximal (in this example) five other nodes, the amount of timeslots can be calculated as follows:

$$k \bullet (2 \bullet n - 2)$$

k… any real number, n… amount of nodes

Since this example is based on six nodes, the outcome of that easy calculation is that "every" multiple of ten timeslots is reasonable. In order not to make the TDMA

algorithm too compact and complex, but still having the flexibility to make changes with hindsight, an amount of twenty timeslots is used for each cycle.

| Rec. /Trans. Cycle | | TS 1-20 | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Receiving Node | Sending Node | | | | | | | | | | | | | | | | | | | | |
| Node 1 | Node 2 | | | x | | | | x | | | | x | | | | x | | | | x | |
| | Node 3 | | | | X | | | | x | | | | x | | | | x | | | | |
| | Node 6 | x | x | | X | | x | | x | x | | | | x | x | | x | x | | | x |
| Node 2 | Node 3 | | x | x | | | | | | | | | x | x | | | | | | | |
| | Node 5 | | | | | | | | | | x | | | | | | | | | | |
| Node 3 | Node 1 | | x | x | | | x | x | | | x | x | | | x | x | | | x | x | |
| | Node 4 | x | | | X | x | | | | | | | x | x | | | x | | | | |
| | Node 5 | | | | | | | | x | x | | | | | | | | x | | | |
| Node 4 | Node 3 | x | | | | x | | | | | | x | | | | x | | | | | |
| Node 5 | Node 3 | | | | | | x | x | | | | | | | | | x | x | | | |
| Node 6 | Node 1 | | | | X | x | | | x | x | | | | | x | x | | | x | x | |
| | Node 2 | | | | | | x | x | | | | | | | | | x | x | | | |
| | Node 3 | | | | | | | | | | x | | | | | | | | | | x |
| | Node 5 | | x | x | | | | | | | | | x | x | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |

| Rec. /Trans. Cycle | | TS 21 – 40 | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 21 | . | . | . | 25 | . | . | . | . | 30 | . | . | . | . | 35 | . | . | . | . | 40 |
| Receiving Node | Sending Node | | | | | | | | | | | | | | | | | | | | |
| Node 1 | Node 2 | | | x | | | | x | | | | x | | | | x | | | | x | |
| | Node 3 | | | | x | | | | x | | | | x | | | | x | | | | |
| | Node 6 | x | x | | x | | x | | x | x | | | | x | x | | x | x | | | x |
| Node 2 | Node 3 | | x | x | | | | | | | | | x | x | | | | | | | |
| | Node 5 | | | | | | | | | | x | | | | | | | | | | |
| Node 3 | Node 1 | | x | x | | | x | x | | | x | x | | | x | x | | | x | x | |
| | Node 4 | x | | | x | x | | | | | | | x | x | | | x | | | | |
| | Node 5 | | | | | | | | x | x | | | | | | | | x | | | |
| Node 4 | Node 3 | x | | | | x | | | | | | x | | | | x | | | | | |
| Node 5 | Node 3 | | | | | | x | x | | | | | | | | | x | x | | | |
| Node 6 | Node 1 | | | | x | x | | | x | x | | | | | x | x | | | x | x | |
| | Node 2 | | | | | | x | x | | | | | | | | | x | x | | | |
| | Node 3 | | | | | | | | | | x | | | | | | | | | | x |
| | Node 5 | | x | x | | | | | | | | | x | x | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |

Figure 4.8 TDMA Receiving and Transmission Cycle from timeslot 1 – 40

The sequence in which the sending and receiving nodes are listed in the figure above, again does not give any information on how the packets are received or queued. They are just listed in numerical order, due to overview issues.

The TDMA cycle shown in figure 4.8 is of course just one out of several possibilities. There are a number of different TDMA cycle solutions for the same traffic assumption.
The TDMA algorithm in this example was generated to emphasize the main challenges and their solutions.
One aim is to create a cycle, where nodes interfere in their transmissions as little as possible, although some of them might have a high amount of data transmitted within short time periods (low pure rate values and high capacity values). For example, since node one sends to node three two packets within four timeslots, node one should avoid sending to node six at the same time because of queuing issues and a very short deadline to node three. Thus it should be avoided for nodes to send or receive two or more packets at the same time, and nevertheless to meet the deadlines if simultaneous transmissions are inevitable.

### *4.3.4 Receiver and transmitter port queuing cycles*

In this example simultaneous transmissions are consciously used in order to include queuing and make it more realistic for further simulations. As shown in the next figure, the queued packages in this example do not always meet the pure rate. However they are always within the deadline frame, what is satisfying for HRT constraints.

| Receiving queue cycle Node 1 | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | .. | 4 | .. | 8 | 9 | .. | 16 | 17 | 18 | .. | 24 | .. | 28 | 29 | .. | 36 | 37 | .. | 40 |
| Node | | | N3 | | N3 | + N6 | | N3 | + N6 | | | N3 | | N3 | + N6 | | N3 | + N6 | | |
| | | | N6 | | N6 | | | N6 | | | | N6 | | N6 | | | N6 | | | |

Figure 4.9 Queuing cycle for node one at the receiving port

The figure above shows the queuing cycle at the receiving port for node one, which is the only node where receiving queues occur. The sequence in which the nodes are listed in the figures does again, not give any information on the order they are queued in, and are just chosen randomly. This is the same case for receiving and transmitting port queues.
In timeslot eight for example nodes three and six send their data simultaneously to node one, whereas node one can only process data from one of the two at a time. The packet from the second node has to wait in the queue until the next timeslot to get dealt with. At the same time, in timeslot five, another packet from node six arrives and gets added to the queue.
After the same principle the queues at the transmitting ports for each node can be figured out, as shown in the next graphs.

| Transmitting queue cycle Node 1 | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | .. | 11 | .. | 14 | 15 | .. | 18 | 19 | .. | 31 | .. | 34 | 35 | .. | 38 | 39 | .. |
| Node | | | | | N6 | N6 | | N6 | N6 | | | | N6 | N6 | | N6 | N6 | |
| | | | | | N3 | N3 | | N3 | N3 | | | | N3 | N3 | | N3 | N3 | |

Figure 4.10 Queuing cycle for node one at the transmitting port

| Transmitting queue cycle Node 2 | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | .. | 26 | 27 | .. |
| Node | | | | | | | N1 | | | | | | | | | | N1 | |
| | | | | | | | N6 | | | | | | | | | | N6 | |

Figure 4.11 Queuing cycle for node two at the transmitting port

| Transmitting queue cycle Node 3 | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | .. | 10 | .. | 15 | 16 | 17 | .. | 20 | .. | 26 | 27 | 28 | .. |
| Node | | | | | | | | | | N1 | + | | | | N1 | + | | |
| | | | | | | | | | | N5 | N5 | | | | N5 | N5 | | |

Figure 4.12 Queuing cycle for node three at the transmitting port

Figures 4.10 to 4.12 show the queuing cycles at the transmission ports for nodes one, two and three, which are the only ones where such queues occur. The most important thing is that all messages arrive at their destination before their deadline expires. This example TDMA algorithm was constipated in order to fulfil this goal theoretically.

### *4.3.5 Discussion of the realistic example*

As mentioned before in this paper, there are a variety of possibilities to generate a TDMA algorithm for a certain traffic assumption. The aim of this thesis was to find a solution which is on the one hand realistic enough to be feasible, and on the other hand easy enough to make it variable, transparent, easy explainable and simulative. Therefore the traffic assumption had only partially very hard constraints regarding pure rate, capacity and dead line (e.g. node one to node three). Also part of the development was not trying to find a “perfect” solution. Furthermore it was more important not to avoid queuing in order to make it more realistic for the simulation part, since in bigger networks queues are not avoidable anyway. In case of queues a main interest was on the one hand not to violate dead lines, but, on the other hand, not to avoid exceeding the pure rate.

# 5. EXAMPLES AND CORRESPONDING UTILIZATION CALCULATION

## 5.1 More Examples

The previous examples show that the development of receiving- and transmitting cycles very much depends on the amount of nodes, the amount of timeslots per TDMA cycle, and on the traffic assumption. Thus, there is a variety of possible solutions of TDMA schedules for one and the same traffic assumption. To validate the method developed in this project, several examples with different architectures and traffic assumptions were created in order to observe the utilization of the whole algorithm. The utilization depends on the usage of the available time slots for transmitting and receiving. It is easily calculated by computing the ratio of used time slots to the total number of time slots for each node at the sending and receiving port. The utilization for the whole algorithm is calculated by summing up the sending – and receiving utilizations for each node and divide the value by the amount of nodes in the network.

In the following examples various traffic assumptions are made for a different number of nodes and the transmitting- and receiving cycle for each node is designed based on these assumptions. Starting with a rather easy example with four nodes, the complexity increases with each example up to an architecture with eight nodes. Also the amount of timeslots within one TDMA cycle is varied for each example.
At the end of each example the utilization is calculated in order to validate the algorithm. All of the following examples were designed in a way that all deadlines are met, as visible in the according sending- and receiving cycles.

Example 1: The first example consists of four nodes and uses 18 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Traffic assumption, node 1 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 2 | 8 | 2 | 8 |
| Node 3 | 12 | 2 | 18 |
| Node 4 | 18 | 4 | 18 |

| Traffic assumption, node 2 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 18 | 5 | 18 |
| Node 3 | 9 | 4 | 9 |

| Traffic assumption, node 3 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 2 | 18 | 7 | 18 |
| Node 4 | 6 | 3 | 6 |

| Traffic assumption, node 4 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 3 | 18 | 5 | 18 |

Figure 5.1 Traffic assumption for example 1.

| Receiving Node | Sending Node | First cycle TS 1-18 | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Node 1 | Node 2 | | | | | x | x | x | x | | | | | | x | | | | |
| Node 2 | Node 1 | x | X | | | | | | | x | x | | | | | | | x | x |
| | Node 3 | | | | x | x | x | | | | x | x | x | | | | x | | |
| Node 3 | Node 1 | | | x | x | | | | | | | | | | | x | x | | |
| | Node 2 | x | X | x | x | | | | | | x | x | x | x | | | | | |
| | Node 4 | | | | | | | x | x | x | | | | | x | | | x | |
| Node 4 | Node 1 | | | | | x | x | x | x | | | | | | | | | | |
| | Node 3 | x | X | x | | | | x | x | x | | | | x | x | x | | | |

| Receiving Node | Sending Node | Second cycle TS 1-18 | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Node 1 | Node 2 | | | | | x | x | x | x | | | | | | x | | | | |
| Node 2 | Node 1 | x | X | | | | | | | x | x | | | | | | | x | x |
| | Node 3 | | | | x | x | x | | | | x | x | x | | | | x | | |
| Node 3 | Node 1 | | | x | x | | | | | | | | | | | x | x | | |
| | Node 2 | x | X | x | x | | | | | | x | x | x | x | | | | | |
| | Node 4 | | | | | | | x | x | x | | | | | x | | | x | |
| Node 4 | Node 1 | | | | | x | x | x | x | | | | | | | | | | |
| | Node 3 | x | X | x | | | | x | x | x | | | | x | x | x | | | |

Figure 5.2 TDMA receiving and transmitting cycle for the first 36 time slots.

Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 5/18 = 0.277
For node 2 = 13/18 = 0.722
For node 3 = 17/18 = 0.944
For node 4 = 13/18 = 0.722

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 14/18 = 0.778
For node 2 = 14/18 = 0.778

For node 3 = 16/18 = 0.889
For node 4 = 5/18 = 0.277

Utilization of the whole algorithm

Sending utilization = (0.778 + 0.778 + 0.889 + 0.277)/4 = 0.681
Receiving utilization = (0.227 + 0.772 + 0.944 + 0.722)/4 = 0.681

Example 2: The second example consists of five nodes and uses 16 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Traffic assumption, node 1 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 4 | 6 | 3 | 6 |
| Node 5 | 16 | 5 | 16 |

| Traffic assumption, node 2 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 1 | 16 | 2 | 16 |
| Node 3 | 12 | 4 | 16 |
| Node 4 | 16 | 2 | 24 |

| Traffic assumption, node 3 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 4 | 2 | 8 |

| Traffic assumption, node 4 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 2 | 8 | 3 | 8 |
| Node 3 | 8 | 1 | 8 |
| Node 5 | 10 | 2 | 10 |

| Traffic assumption, node 5 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 2 | 10 | 4 | 16 |
| Node 3 | 8 | 2 | 16 |
| Node 4 | 8 | 2 | 8 |

Figure 5.3 Traffic assumption for example 2

| Rec. /Trans. Cycle | | First Cycle TS 1-16 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Receiving Node | Sending Node | | | | | | | | | | | | | | | | |
| Node 1 | Node 2 | | | | | | | x | x | | | | | | | | |
| | Node 3 | x | x | | | x | x | | | x | x | | | x | x | | |
| Node 2 | Node 4 | | | | x | x | | x | x | | x | | x | | | | |
| | Node 5 | x | x | X | | | x | | | x | | | | x | x | x | |
| Node 3 | Node 2 | x | x | X | x | | | | | | | | | x | x | x | x |
| | Node 4 | | | | | | x | | | | | x | | | | | |
| | Node 5 | | | | | x | | | x | | x | | x | | | | |
| Node 4 | Node 1 | x | x | X | | | | | x | x | | | x | x | x | x | |
| | Node 2 | | | | | x | x | | | | | | | | | | |
| | Node 5 | | | | x | | | x | | | | x | | | | | x |
| Node 5 | Node 1 | | | | x | x | x | | | | x | x | | | | | |
| | Node 4 | x | x | | | | | | | | | | | | x | | x |

| Rec. /Trans. Cycle | | Second Cycle TS 1-16 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Receiving Node | Sending Node | | | | | | | | | | | | | | | | |
| Node 1 | Node 2 | | | | | | | x | x | | | | | | | | |
| | Node 3 | x | x | | | x | x | | | x | x | | | x | x | | |
| Node 2 | Node 4 | | | | x | x | | x | x | | x | | x | | | | |
| | Node 5 | x | x | X | | | x | | | x | | | | x | x | x | |
| Node 3 | Node 2 | x | x | X | x | | | | | | | | | x | x | x | x |
| | Node 4 | | | | | | x | | | | | x | | | | | |
| | Node 5 | | | | | x | | | x | | x | | x | | | | |
| Node 4 | Node 1 | x | x | X | | | | | x | x | | | x | x | x | x | |
| | Node 2 | | | | | x | x | | | | | | | | | | |
| | Node 5 | | | | x | | | x | | | | x | | | | | x |
| Node 5 | Node 1 | | | | x | x | x | | | | x | x | | | | | |
| | Node 4 | x | x | | | | | | | | | | | | x | | x |

Figure 5.4 TDMA receiving and transmitting cycle for the first 32 time slots.

Utilization of the time slots by each node for sending and receiving is calculated below.

Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 10/16 = 0.625
For node 2 = 14/16 = 0.875
For node 3 = 14/16 = 0.875
For node 4 = 15/16 = 0.937
For node 5 = 9/16 = 0.562

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 14/16 = 0.875
For node 2 = 12/16 = 0.750
For node 3 = 8/16 = 0.500
For node 4 = 12/16 = 0.750
For node 5 = 16/16 = 1.000

Utilization of the whole algorithm

Receiving utilization = (0.625 + 0.875 + 0.875 + 0.937 + 0.562)/5 = 0.775
Sending utilization = (0.875 + 0.750 + 0.500 + 0.750 + 1.000)/5 = 0.775

Example 3: The third example also consists of five nodes and uses 24 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Traffic assumption, node 1 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 2 | 12 | 1 | 12 |
| Node 3 | 8 | 1 | 8 |
| Node 4 | 6 | 2 | 6 |
| Node 5 | 24 | 5 | 24 |

| Traffic assumption, node 2 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 4 | 24 | 11 | 24 |
| Node 5 | 12 | 2 | 12 |

| Traffic assumption, node 3 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 12 | 4 | 24 |
| Node 2 | 24 | 10 | 24 |

| Traffic assumption, node 4 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 3 | 24 | 8 | 24 |

| Traffic assumption, node 5 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 8 | 4 | 8 |

Figure 5.5 Traffic assumption for example 3

| Receiving Node | Sending Node | First cycle TS 1-24 | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Node 1 | Node 3 | | | | | x | X | x | x | | | | | x | x | x | x | | | | | | | | |
| | Node 5 | x | x | x | x | | | | | x | x | x | x | | | | | x | x | x | x | | | | |
| Node 2 | Node 1 | x | | | | | | | | | | | | x | | | | | | | | | | | |
| | Node 3 | | x | x | x | | | | | x | x | x | x | | | | | x | x | x | | | | | |
| Node 3 | Node 1 | | x | | | | | | | | x | | | | | | | | x | | | | | | |
| | Node 4 | x | | x | x | x | | | | x | | x | x | x | | | | x | | x | x | x | | | |
| Node 4 | Node 1 | | | x | x | | | | | x | | x | | | | x | x | | | | | x | x | | |
| | Node 2 | | | | | x | X | x | x | | x | | x | | | | | x | x | x | x | | | x | |
| Node 5 | Node 1 | | | | | x | X | x | x | | | | x | | | | | | | | | | | | |
| | Node 2 | x | x | x | x | | | | | | | | | x | x | x | x | | | | | | | | |

| Receiving Node | Sending Node | Second cycle TS 1-24 | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Node 1 | Node 3 | | | | | x | X | x | x | | | | | x | x | x | x | | | | | | | | |
| | Node 5 | x | x | x | x | | | | | x | x | x | x | | | | | x | x | x | x | | | | |
| Node 2 | Node 1 | x | | | | | | | | | | | | x | | | | | | | | | | | |
| | Node 3 | | x | x | x | | | | | x | x | x | x | | | | | x | x | x | | | | | |
| Node 3 | Node 1 | | x | | | | | | | | x | | | | | | | | x | | | | | | |
| | Node 4 | x | | x | x | x | | | | x | | x | x | x | | | | x | | x | x | x | | | |
| Node 4 | Node 1 | | | x | x | | | | | x | | x | | | | x | x | | | | | x | x | | |
| | Node 2 | | | | | x | X | x | x | | x | | x | | | | | x | x | x | x | | | x | |
| Node 5 | Node 1 | | | | | x | X | x | x | | | | x | | | | | | | | | | | | |
| | Node 2 | x | x | x | x | | | | | | | | | x | x | x | x | | | | | | | | |

Figure 5.6 TDMA receiving and transmitting cycle for the first 48 time slots.

Utilization calculation

Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 20/24 = 0.833
For node 2 = 12/24 = 0.500
For node 3 = 15/24 = 0.625
For node 4 = 19/24 = 0.791
For node 5 = 13/24 = 0.541

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 18/24 = 0.750
For node 2 = 19/24 = 0.791
For node 3 = 18/24 = 0.750
For node 4 = 12/24 = 0.500
For node 5 = 12/24 = 0.500

Utilization of the whole algorithm
Receiving utilization = (0.833 + 0.500 + 0.625 + 0.791 + 0.541)/5 = 0.658
Sending utilization = (0.750 + 0.791 + 0.750 + 0.500 + 0.500)/5 = 0.658

Example 4: The fourth example consists of six nodes and uses 10 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Traffic assumption, node 1 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 4 | 10 | 4 | 10 |
| Node 5 | 10 | 2 | 20 |
| Node 6 | 6 | 1 | 8 |

| Traffic assumption, node 2 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 10 | 3 | 20 |
| Node 3 | 10 | 4 | 10 |

| Traffic assumption, node 3 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 5 | 10 | 4 | 10 |
| Node 6 | 8 | 2 | 16 |

| Traffic assumption, node 4 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 2 | 10 | 4 | 20 |
| Node 3 | 8 | 1 | 8 |
| Node 5 | 10 | 2 | 10 |

| Traffic assumption, node 5 sending | | | |
|---|---|---|---|
| Receiving Nodes | Pure rate | Capacity | Dead line |
| Node 1 | 4 | 1 | 10 |

| Traffic assumption, node 6 sending | | | |
|---|---|---|---|
| Receiving Node | Pure rate | Capacity | Dead line |
| Node 1 | 10 | 1 | 10 |
| Node 2 | 10 | 2 | 10 |
| Node 3 | 10 | 2 | 20 |
| Node 4 | 10 | 1 | 20 |
| Node 5 | 10 | 1 | 10 |

Figure 5.7 Traffic assumption for example 4

| Rec. /Trans. Cycle | | First cycle TS 1-10 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Receiving Node | Sending Node | | | | | | | | | | |
| Node 1 | Node 2 | x | x | x | | | | | | | |
| | Node 5 | | | | x | | | | x | | |
| | Node 6 | | | | | x | x | x | | | |
| Node 2 | Node 4 | | | x | x | x | x | | | | |
| | Node 6 | x | x | | | | | | | | |
| Node 3 | Node 2 | | | | x | x | | | | | |
| | Node 4 | x | | | | | | | | x | |
| | Node 6 | | x | x | | | | | | | |
| Node 4 | Node 1 | | | x | x | x | x | | | | |
| | Node 6 | | x | | | | | | | | |
| Node 5 | Node 1 | x | x | | | | | | | | |
| | Node 3 | | | x | x | x | x | | | | |
| | Node 4 | | | | | | | | | x | x |
| | Node 6 | | | | | | | | x | | |
| Node 6 | Node 1 | | x | | | | | | x | | |
| | Node 3 | x | x | | | | | | | x | x |

| Rec. /Trans. Cycle | | Second cycle TS 1-10 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Receiving Node | Sending Node | | | | | | | | | | |
| Node 1 | Node 2 | x | x | x | | | | | | | |
| | Node 5 | | | | x | | | | x | | |
| | Node 6 | | | | | x | x | x | | | |
| Node 2 | Node 4 | | | x | x | x | x | | | | |
| | Node 6 | x | x | | | | | | | | |
| Node 3 | Node 2 | | | | x | x | | | | | |
| | Node 4 | x | | | | | | | | x | |
| | Node 6 | | x | x | | | | | | | |
| Node 4 | Node 1 | | | x | x | x | x | | | | |
| | Node 6 | | x | | | | | | | | |
| Node 5 | Node 1 | x | x | | | | | | | | |
| | Node 3 | | | x | x | x | x | | | | |
| | Node 4 | | | | | | | | | x | x |
| | Node 6 | | | | | | | | x | | |
| Node 6 | Node 1 | | x | | | | | | x | | |
| | Node 3 | x | x | | | | | | | x | x |

Figure 5.8 TDMA receiving and transmitting cycle for the first 20 time slots.

| Receiving queue cycle Node 3 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| From nodes | | | | N2<br>N6 | N2<br>N6 | | | | | |

| Receiving queue cycle Node 5 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| From nodes | N1<br>N6 | | | | | | | | | |

| Receiving queue cycle Node 6 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| From nodes | | N1<br>N3 | | | | | | | | |

Figure 5.9 Queuing cycle for nodes 3, 5 and 6 at their receiving port.

| Transmitting queue cycle Node 1 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| To nodes | | N5<br>N6 | | | | | | | | |

| Transmitting queue cycle Node 4 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| To nodes | | | | | | | | | N3<br>N5 | |

| Transmitting queue cycle Node 6 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| To nodes | N2<br>N4<br>N5 | N2<br>N3<br>N4 | | | N1<br>N3 | | N1<br>N5 | | | |

Figure 5.10 Queuing cycle for nodes 1, 4 and 6 at their transmitting port.

Since all these nodes with queued packets have utilization of their time slots less than 100% (shown below in the calculations), they use the remaining free slots for the queued packets without missing any deadline.

Utilization calculation
Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 9/10 = 0.9
For node 2 = 6/10 = 0.6
For node 3 = 6/10 = 0.6
For node 4 = 5/10 = 0.5
For node 5 = 9/10 = 0.9
For node 6 = 6/10 = 0.6

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 8/10 = 0.8
For node 2 = 7/10 = 0.7
For node 3 = 8/10 = 0.8
For node 4 = 8/10 = 0.8
For node 5 = 3/10 = 0.3
For node 6 = 7/10 = 0.7

Utilization of the whole algorithm
Receiving utilization = (0.9 + 0.6 + 0.6 + 0.5 + 0.9 + 0.6)/6 = 0.683
Sending utilization = (0.8 + 0.7 + 0.8 + 0.8 + 0.3 + 0.7)/6 = 0.683

Example 5: The fifth example consists of seven nodes and uses 24 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Sending Node | Receiving Node | Pure rate | Capacity | Deadline |
|---|---|---|---|---|
| Node 1 | Node 2 | 8 | 3 | 8 |
| | Node 4 | 16 | 5 | 16 |
| | Node 6 | 24 | 3 | 24 |
| Node 2 | Node 3 | 6 | 2 | 6 |
| | Node 5 | 12 | 3 | 24 |
| | Node 7 | 24 | 6 | 24 |
| Node 3 | Node 1 | 4 | 2 | 8 |
| | Node 5 | 24 | 3 | 24 |
| | Node 7 | 12 | 4 | 24 |
| Node 4 | Node 2 | 24 | 4 | 24 |
| | Node 6 | 12 | 6 | 12 |
| Node 5 | Node 1 | 24 | 4 | 24 |
| | Node 4 | 16 | 4 | 24 |
| Node 6 | Node 3 | 24 | 8 | 24 |
| Node 7 | Node 1 | 24 | 1 | 24 |
| | Node 2 | 24 | 2 | 24 |
| | Node 3 | 24 | 1 | 24 |
| | Node 4 | 24 | 1 | 24 |
| | Node 5 | 24 | 4 | 24 |
| | Node 6 | 24 | 1 | 24 |

Figure 5.11 Traffic assumption for example 5

| Receiving Node | Sending Node | First cycle TS 1-24 | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Node 1 | Node 3 | x | x | | | x | x | | | x | x | | | x | x | | | x | x | | | x | x | | |
| | Node 5 | | | | | | | x | x | | | x | x | | | | | | | | | | | | |
| | Node 7 | | | x | | | | | | | | | | | | | | | | | | | | | |
| Node 2 | Node 1 | x | x | x | | | | | | x | x | x | | | | | | x | x | x | | | | | |
| | Node 4 | | | | | | | x | x | | | | | | | | | | | | x | x | | | |
| | Node 7 | | | | x | x | | | | | | | | | | | | | | | | | | | |
| Node 3 | Node 2 | x | x | | | | | x | x | | | | | x | x | | | | | x | x | | | | |
| | Node 6 | | | x | x | x | x | | | x | x | x | x | | | | | | | | | | | | |
| | Node 7 | | | | | | | | | | | | | | | x | | | | | | | | | |
| Node 4 | Node 1 | | | | x | x | x | x | x | | | | | | | | | | | | x | x | x | x | x |
| | Node 5 | x | x | x | | | | | | x | | | | | | | x | x | x | x | | | | | |
| | Node 7 | | | | | | | | | | x | | | | | | | | | | | | | | |
| Node 5 | Node 2 | | | x | x | x | | | | | | | | | | x | x | x | | | | | | | |
| | Node 3 | | | | | | | | | | | x | x | | | | | | | | | | | x | |
| | Node 7 | x | x | | | | x | x | | | | | | | | | | | | | | | | | |
| Node 6 | Node 1 | | | | | | | | | | | | x | x | x | | | | | | | | | | |
| | Node 4 | x | x | x | x | x | x | | | | | | | | | x | x | x | x | x | | | x | | |
| | Node 7 | | | | | | | | x | | | | | | | | | | | | | | | | |
| Node 7 | Node 2 | | | | | | x | | | x | x | x | x | | | | | | x | | | | | | |
| | Node 3 | | | x | x | | | x | x | | | | | | | x | x | | | x | x | | | | |

| Receiving Node | Sending Node | Second cycle TS 1-24 | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Node 1 | Node 3 | x | x | | | x | x | | | x | x | | | x | x | | | x | x | | | x | x | | |
| | Node 5 | | | | | | | x | x | | | x | x | | | | | | | | | | | | |
| | Node 7 | | | x | | | | | | | | | | | | | | | | | | | | | |
| Node 2 | Node 1 | x | x | x | | | | | | x | x | x | | | | | | x | x | x | | | | | |
| | Node 4 | | | | | | | x | x | | | | | | | | | | | | x | x | | | |
| | Node 7 | | | | x | x | | | | | | | | | | | | | | | | | | | |
| Node 3 | Node 2 | x | x | | | | | x | x | | | | | x | x | | | | | x | x | | | | |
| | Node 6 | | | x | x | x | x | | | x | x | x | x | | | | | | | | | | | | |
| | Node 7 | | | | | | | | | | | | | | | x | | | | | | | | | |
| Node 4 | Node 1 | | | | x | x | x | x | x | | | | | | | | | | | | x | x | x | x | x |
| | Node 5 | x | x | x | | | | | | x | | | | | | | x | x | x | x | | | | | |
| | Node 7 | | | | | | | | | | x | | | | | | | | | | | | | | |
| Node 5 | Node 2 | | | x | x | x | | | | | | | | | | x | x | x | | | | | | | |
| | Node 3 | | | | | | | | | | | x | x | | | | | | | | | | | x | |
| | Node 7 | x | x | | | | x | x | | | | | | | | | | | | | | | | | |
| Node 6 | Node 1 | | | | | | | | | | | | x | x | x | | | | | | | | | | |
| | Node 4 | x | x | x | x | x | x | | | | | | | | | x | x | x | x | x | | | x | | |
| | Node 7 | | | | | | | | x | | | | | | | | | | | | | | | | |
| Node 7 | Node 2 | | | | | | x | | | x | x | x | x | | | | | | x | | | | | | |
| | Node 3 | | | x | x | | | x | x | | | | | | | x | x | | | x | x | | | | |

Figure 5.12 TDMA receiving and transmitting cycle for the first 48 time slots.

Utilization calculation

Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 17/24 = 0.708
For node 2 = 15/24 = 0.625
For node 3 = 17/24 = 0.708
For node 4 = 19/24 = 0.792
For node 5 = 13/24 = 0.542
For node 6 = 16/24 = 0.667
For node 7 = 14/24 = 0.583

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 22/24 = 0.917
For node 2 = 20/24 = 0.833
For node 3 = 23/24 = 0.958
For node 4 = 16/24 = 0.667
For node 5 = 12/24 = 0.500
For node 6 = 8/24 = 0.333
For node 7 = 10/24 = 0.417

Utilization of the whole algorithm
Receiving utilization = (0.708 + 0.625 + 0.708 + 0.792 + 0.542 + 0.667 + 0.583)/7
= 0.661
Sending utilization = (0.917 + 0.833 + 0.958 + 0.667 + 0.500 + 0.333 + 0.417)/7
= 0.661

Example 6: The fourth example consists of eight nodes and uses 28 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below.

| Sending node | Receiving node | Pure rate | Capacity | Deadline |
|---|---|---|---|---|
| Node 1 | Node 2 | 28 | 5 | 28 |
| | Node 3 | 28 | 4 | 28 |
| | Node 4 | 28 | 5 | 28 |
| | Node 5 | 28 | 3 | 28 |
| | Node 6 | 28 | 3 | 28 |
| | Node 7 | 28 | 2 | 28 |
| | Node 8 | 28 | 2 | 28 |
| Node 2 | Node 1 | 10 | 3 | 10 |
| | Node 3 | 14 | 3 | 28 |
| | Node 5 | 28 | 5 | 28 |
| Node 3 | Node 5 | 7 | 3 | 7 |
| | Node 7 | 14 | 5 | 14 |
| | Node 8 | 28 | 2 | 28 |
| Node 4 | Node 2 | 14 | 5 | 14 |
| | Node 6 | 14 | 6 | 14 |
| | Node 8 | 28 | 4 | 28 |
| Node 5 | Node 1 | 28 | 10 | 28 |
| | Node 3 | 28 | 6 | 28 |
| | Node 4 | 28 | 5 | 28 |
| Node 6 | Node 2 | 10 | 3 | 10 |
| | Node 4 | 28 | 10 | 28 |
| | Node 5 | 28 | 4 | 28 |
| Node 7 | Node 1 | 28 | 6 | 28 |
| | Node 6 | 28 | 7 | 28 |
| | Node 8 | 10 | 4 | 28 |
| Node 8 | Node 7 | 7 | 3 | 7 |

Figure 5.13 Traffic assumption for example 6

| Receiving node | Sending node | First cycle TS 1-28 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| Node 1 | Node 2 | x | x | x | | | | | | | | x | x | x | | | | | | | | x | x | x | | | | | |
| | Node 5 | | | | x | x | x | | | | x | | | | x | x | x | | | | | | | | x | x | | x | |
| | Node 7 | | | | | | | | | x | | | | | | | | x | x | x | x | | | | | | x | | |
| Node 2 | Node 1 | | | | | | | | | x | x | | | | | | | | | x | | | | | | | | x | x |
| | Node 4 | | | | x | x | x | x | x | | | | | | | x | x | x | x | x | | | | | | | | | |
| | Node 6 | x | x | x | | | | | | | | x | x | x | | | | | | | | x | x | x | | | | | |
| Node 3 | Node 1 | | | | | | | x | x | | | x | | | | | | | | | x | | | | | | | | |
| | Node 2 | | | | x | x | x | | | | | | | | | | | | x | x | x | | | | | | | | |
| | Node 5 | x | x | x | | | | | | | | | | x | | | | x | | | | | x | | | | | | |
| Node 4 | Node 1 | | | x | | | | | | | | | x | x | | | | | | | | x | x | | | | | | |
| | Node 5 | | | | | | | x | x | x | | | | | | | | | | x | x | | | | | | | | |
| | Node 6 | | | | x | x | x | | | | x | | | | | x | x | x | x | | | | | | x | x | | | |
| Node 5 | Node 1 | | | | x | x | x | | | | | | | | | | | | | | | | | | | | | | |
| | Node 2 | | | | | | | x | | | | | | | | | | | | | | | | | | x | x | x | x |
| | Node 3 | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | |
| | Node 6 | | | | | | | x | | | | | | | x | | | | | x | x | | | | | | | | |
| Node 6 | Node 1 | | | | | | | | | | | | | | x | x | | x | | | | | | | | | | | |
| | Node 4 | x | x | x | | | | | | x | x | x | | | | | | | | | x | x | x | x | x | x | | | |
| | Node 7 | | | | | x | x | x | x | | | | | | | | x | | | | | | | | | x | | x | |
| Node 7 | Node 1 | | | | | | | | | | | | | | | | | | | | | | | | | x | x | | |
| | Node 3 | | | | x | x | x | x | | | | x | | | | | | | x | x | x | x | | | | x | | | |
| | Node 8 | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | |
| Node 8 | Node 1 | | | | | | | | | | | | | | | | x | | x | | | | | | | | | | |
| | Node 3 | | | | | | | | | | | | x | x | | | | | | | | | | | | | | | |
| | Node 4 | | | | | | | | | | | | | | x | | | | | | | | | | | | x | x | x |
| | Node 7 | x | x | x | x | | | | | | x | x | | | x | x | | | | | | x | x | x | x | | | | |

| Receiving node | Sending node | Second cycle TS 1-28 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| Node 1 | Node 2 | x | x | x | | | | | | | | x | x | x | | | | | | | | x | x | x | | | | | |
| | Node 5 | | | | x | x | x | | | | x | | | | x | x | x | | | | | | | | x | x | | x | |
| | Node 7 | | | | | | | | | x | | | | | | | | x | x | x | x | | | | | | x | | |
| Node 2 | Node 1 | | | | | | | | | x | x | | | | | | | | | x | | | | | | | | x | x |
| | Node 4 | | | | x | x | x | x | x | | | | | | | x | x | x | x | x | | | | | | | | | |
| | Node 6 | x | x | x | | | | | | | | x | x | x | | | | | | | | x | x | x | | | | | |
| Node 3 | Node 1 | | | | | | | x | x | | | x | | | | | | | | | x | | | | | | | | |
| | Node 2 | | | | x | x | x | | | | | | | | | | | | x | x | x | | | | | | | | |
| | Node 5 | x | x | x | | | | | | | | | | x | | | | x | | | | | x | | | | | | |
| Node 4 | Node 1 | | | x | | | | | | | | | x | x | | | | | | | | x | x | | | | | | |
| | Node 5 | | | | | | | x | x | x | | | | | | | | | | x | x | | | | | | | | |
| | Node 6 | | | | x | x | x | | | | x | | | | | x | x | x | x | | | | | | x | x | | | |
| Node 5 | Node 1 | | | | x | x | x | | | | | | | | | | | | | | | | | | | | | | |
| | Node 2 | | | | | | | x | | | | | | | | | | | | | | | | | | x | x | x | x |
| | Node 3 | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | |
| | Node 6 | | | | | | | x | | | | | | | x | | | | | x | x | | | | | | | | |
| Node 6 | Node 1 | | | | | | | | | | | | | | x | x | | x | | | | | | | | | | | |
| | Node 4 | x | x | x | | | | | | x | x | x | | | | | | | | | x | x | x | x | x | x | | | |
| | Node 7 | | | | | x | x | x | x | | | | | | | | x | | | | | | | | | x | | x | |
| Node 7 | Node 1 | | | | | | | | | | | | | | | | | | | | | | | | | x | x | | |
| | Node 3 | | | | x | x | x | x | | | | x | | | | | | | x | x | x | x | | | | x | | | |
| | Node 8 | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | | x | x | x | | | | |
| Node 8 | Node 1 | | | | | | | | | | | | | | | | x | | x | | | | | | | | | | |
| | Node 3 | | | | | | | | | | | | x | x | | | | | | | | | | | | | | | |
| | Node 4 | | | | | | | | | | | | | | x | | | | | | | | | | | | x | x | x |
| | Node 7 | x | x | x | x | | | | | | x | x | | | x | x | | | | | | x | x | x | x | | | | |

Figure 5.14 TDMA receiving and transmitting cycle for the first 56 time slots.

Queuing occurs at the receivers of node 2, 3,5,6,7 and 8. The cycle is shown for each of these nodes.

| Rx queuing for node 2 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | | | | | | | | | | | | | N1<br>N4 | | | | | | | | | |

| Rx queuing for node 3 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | | | | | | | | | | | | | | N1<br>N2 | | | | | | | | |

| Rx queuing for node 5 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | N2<br>N6 | | | | | | | | | | | | | | | | | | | | | |

| Rx queuing for node 6 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | | | | | | | | | | | | | | | | | | | N4<br>N7 | | | |

| Rx queuing for node 7 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | | | | | | | | | | | | | | | | | | | N1<br>N3 | | | |

| Rx queuing for node 8 | Time slots | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| From nodes | | | | | | | | | | | | | | N4<br>N7 | | | | | | | | | | | | | | |

Figure 5.15 Queuing cycle for nodes 2, 3, 5, 6, 7, and 8 at their receiving port.

The utilization calculation for this complex traffic is as follows.

Receiving port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 25/28 = 0.893
For node 2 = 24/28 = 0.857
For node 3 = 16/28 = 0.571
For node 4 = 20/28 = 0.714
For node 5 = 24/28 = 0.857
For node 6 = 22/28 = 0.786
For node 7 = 24/28 = 0.857
For node 8 = 20/28 = 0.714

Sending port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 24/28 = 0.857
For node 2 = 20/28 = 0.714
For node 3 = 24/28 = 0.857
For node 4 = 26/28 = 0.929
For node 5 = 21/28 = 0.750
For node 6 = 23/28 = 0.821
For node 7 = 25/28 = 0.893
For node 8 = 12/28 = 0.429

Utilization of the whole algorithm

Receiving util. = (0.893 + 0.857 + 0.571 + 0.714 + 0.857 + 0.786 + 0.857 + 0.714)/8
= 0.781
Sending util. = (0.857 + 0.714 + 0.857 + 0.929 + 0.750 + 0.821 + 0.893 + 0.429)/8
= 0.781

Example 7: The seventh example also consists of eight nodes and uses 14 time slots per TDMA cycle. The traffic assumption and the sending/receiving schedule are shown in the figures below. For this example, we have assumed a simple traffic, not complex as the previous one.

| Sending Node | Receiving Node | Pure rate | Capacity | Deadline |
|---|---|---|---|---|
| Node 1 | Node 4 | 7 | 3 | 7 |
| | Node 8 | 12 | 2 | 14 |
| Node 2 | Node 1 | 14 | 4 | 14 |
| | Node 7 | 9 | 4 | 14 |
| Node 3 | Node 6 | 6 | 2 | 6 |
| | Node 8 | 14 | 6 | 14 |
| Node 4 | Node 2 | 14 | 8 | 14 |
| | Node 5 | 14 | 2 | 14 |
| Node 5 | Node 4 | 14 | 3 | 14 |
| | Node 7 | 14 | 4 | 14 |
| Node 6 | Node 3 | 7 | 4 | 7 |
| | Node 5 | 14 | 4 | 14 |
| Node 7 | Node 2 | 7 | 2 | 7 |
| | Node 6 | 14 | 6 | 14 |
| Node 8 | Node 1 | 6 | 2 | 6 |
| | Node 3 | 14 | 3 | 14 |

Figure 5.16 Traffic assumption for example 7.

| Receiving node | Sending node | First cycle TS 1-14 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Node 1 | Node 2 | | | | | x | x | | | x | | | | | x |
| | Node 8 | x | x | | | | | x | x | | | | x | x | |
| Node 2 | Node 4 | | | x | x | x | x | x | | | x | x | x | | |
| | Node 7 | x | x | | | | | | x | x | | | | | |
| Node 3 | Node 6 | x | x | x | x | | | | x | x | x | x | | | |
| | Node 8 | | | | | x | x | | | | | | | | x |
| Node 4 | Node 1 | x | x | x | | | | | x | x | x | | | | |
| | Node 5 | | | | x | x | x | | | | | | | | |
| Node 5 | Node 4 | x | x | | | | | | | | | | | | |
| | Node 6 | | | | | x | x | x | | | | | x | | |
| Node 6 | Node 3 | x | x | | | | | x | x | | | | | x | x |
| | Node 7 | | | x | x | x | x | | | | | x | x | | |
| Node 7 | Node 2 | x | x | x | x | | | | | | x | x | x | x | |
| | Node 5 | | | | | | | x | x | x | | | | | x |
| Node 8 | Node 1 | | | | x | x | | | | | | | | x | x |
| | Node 3 | | | x | | | x | | | x | x | x | x | | |

| Rx node | Tx node | Second cycle TS 1-14 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Node 1 | Node 2 | | | | | x | x | | | x | | | | | x |
| | Node 8 | x | x | | | | | x | x | | | | x | x | |
| Node 2 | Node 4 | | | x | x | x | x | x | | | x | x | x | | |
| | Node 7 | x | x | | | | | | x | x | | | | | |
| Node 3 | Node 6 | x | x | x | x | | | | x | x | x | x | | | |
| | Node 8 | | | | | x | x | | | | | | | | x |
| Node 4 | Node 1 | x | x | x | | | | | x | x | x | | | | |
| | Node 5 | | | | x | x | x | | | | | | | | |
| Node 5 | Node 4 | x | x | | | | | | | | | | | | |
| | Node 6 | | | | | x | x | x | | | | | x | | |
| Node 6 | Node 3 | x | x | | | | | x | x | | | | | x | x |
| | Node 7 | | | x | x | x | x | | | | | x | x | | |
| Node 7 | Node 2 | x | x | x | x | | | | | | x | x | x | x | |
| | Node 5 | | | | | | | x | x | x | | | | | x |
| Node 8 | Node 1 | | | | x | x | | | | | | | | x | x |
| | Node 3 | | | x | | | x | | | x | x | x | x | | |

Figure 5.17 TDMA receiving and transmitting cycle for the first 28 time slots.

Utilization calculations

Sending port utilization
For node 1 = received packets in one cycle / number of time slots in one cycle
= 10/14 = 0.714
For node 2 = 12/14 = 0.857
For node 3 = 12/14 = 0.857
For node 4 = 10/14 = 0.714
For node 5 = 7/14 = 0.500
For node 6 = 12/14 = 0.857
For node 7 = 10/14 = 0.714
For node 8 = 9/14 = 0.643

Receiving port utilization
For node 1 = transmitted packets in one cycle / number of time slots in one cycle
= 10/14 = 0.714
For node 2 = 12/14 = 0.857
For node 3 = 11/14 = 0.785
For node 4 = 9/14 = 0.643
For node 5 = 6/14 = 0.429
For node 6 = 12/14 = 0.857
For node 7 = 12/14 = 0.857
For node 8 = 10/14 = 0.714

Utilization of the whole algorithm

Sending utiliz. = (0.714 + 0.857 + 0.857 + 0.714 + 0.500 + 0.857 + 0.714 + 0.643)/8
= 0.732
Receiving util. = (0.714 + 0.857 + 0.785 + 0.643 + 0.429 + 0.857 + 0.857 + 0.714)/8
= 0.732

The previous examples were created in order to show that the quality of the developed algorithm strongly depends on the way the TDMA schedule is designed. It shows that the method, developed in this project, makes a very high node and network utilization possible. As long as the amount of receptions and transmissions stays below or equals the amount of timeslots, the problem of deadlines being missed because of queuing is a minor threat and, as mentioned before, depends on the schedule design. The disadvantage is admittedly that the TDMA schedule design becomes quiet complicated with an increasing amount of nodes. Thus, the method is not very applicable for huge network environments with a large numbers of nodes.

The table below gives an overview on the different utilizations of the previous examples. The notation shows the sending node, the receiving node, the pure rate, the capacity and the deadline in the order of their appearance. For example, 43(18,5,18) means node 4 sending 5 packets to node 3 with pure rate of 18 and deadline 18.

5.2 Summarized Table

| No. of Nodes | No. of Time slots in one cycle | Traffic distribution | Utilization |
|---|---|---|---|
| 4 | 18 | 12(8,2,8), 13(12,2,18), 14(18,4,18), 21(18,6,18), 23(9,4,9), 32(18,7,18), 34(6,3,6), 43(18,5,18) | 0.681 |
| 5 | 16 | 14(6,3,6), 15(16,5,16), 21(16,2,16), 23(12,4,16), 24(12,4,16), 24(16,2,24), 31(4,2,8), 42(8,3,8), 43(8,1,8), 45(10,2,10), 52(10,4,16), 53(8,2,16), 54(8,2,8) | 0.775 |
| 5 | 24 | 12(12,1,12), 13(8,1,8), 14(6,2,6), 15(24,5,24), 24(24,11,24), 25(12,2,12), 31(12,4,24), 32(24,10,24), 43(24,8,24), 51(8,4,8) | 0.658 |
| 6 | 10 | 14(10,4,10), 15(10,2,20), 16(6,1,8), 21(10,3,20), 23(10,4,10), 35(10,4,10), 36(8,2,16), 42(10,4,20), 43(8,1,8), 45(10,2,10), 51(4,1,10), 61(10,1,10), 62(10,2,10), 63(10,2,20), 64(10,1,20), 65(10,1,10) | 0.683 |
| 7 | 24 | 12(8,3,8), 14(16,5,16), 16(24,3,24), 23(6,2,6), 25(12,3,24), 27(24,6,24), 31(4,2,8), 35(24,3,24), 37(12,4,24), 42(24,4,24), 46(12,6,12), 51(24,4,24), 54(16,4,24), 63(24,8,24), 71(24,1,24), 72(24,2,24), 73(24,1,24), 74(24,1,24), 75(24,4,24), 76(24,1,24) | 0.661 |
| 8 | 28 | 12(28,5,28), 13(28,4,28), 14(28,5,28), 15(28,3,28), 16(28,3,28), 17(28,2,28), 18(28,2,28), 21(10,3,10), 23(14,3,28), 25(28,5,28), 35(7,3,7), 37(14,5,14), 38(28,2,28), 42(14,5,14), 46(14,6,14), 48(28,4,28), 51(28,10,28), 53(28,6,28), 54(28,5,28), 62(10,3,10), 64(28,10,28), 65(28,4,28), 71(28,6,28), 76(28,7,28), 78(10,4,28), 87(7,3,7) | 0.781 |
| 8 | 14 | 14(7,3,7), 18(12,2,14), 21(14,4,14), 27(9,4,14), 36(6,2,6), 38(14,6,14), 42(14,8,14), 45(14,2,14), 54(14,3,14), 57(14,4,14), 63(7,4,7), 65(14,4,14), 72(7,2,7), 76(14,6,14), 81(6,2,6), 83(14,3,14) | 0.732 |

Table 5.1 Generalized form of examples 1 to 7.

# 6. SIMULATION

In order to validate the developed TDMA algorithm applied on switched Ethernet, either a simulation of such an architecture or a calculation of the utilization has to be made. This chapter deals with a simple simulation of a switched Ethernet environment with a certain amount of nodes, sending according to a predefined TDMA cycle.

## 6.1 Simulation environments

There is a variety of possible network simulators, with which it should be possible to implement a network architecture as it is demanded for the purpose of this project.

One possible simulator is NS2, which is a discrete event network simulator and is mainly used for the simulation of routing and multicast protocols. It supports multiple network protocols for wired and wireless networks and is, as a result of its General Public License, one of the most popular simulators nowadays. The reason why NS2 was not used for the simulation part of this project was that it is a very voluminous and powerful simulator, what includes quite a long period of vocational adjustment, including learning the used script language “Tool Command Language” (Tcl), which is used for the simulation interface.

Another network simulator relevant for the purpose of this work is “IT Guru” from OPNET Technologies. IT Guru simulates the behaviour of routers, switches, network protocols and applications. It is a commercial modelling and simulation tool to plan, analyze and optimize networks as well as for finding and preventing network errors. IT Guru was not used for the simulation part of this paper because it is first of all, as mentioned before, a commercial tool we did not have access to, and the main functionality of the simulator seems to primarily aim on simulating different already existing network protocols and architectures and thus seems not to be designed for implementing and simulating not yet existing media access protocols.

Another possibility of simulating the developed media access algorithm is by implementing a simulator independently. This could be done either in Matlab or in any other programming language. Unfortunately this task is very time consuming and needs a great deal of vocational adjustment in the respective programming language, which eliminates this alternative.

The environment which seemed to be most appropriate for simulating a TDMA algorithm applied on switched Ethernet, was the on Matlab based Simulink extension TRUE TIME. The decision of using TRUE TIME was based on the fact that it seemed to be powerful and flexible enough to meet our requirements, while being not too extensive and complicated and thus practicable without an overly long vocational adjustment period.

## 6.2 TRUE TIME simulator

Anton Cervin, Dan Henriksson, Martin Ohlin and Johan Eker from Lund University, Department of Automatic Control, developed the Matlab / Simulink- based simulator TRUE TIME, which is a Simulink block library extension and a collection of C++ functions with corresponding Matlab MEX interfaces. TRUE TIME was created in order to simulate distributed real-time control systems and simple models of networks and network protocols. It supports the mainly used

medium access protocols, Carrier Sense with Multiple Access and Collision Detection (CSMA/CD), Carrier Sense with Multiple Access and Collision Avoidance (CSMA/CA), Frequency Division Multiple Access (FDMA), Time Division Multiple Access (TDMA), token ring and switched Ethernet.

The TRUE TIME Matlab extension consists of four elements, the True Time Kernel, the True Time Network, the True Time Wireless network and the True Time Battery as shown in the figure below.

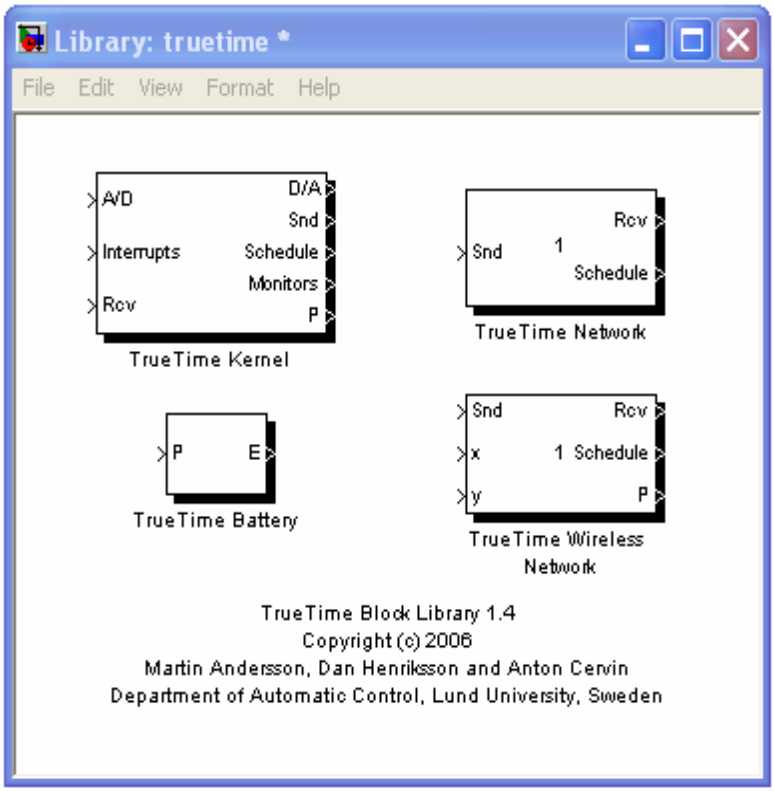



Figure 6.1 Simulink TRUE TIME library

The True Time Kernel element represents a node with an event driven real time kernel. Its function is to execute user defined tasks and interrupt handlers, which are defined by code functions, written by the user. These code functions can either be written in C++ or in Matlab m-files. The True Time Kernel is also equipped with Analog / Digital (A/D) and Digital / Analog (D/A) converters, which are less important for the aimed simulation of the TDMA algorithm.

The True Time Kernel is on the one hand configured by certain function block parameters, and on the other hand by an initialization script, written by the user. The function block parameters shown in figure …, give the opportunity to configure different constants for the simulation, and to indicate the name of the code function used for initializing. The “Init function argument” provides the possibility of adding an optional argument to the initialization script. An enabled “Battery” checkbox means that the kernel depends on a power source, what is not relevant for our simulation. With the clock drift parameter a time drift of the local time compared to the nominal

time can be generated. A value of 0.05 for example would mean that the local time runs 5 % faster than the actual simulation time. The clock offset is a constant time offset from the nominal time.

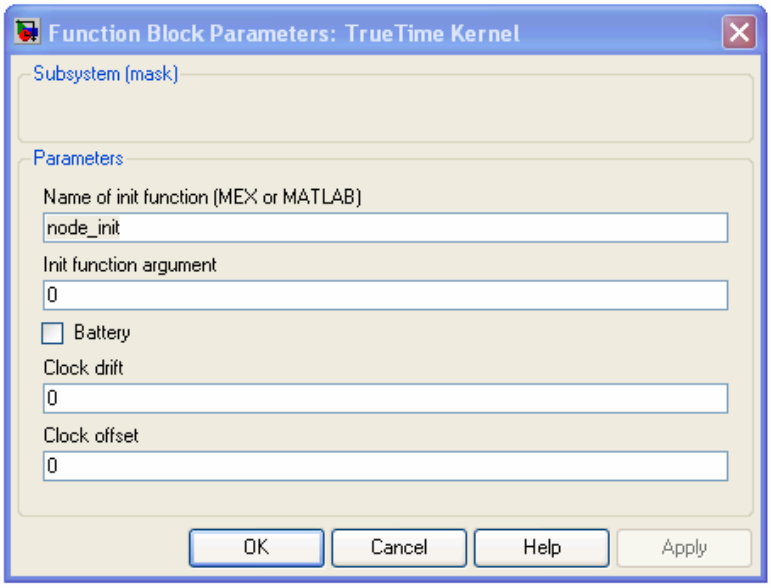



Figure 6.2 Function Block Parameters of the True Time Kernel

The event driven “True Time Network” element simulates the medium access method and packet based traffic between the different nodes. The supported network models are, as mentioned before, TDMA, Switched Ethernet, CSMA/CD, CSMA/CA, Round Robin and FDMA. A message transmitted through the network contains information about the sending and receiving node, user data, and real time attributes like priority or deadline and length of the message. Messages sent over the network are simulated to be temporally stored in a buffer at the receiving node.
The “True Time Network” parameters are configured using a “Function Block Parameters” interface. The initialization of the network takes place in the init script of the different nodes.

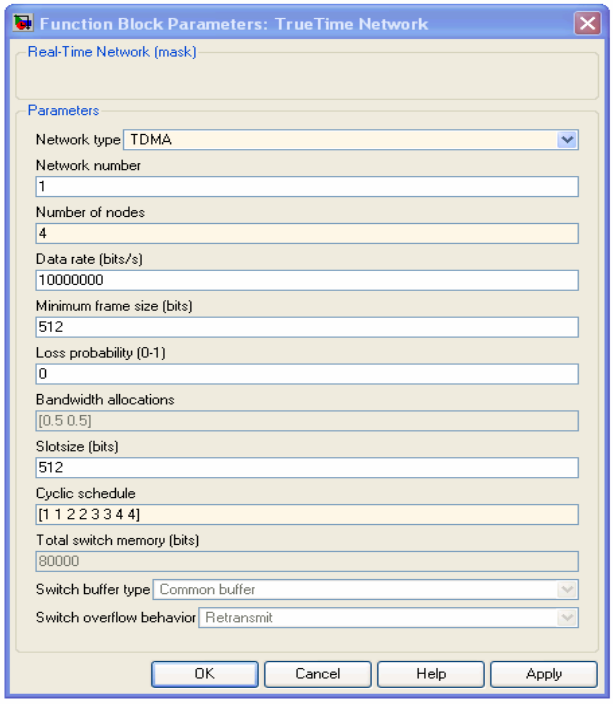



Figure 6.3 Function Block Parameters of the True Time Network

The function block parameters of the "True Time Network" are shown in the figure above, and include the following attributes:

- Network type: Offers the choice between the different network models. (CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin, FDMA, TDMA, Switched Ethernet)
- Network number: Indicates the number of the network block. If multiple network elements are used, each block has to have a unique number, starting from 1 and upwards.
- Number of nodes: Indicates the amount of nodes connected to the network element.
- Data rate (bit/s): To determine the speed of the network
- Minimum frame size (bits): States the minimum length of a frame. Frames shorter than the minimum frame size are padded to the desired size.
- Pre-processing delay(s): To specify a certain delay a message experiences by the sending end of the network interface.
- Post-processing delay(s): To specify a certain delay a message experiences by the receiving end of the network interface
- Loss probability: Indicates the probability of a message getting lost on the way to its destination.

- Bandwidth allocation: Is only used for the FDMA model and has to be maximal one, summing up all the shares for the sender nodes.

- Slot size (bits): States the size of a TDMA sending slot

- Cyclic schedule: Indicates the different node IDs and their assigned timeslots within one TDMA cycle. A zero forbids every node to send its data on the medium.

- Total Switch memory (bits): to configure the memory an Ethernet switch as available when the switched Ethernet network model is selected.

- Switch buffer type: offers the choice between "common buffer" and "symmetric output buffers"

- Switch overflow behaviour: Offers the choice between "retransmit" and "drop"

The "True Time Wireless Network" element offers approximately the same possibilities as the parameters discussed above, just for wireless networks and is thus not of interest in this project. Neither is the "True Time Battery", which represents a power supply element.

### 6.3 Modelling of a simulation example

For the simulation an example for the TDMA algorithm was developed which has a very high utilization at each node's receiving and transmitting port, in order to inspect the efficiency of the developed method. In the following example the architecture consists of four nodes, connected wire full duplex links with an Ethernet switch. One TDMA cycle is assumed to consist of twelve timeslots. The traffic assumption for this example is shown in the figure below.

| Traffic assumption | | | | |
|---|---|---|---|---|
| Sending Node | Receiving Node | Pure rate [Timeslots] | Capacity [packets] | Dead line [Timeslots] |
| Node 1 | Node 2 | 6 | 3 | 6 |
| Node 1 | Node 3 | 12 | 6 | 12 |
| Node 2 | Node 1 | 11 | 6 | 12 |
| Node 2 | Node 4 | 5 | 2 | 7 |
| Node 3 | Node 2 | 8 | 4 | 9 |
| Node 3 | Node 4 | 6 | 3 | 7 |
| Node 4 | Node 1 | 12 | 6 | 13 |
| Node 4 | Node 3 | 12 | 6 | 12 |

Figure 6.4 Traffic assumption for the simulated example

As explained in the previous chapter, based on this traffic assumption the transmitting and receiving cycles can be generated, which are shown in the following figures.

| Transmitter cycle Node 1 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS -> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| rec. Nodes -> | Node 2 | Node 3 | Node 2 | Node 3 | Node 2 | Node 3 | Node 2 | Node 3 | Node 2 | Node 3 | Node 2 | Node 3 |

| Transmitter cycle Node 2 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS -> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| rec. Nodes -> | Node 1 | Node 4 | Node 1 | Node 4 | Node 1 | Node 4 | Node 1 | Node 4 | Node 1 | Node 4 | Node 1 | Node 4 |

| Transmitter cycle Node 3 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS -> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| rec. Nodes -> | Node 4 | Node 2 | Node 4 | Node 2 | Node 4 | Node 2 | Node 4 | Node 2 | Node 4 | Node 2 | Node 4 | Node 2 |

| Transmitter cycle Node 4 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS -> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| rec. Nodes -> | Node 3 | Node 1 | Node 3 | Node 1 | Node 3 | Node 1 | Node 3 | Node 1 | Node 3 | Node 1 | Node 3 | Node 1 |

Figure 6.5 Transmitter cycles of node 1 – 4 for the simulated example

| Receiving / Transmitting Cycle | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timeslots -> | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Rec. Node | Send-Node | | | | | | | | | | | | |
| Node 1 | Node 2 | X | | X | | X | | X | | X | | X | |
| | Node 4 | | X | | X | | X | | X | | X | | X |
| Node 2 | Node 1 | X | | X | | X | | X | | X | | X | |
| | Node 3 | | X | | X | | X | | X | | X | | X |
| Node 3 | Node 1 | | X | | X | | X | | X | | X | | X |
| | Node 4 | X | | X | | X | | X | | X | | X | |
| Node 4 | Node 2 | | X | | X | | X | | X | | X | | X |
| | Node 3 | X | | X | | X | | X | | X | | X | |

Figure 6.6 Receiving / Transmitting Cycle of the simulated example

Again, the sequence in which the nodes are listed in the respective tables does not give any information on the order the packets arrive at their destination. They are just listed in numerical order, due to overview issues.

The figures above make it apparent that each node uses 100 % of its link capacity in both directions, sending and receiving. In order to avoid complication in the Matlab code and due to

some restrictions in the instruction set, the example is kept rather easy and without occurrence of simultaneous transmissions to or from one node.

## 6.4 Implementation of the simulation example

To make network simulations with TRUE TIME possible, it first of all has to be embedded in Matlab and Simulink. Once the TRUE TIME Simulink library is available and the respective paths refer to the right sources, the desired network can be designed with Simulink.

### *6.4.1 Network Architecture*

The general network architecture is shown in the figure below.

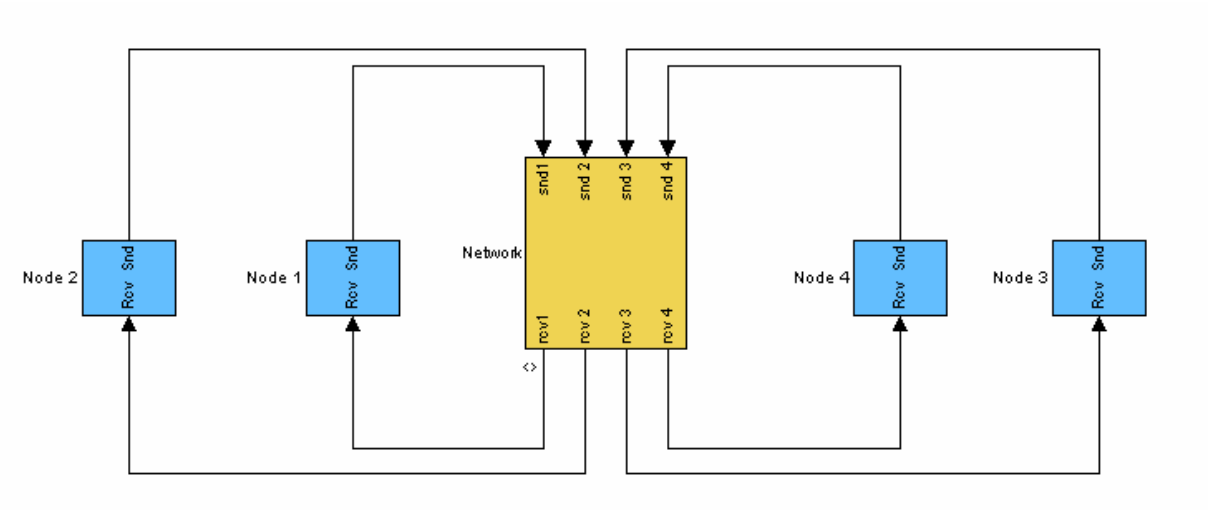



Figure 6.7 General Simulink network architecture of the simulation example

The nodes and the network element in figure … are just illustrative to show the general architecture of the network. The core elements like the True Time Kernel and the True Time Network element are in the subsystem of the respective elements, and are shown in the following figures.

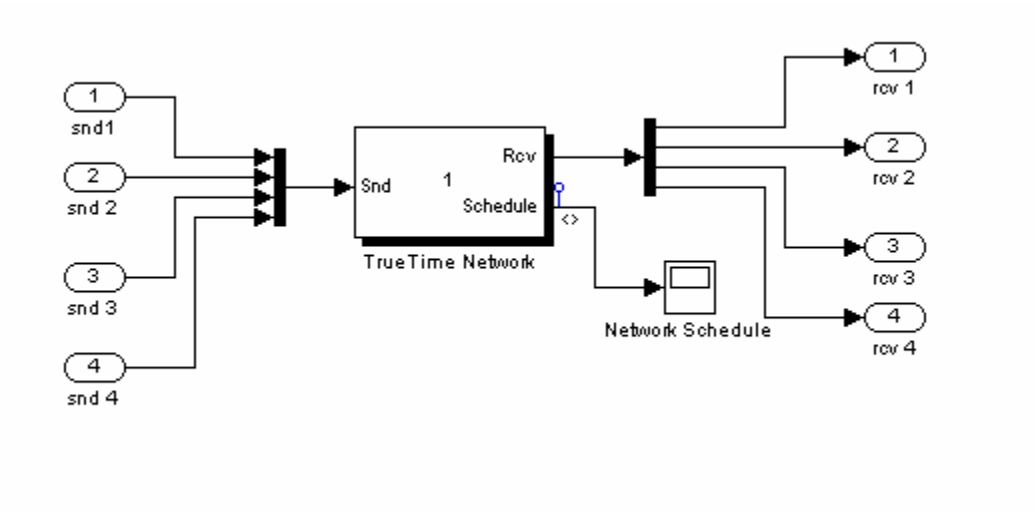



Figure 6.8 Subsystem of the network element

Figure 6.8 shows the subsystem of the network element from the general architecture in figure 6.7 It shows four input ports and four output ports connected wire a multiplexer and a de-multiplexer, respectively, with the True Time Network element at its sending and receiving port. A scope is used to visualize the traffic from the different nodes which is sent through the network, and is connected to the schedule port of the network element.

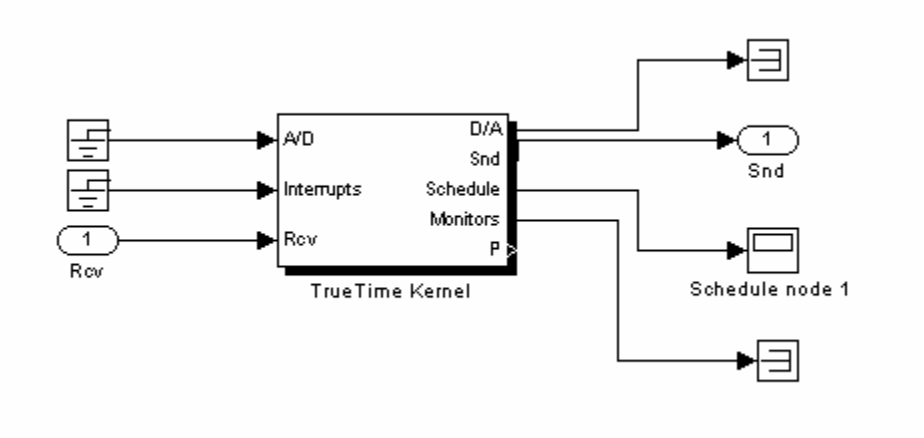



Figure 6.9 Subsystem of the node 1, illustrates the architecture of the end node

The subsystem of the nodes is kept rather simple, since only the receiving port and the sending port are used. For visualization of the traffic, again a scope is connected to the schedule port. All the other ports are unused and thus connected to “Ground”, in order not to get warning messages for not connected input ports.

### *6.4.2 Parameter configuration of the network architecture*

Following the main parameter settings of the simulation for the “True Time Network” element and the “True Time Kernel” are explained. Some of the values, like the data rate and the minimum frame size, change for different simulation scenarios but this will be mentioned a bit further in this paper.

The network type in the “True Time Network” function block parameters interface is set to “Switched Ethernet”, with the network number 1 and the value 4 for the number of nodes. The loss probability is set to 0, the switch buffer type is set to “Common buffer” and the switch overflow behaviour is “Drop”. Since the block is used in switched Ethernet mode, the parameters “Bandwidth allocations”, “Slotsize” and “Cyclic schedule” are not relevant.

The function block parameters “Init function argument “, “Clock drift” and “Clock offset” of the “True Time Kernel” are set to 0 each. The battery icon is disabled and the name of the different Matlab m- files, used for initializing the Kernel individually for each node, is stated in the first line. (For node 1 this would for example be “node1_init”)

The code for the initialization scripts for each node, the traffic generation and reception as well as for the interrupt handlers can be found in the appendix.

### 6.5 Simulation results

The graphs, shown in figure … were generated, using a data rate of 100.000.000 bits/s (100Mbit/s) and a minimum frame size of 100 Kbit. Frames get padded to the size of 100Kbit if they are appointed to be smaller in the traffic generation m- file. In this example each frame has the same length of 100 Kbit for every transmission. The very high value for the frame size was only used due to visibility issues, because for very small frame sizes the traffic just appears as a peak in the graph.

As an experiment, and to give a comparison between the different media access methods, the figures below do not show the traffic model of an Ethernet switch. CSMA/CD (Ethernet) was chosen instead of switched Ethernet at the “True Time Network” function block parameters. The two following figures show exactly the same, only the scaling is different in the second figure to give a more detailed view on the graph.

A high signal in the following graphs means sending or executing, a medium signal means waiting (e.g. for media access), and a low signal means idle.

Since the utilization of the simulated example is very high (100 %), figure 6.10 and 6.11 show that this amount of traffic is not manageable for shared medium Ethernet with CSMA/CD media access algorithm. A low level of the graph means that the node is idle, medium level that the node is waiting for media access, and a high level that the node is sending. The network schedule shows that packages miss their deadlines, since the bus is permanently occupied from other nodes. Node four for example has to wait for twelve sending periods until it gets access to the medium. (0.18 – 0.3)

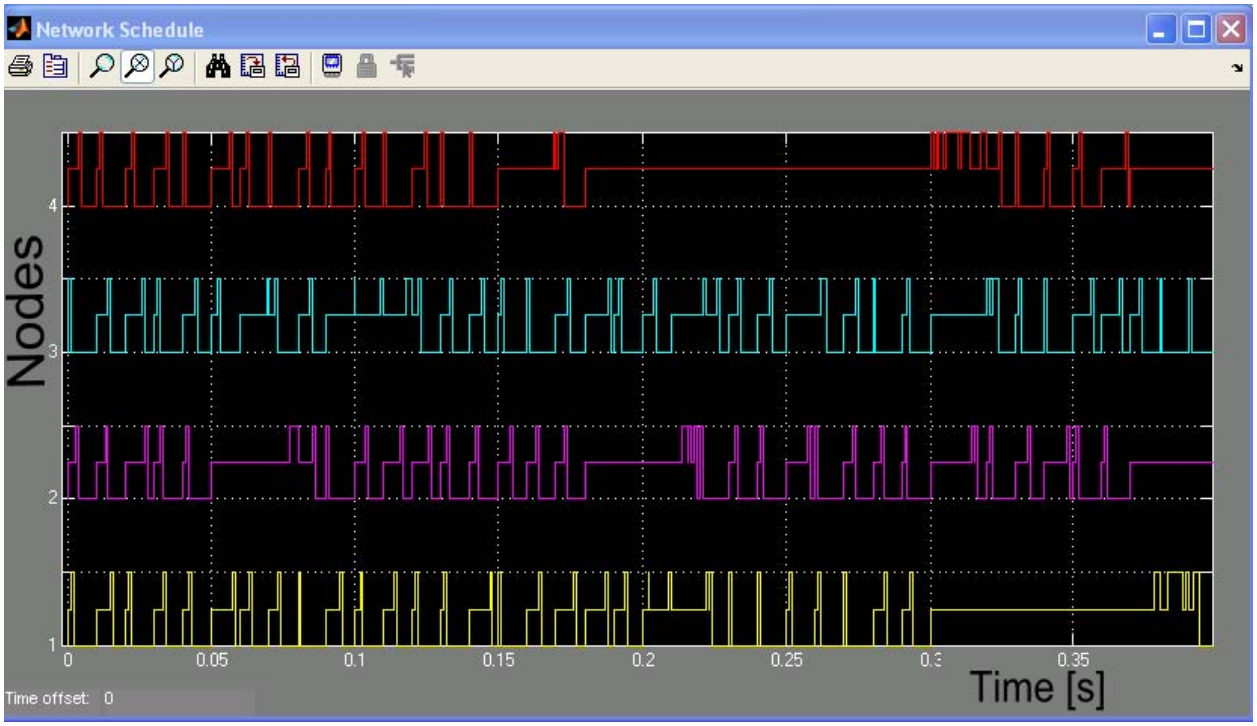



Figure 6.10 Network schedule using Ethernet, a low level of the graph means that the node is idle, medium level that the node is waiting for media access, and high level that the node is sending.

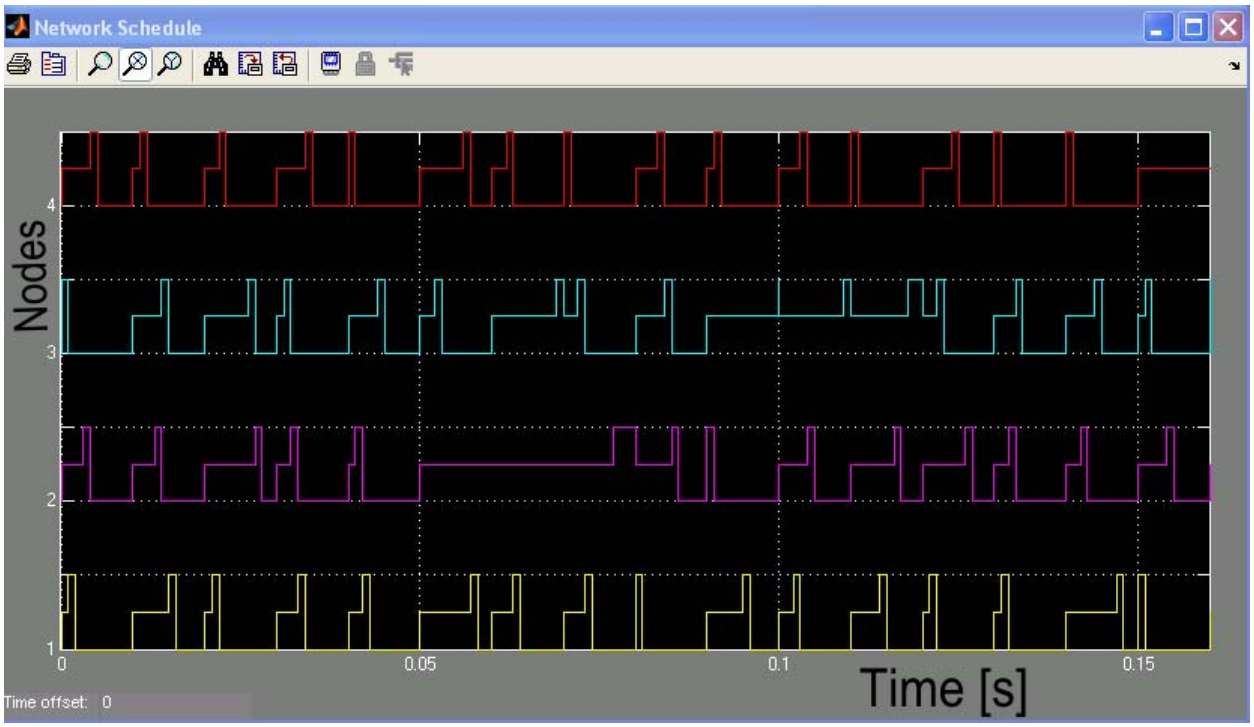



Figure 6.11 Zoomed network schedule using Ethernet, a low level of the graph means that the node is idle, medium level that the node is waiting for media access, and high level that the node is sending.

Using the same parameter settings, but changing to switched Ethernet at the block parameter interface results in the network schedule shown in the figure below.

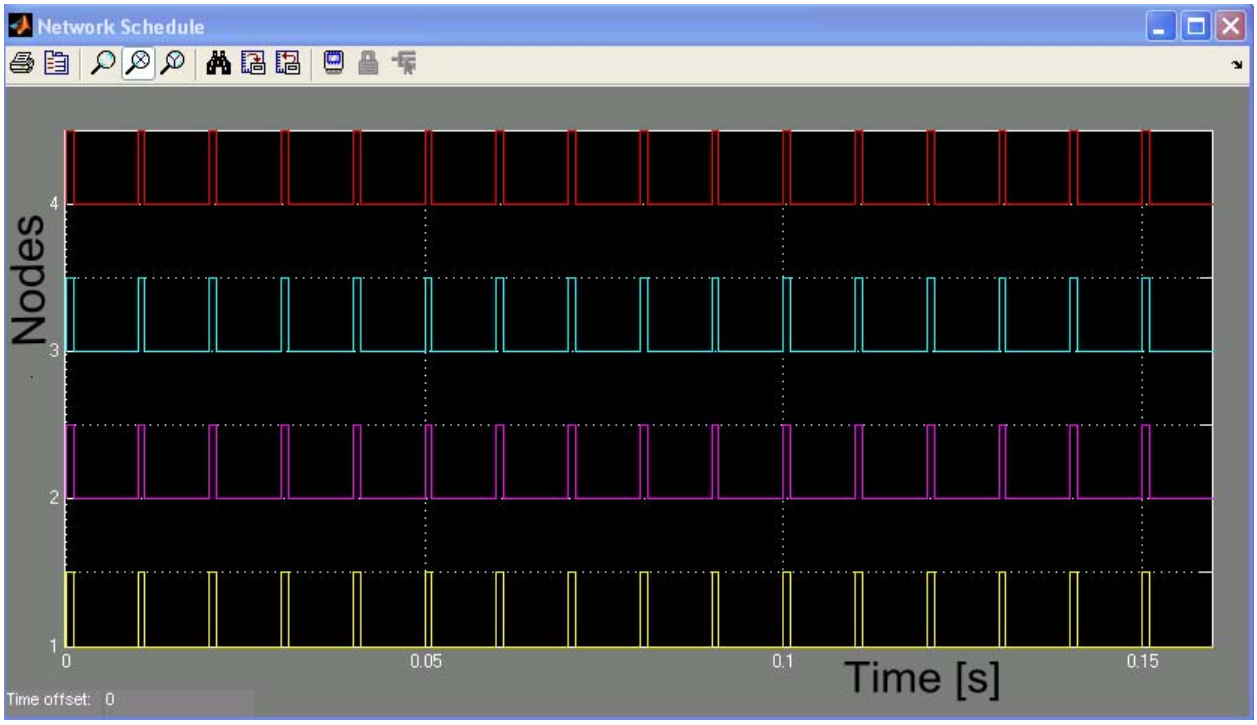



Figure 6.12 Network schedule of a TDMA algorithm applied on switched Ethernet, 100 Kbit 100 times per second, Bandwidth 100 Mbit / s

The figure above shows, that every node has the possibility to send its data according to the time schedule, appointed in the init- scripts of each node. The nodes are not forced to wait for access to the medium at any time. The reason why the peaks in the graph appear for each node simultaneously is because the chosen schedule is very symmetric for each node, which means that every node is either sending or receiving at the same time with all the other nodes.

Since the data rate is 100 Mbit/s in the previous simulation and the frame size was chosen with 100 Kbit with a sending rate of 100 times per second (period 0.01 in the init script), only one tenth of the available network capacity was used. However because every node has claim on 100% of the link capacity, the amount of data sent over the network can still be enlarged by the factor ten. The following graph shows the same simulation as before, only with 1000 Kbit sent 100 times per second.

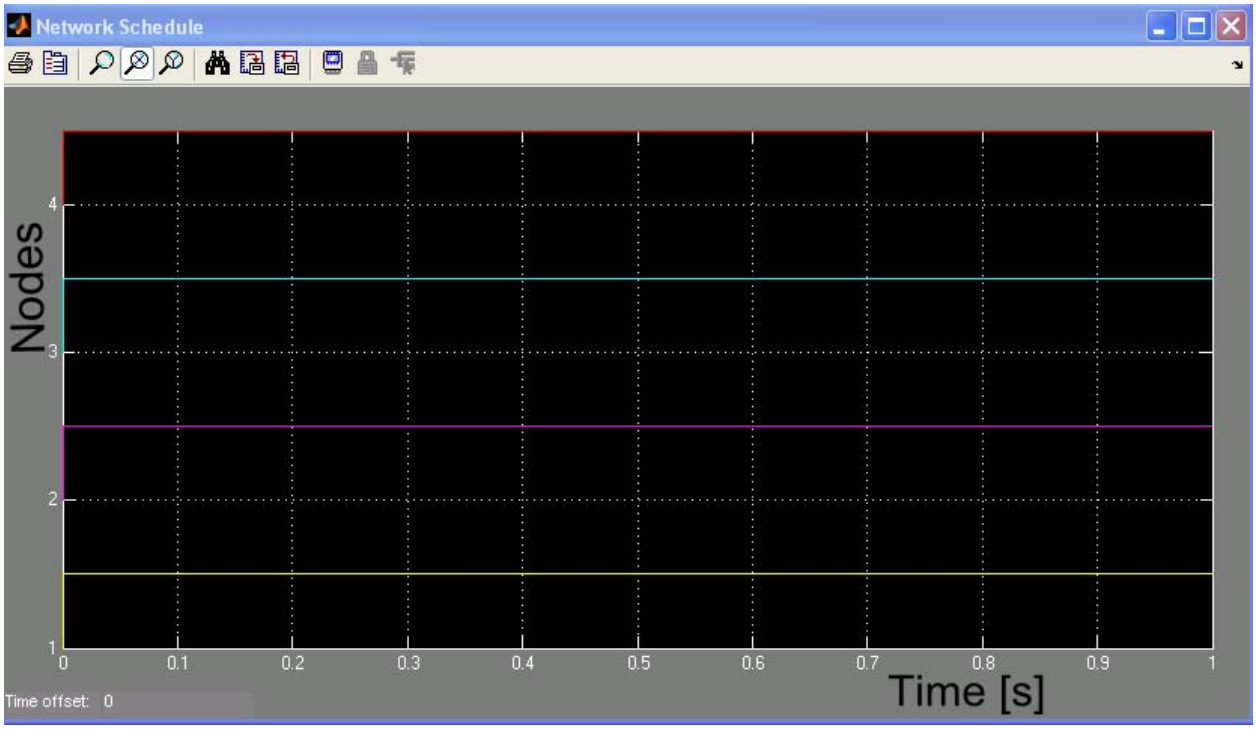



Figure 6.13 Network schedule of a TDMA algorithm applied on switched Ethernet, 1000 Kbit 100 times per second, Bandwidth 100 Mbit / s

As shown in the figure above, every node is busy with either receiving or transmitting data at every time of the simulation and thus, a workload of 100 % is reached.

An extract of the simulation outcome at the Matlab command window can be found in the abstract. This extract gives information on which node sends to which destination with the corresponding timestamp and the message content. The value of message always depends on the source and the destination node. The first number states the number of the node which sends the message and the second number stands for the destination node. A message value of “41” for example means that node four sends to node one.

# 7. CONCLUSION

In this master's thesis a possible solution for combining the advantages of high performance switched Ethernet and the in- vehicle protocol FlexRay was developed. Therefore thorough research in the area of vehicle networks, the requirements they have to meet and research on existing standards was made. The FTDMA algorithm FlexRay uses was chosen to be applied on switched Ethernet, because it offers the possibility to accommodate hard real time traffic and soft real time traffic and thus, offers the highest amount of flexibility. Since the focus of this work was to provide real time capability for switched Ethernet, the developed media access method concentrates on the static TDMA schedule of FTDMA. The event-triggered part for soft real time traffic was neglected.

A media access algorithm was developed on the basis of a static TDMA schedule for each node. Due to the full duplex connection to the Ethernet switch and thus the possibility to send and receive simultaneously, a schedule for each sending – and transmitting port was necessary. The designed TDMA schedules were based on certain traffic assumptions, made for a certain amount of nodes. Several examples with different traffic assumptions, number of nodes and cycle lengths were developed with the respective TDMA algorithms.

One way to validate the developed media access algorithm was to calculate the utilization for each example. Since there are a lot of different ways of how to design such a TDMA schedule for a certain traffic assumption, the functionality of the algorithm strongly depends on the traffic assumption and the corresponding schedule design. Calculating the utilization of the whole architecture shows that the method developed in this project also works for high utilized networks and nodes, as long as the TDMA schedule is designed properly and with foresight concerning deadlines.

Unfortunately the complexity and difficulty of designing an adequate TDMA algorithm increases enormously with the amount of nodes. Thus, the method is probably not very practical for huge network architectures with large numbers of nodes.

The second way to validate was to simulate a simple four node example with 100 % utilization for each node's sending and receiving port. The simulation shows that the developed algorithm works without violating any deadlines and consequently provides real time capability, what was the aim of this project.

Since this work shows the functionality and the real time capability of the developed media access method, possible future work would be to include the time triggered FTDMA part for soft real time traffic in the algorithm and implement it in the simulation part. Also the synchronisation between the nodes was assumed to already be assured and was not taken into account during the development of the schedules, what is another task for future work.

# 8. REFERENCES

[1] Ataide, F.; Santos, M.M.; Vasques, F.; "A comparison of the communication impact in CAN and TTP/C networks when supporting steer-by-wire systems" ;
Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on Volume 2, 8-10 Dec. 2004 Page(s):1078 - 1083 Vol. 2

[2] B. Muller, T. Fuhrer, F. Hartwich, R. Hugel, H. Weiler, Robert Bosch GmbH, "Fault Tolerant TTCAN Networks", www.can-cia.de/can/**ttcan**/mueller.pdf (accessed May, 2006)

[3] BMW Group, Harald Heinecke; "Automotive System Design – Challenges and Potential" Proceedings of the Design, Automation and Test in Europe Conference and Exhibition; 2005

[4] Casier, H.; Moens, P.; Appeltans, K.; "Technology considerations for automotive" Solid-State Device Research conference, 2004. ESSDERC 2004. Proceeding of the 34th European 21-23 Sept. 2004 Page(s):37 – 41

[5] F. Hartwich, B. Muller, T. Fuhrer, R. Hugel, Robert Bosch GmbH, "CAN Network with Time Triggered Communication", www.semiconductors.bosch.de/pdf/CiA2000Paper_2.pdf (accessed May, 2006)

[6] G. Cena, A. Valenzano. "Performance Analysis of Byteflight Networks" Factory Communication Systems, Proceedings of the IEEE, Sept. 2004 Page(s):157 - 166

[7] G.Leen, D. Heffernan, "Time-triggered controller area network" Computing & Control Engineering Journal, Dec. 2001. Page(s): 245-256

[8] H. Kopetz, G. Bauer, "The Time-Triggered Architecture" Proceedings of the IEEE, Volume 91, No.1, Jan, 2003. Page(s):112-126

[9] H. Kopetz, W. Elmenreich, C. Mack. "A Comparison of LIN and TTP/A" WFCS-2000 September 2000, Page(s): 99-107

[10] Intel; http://www.intel.com/design/mcs96/papers/autolxbk.htm; "Introduction to In-Vehicle Networking" (accessed May, 2006)

[11] J. Ferreia, P. Pedreiras, L. Almeida, J.A. Fonsea; "The FFT-CAN Protocol for flexibility in Safety Critical Systems"; IEEE 2002

[12] J.-L Scharbarg, M. Boyer, J. Ermont, C. Fraboul, "TTCAN over mixed CAN/Switched Ethernet Architecture" 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005. Volume 1, Sept. 2005 Page(s):665 - 668

[13] K.H. Johansson, M. Törngren, and L. Nielsen. Vehicle Application of Controller Area Network, www.md.kth.se/RTC/Papers/VehicleApplicationsCan2005.pdf (accessed May, 2006)

[14] Krug, M.; Schedl, A.V.; "New demands for invehicle networks" EUROMICRO 97. 'New Frontiers of Information Technology'., Proceedings of the 23rd EUROMICRO Conference; 1-4 Sept. 1997 Page(s):601 - 605

[15] L.M. Pinho, F. Vasques, "Reliable Real-Time Communication in CAN Networks" IEEE Transaction on Computers, Volume 52, Issue 12, Dec. 2003 Page(s):1594 - 1607

[16] Malaterre, P.; "Automotive trends: rationale and motivations" OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar 13 Nov. 1998 Page(s):1/1 - 1/8

[17] M. Jonsson, A. Åhlander, M. Taveniku, B. Svensson; "Time-Deterministic WDM Star Network for Massively Pareallel Computing in Radar Systems"; Proc. Massively parallel Processing using Optical Interconnections (MPPOI'96), Maui, HI, USA, OCT. 27-29, 1996, pp. 85-93, IEEE Computer Society Press, Los Alamitos, CA, USA

[18] M. Jonsson, K. Börjesson, M. Legardt, "Dynamic Tim-Deterministic Traffic in a Fiber-Optic WDM Star Network" Proc. 9th Euromicro Workshop on Real Time Systems (EWRTS'97) Toledo, Spain, June 11-13, 1997, pp. 25-33, IEEE Computer Society Press, Los Alamitos, CA, USA.

[19] Navet, N.; Song, Y.; Simonot-Lion, F.; Wilwert, C.; "Trends in automotive communication systems" Proceedings of the IEEE; Volume 93, Issue 6, June 2005 Page(s):1204 - 1223

[20] N. Navet, "Controller area network [automotive applications]" Potentials IEEE; Volume 17, Issue 4, Oct-Nov 1998 Page(s):12 - 14

[21] T. Fuhrer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther, Robert Bosch GmbH, "Time Triggered Communication on CAN", www.**can**-cia.de/**can**/tt**can**/fuehrer.pdf (accessed May, 2006)

[22] Thoma, P.; "Future needs for automotive electronics" Computer Design: VLSI in Computers and Processors, 1994. ICCD '94. Proceedings., IEEE International Conference on 10-12 Oct. 1994 Page(s):532 - 539

[23] T. Nolte, H. Hansson, and L.L. Bello. Implementing Next Generation Automotive Communications, www.mrtc.mdh.se/publications/0773.pdf (accessed May, 2006)

[24] Volvo Technology Corporation; Patrik Isaksson; "Communication technologies for automotive systems" 2003

[25] W. Xing, H. Chen, H. Ding, "The Application of Controller Area Network on Vehicle" Vehicle Electronics Conference, Volume 1, Sept. 1999 Pages(s): 455-458

[26] D. Henriksson, A. Cervin, Karl-Erik Årzén, "TRUE TIME: Real-time Control System Simulation with Matlab / Simulink", Department of Automatic Control, Lund Institute of Technology.

[27] F. Halsall, "Data Communication, Computer Networks and Open Syetms", 4th ed, Addisson-Wesley, ISBN 0-201-42293-X

[28] FlexRay Automotive Communication Bus, http://zone.ni.com/devzone/cda/tut/p/id/3352 (accessed Oct, 2006)

[29] R. Makowitz, C. Temple, "FlexRay – A Communication Network for Automotive Control System", IEEE 2006.

[30] T. Pop, P. Pop, P. Eles, Z.Peng, A. Andrei, "Timing Analysis of the FlexRay Communication Protocol", Proceedings of the 18th Euromicro Conference on Real-Time Systems, 2006.

[31] J. Loeser, H. Haertig, "Low-Latency Hard Real-Time Communication over Switched Ethernet", Proceedings of the Euromicro Conference on Real-Time Systems, 2004

[32] G. Cena, A. Valenzano, "On the Properties of the Flexible Time Division Multiple Access Technique", IEEE, 2006.

[34] X. Fan, M. Jonsson, "Guaranteed Real-Time Services over Standard Switched Ethernet", Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary, 2005

[35] M. Kim, Z. Park, J. Lee, "Performance Analysis of Switched Ethernets with Different Topologies for Industrial Communications", IEEE 2004.

[36] http://hus.hirschmann.com/English/Industrial_Ethernet_Products/Downloadcenter/Technology_and_White_Paper/White_Papers/PROFInet/index.phtml (accessed Jan 2007)

[37] Feld, J, "PROFINET - scalable factory communication for all applications", Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on.

[38] Ethernet Tutorial, http://www.lantronix.com/learning/net-tutor-switching.html, (accessed Oct 2006)

[39] Ethernet Tutorial, http://www.cse.cuhk.edu.hk/~cslui/CEG4430/switching.html, (accessed Oct 2006)

[40] Ethernet Switches, http://www.radio-electronics.co.uk/info/telecommunications_networks/ (accessed Oct 2006)

[41] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ethernet.pdf, (accessed Oct 2006)

[42] Ethernet Tutorials, http://www.etherwan.com/Images/whatisethernet.pdf (accessed Oct 2006)

[43] http://www.national.com/appinfo/networks/files/ethernet_basics.pdf, (accessed Oct 2006)

# 9. APPENDIX

Source Code of the Simulation part:

Initialization script for node 1

```
function node1_init

% node 1 in a 4 node switched Ethernet environment with TDMA schedule
%
% generates network traffic to node 2 & 3, and receives from node 2 & 4

% Initialize TrueTime kernel

ttInitKernel(1, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority

% Define the values offset and period for each sending task and for the
% receiving task. The values are measured in seconds. The priority is a
% fixed value 1, necessary for generating a periodic task, and equal for
% each node and task.

offset_n2 = 0;
offset_n3 = 0.01;
offset_rec = 0;
period_rec = 0.01;
period = 0.02;
prio = 1;

%Create the periodic sender and receiver tasks, with parameters: ('unique
%name of the task', offset, period, priority, 'name of the m- file with %function code')

ttCreatePeriodicTask('send_task_n1_n2', offset_n2, period, prio, 'n1_sen_n2');
ttCreatePeriodicTask('send_task_n1_n3', offset_n3, period, prio, 'n1_sen_n3');
ttCreatePeriodicTask('rec_task_n1', offset_rec, period_rec, prio, 'rec1');

% Initializing the network, has to be done for each node individually
% when a message is received, an interrupt handler is initiated

ttCreateInterruptHandler('nw_handler', prio, 'interrupt');
ttInitNetwork(1, 'nw_handler'); % node #1 in the network
```

Initialization script for node 2

```
function node2_init

% node 2 in a 4 node switched Ethernet environment with TDMA schedule
%
```

```
% generates network traffic to node 1 & 4, and receives from node 1 & 3

% Initialize TrueTime kernel

ttInitKernel(1, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority

% Define the values offset and period for each sending task and for the
% receiving task. The values are measured in seconds. The priority is a
% fixed value 1, necessary for generating a periodic task, and equal for
% each node and task.

offset_send_n1 = 0;
offset_send_n4 = 0.01;
period_send_n1 = 0.02;
period_send_n4 = 0.02;
offset_rec = 0;
period_rec = 0.01;
prio = 1;

%Create the periodic sender and receiver tasks, with parameters: ('unique
%name of the task', offset, period, priority, 'name of the m- file with
%function code')

ttCreatePeriodicTask('period_send_task_n2_n1', offset_send_n1, period_send_n1, prio,
'send_n2_n1');
ttCreatePeriodicTask('period_send_task_n2_n4', offset_send_n4, period_send_n4, prio,
'send_n2_n4');
ttCreatePeriodicTask('rec_task_n2', offset_rec, period_rec, prio, 'rec2');

% Initializing the network, has to be done for each node individually
% when a message is received, an interrupt handler is initiated

ttCreateInterruptHandler('nw_handler', prio, 'interrupt2');
ttInitNetwork(2, 'nw_handler'); % node #2 in the network

Initialization script for node 3

function node3_init

% node 3 in a 4 node switched Ethernet environment with TDMA schedule
%
% generates network traffic to node 2 & 4, and receives from node 1 & 3

% Initialize TrueTime kernel

ttInitKernel(1, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority
```

```
% Define the values offset and period for each sending task and for the
% receiving task. The values are measured in seconds. The priority is a
% fixed value 1, necessary for generating a periodic task, and equal for
% each node and task.

offset_send_n2 = 0.01;
period_send_n2 = 0.02;
offset_send_n4 = 0;
period_send_n4 = 0.02;
offset_rec = 0;
period_rec = 0.01;
prio = 1;

%Create the periodic sender and receiver tasks, with parameters: ('unique
%name of the task', offset, period, priority, 'name of the m- file with %function code')

ttCreatePeriodicTask('period_send_task_n3_n2', offset_send_n2, period_send_n2, prio,
'send_n3_n2');
ttCreatePeriodicTask('period_send_task_n3_n4', offset_send_n4, period_send_n4, prio,
'send_n3_n4');
ttCreatePeriodicTask('rec_task_n3', offset_rec, period_rec, prio, 'rec3');

% Initializing the network, has to be done for each node individually
% when a message is received, an interrupt handler is initiated

ttCreateInterruptHandler('nw_handler', prio, 'interrupt3');
ttInitNetwork(3, 'nw_handler'); % node #3 in the network
```

Initialization script for node 4

```
function node4_init

% node 4 in a 4 node switched Ethernet environment with TDMA schedule
%
% generates network traffic to node 1 & 3, and receives from node 2 & 3

% Initialize TrueTime kernel

ttInitKernel(1, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority

% Define the values offset and period for each sending task and for the
% receiving task. The values are measured in seconds. The priority is a
% fixed value 1, necessary for generating a periodic task, and equal for
% each node and task.

offset_n1 = 0.01;
offset_n3 = 0;
```

```
offset_rec = 0;
period_rec = 0.01;
period_n1 = 0.02;
period_n3 = 0.02;
prio = 1;

%Create the periodic sender and receiver tasks, with parameters: ('unique
%name of the task', offset, period, priority, 'name of the m- file with
%function code')

ttCreatePeriodicTask('send_task_n4_n1', offset_n1, period_n1, prio, 'send_n4_n1');
ttCreatePeriodicTask('send_task_n4_n3', offset_n3, period_n3, prio, 'send_n4_n3');
ttCreatePeriodicTask('rec_task_n4', offset_rec, period_rec, prio, 'rec4');

% Initializing the network, has to be done for each node individually
% when a message is received, an interrupt handler is initiated

ttCreateInterruptHandler('nw_handler', prio, 'interrupt4');
ttInitNetwork(4, 'nw_handler'); % node #4 in the network
```

Function for periodic sending task, node 1 sending to node 2

```
function [exectime, data] = n1_sen_n2(seg, data)

ttSendMsg([1 2], 12, 300);   % send the message “12” with 300 bits to network
time = ttCurrentTime;                    % 1, node 2.

 % getting the current simulation time and displaying it with a message
% that data was sent

disp('node 1 sending to node 2');
time
exectime = -1; % remaining execution time of current segment
```

Function for periodic sending task, node 1 sending to node 3

```
function [exectime, data] = n1_sen_n3(seg, data)

ttSendMsg([1 3], 13, 300);  % send the message “13” with 300 bits to network time =
ttCurrentTime;              % 1, node 3.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 1 sending to node 3');
time
```

```
exectime = -1; % remaining execution time of current segment

Function for periodic sending task, node 2 sending to node 1

function [exectime, data] = send_n2_n1 (seg, data)

ttSendMsg([1 1], 21, 300);  % send the message “21” with 300 bits to network time =
ttCurrentTime;              % 1, node 1.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 2 sending to node 1');
time
exectime = -1; % remaining execution time of current segment

Function for periodic sending task, node 2 sending to node 4

function [exectime, data] = send_n2_n4 (seg, data)

ttSendMsg([1 4], 24, 300);  % send the message “24” with 300 bits to network time =
ttCurrentTime;              % 1, node 4.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 2 sending to node 4');
time
exectime = -1; % remaining execution time of current segment

Function for periodic sending task, node 3 sending to node 2

function [exectime, data] = send_n3_n2 (seg, data)

ttSendMsg([1 2], 32, 300);  % send the message “32” with 300 bits to network time =
ttCurrentTime;              % 1, node 2.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 3 sending to node 2');
time
exectime = -1; % remaining execution time of current segment

Function for periodic sending task, node 3 sending to node 4

function [exectime, data] = send_n3_n4 (seg, data)
```

```
ttSendMsg([1 4], 34, 300);  % send the message “34” with 300 bits to network time =
ttCurrentTime;              % 1, node 4.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 3 sending to node 4');
time
exectime = -1; % remaining execution time of current segment
```

Function for periodic sending task, node 4 sending to node 1

```
function [exectime, data] = send_n4_n1(seg, data)

ttSendMsg([1 1], 41, 300);  % send the message “41” with 300 bits to network time =
ttCurrentTime;              % 1, node 1.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 4 sending to node 1');
time
exectime = -1; % remaining execution time of current segment
```

Function for periodic sending task, node 4 sending to node 3

```
function [exectime, data] = send_n4_n3(seg, data)

ttSendMsg([1 3], 43, 300);  % send the message “43” with 300 bits to network time =
ttCurrentTime;              %1, node 3.

% getting the current simulation time and displaying it with a message
% that data was sent

disp('node 4 sending to node 3');
time
exectime = -1; % remaining execution time of current segment
```

Periodic receiving function for node 1

```
function [exectime, data] = rec1(seg, data)

% Receiving function, checks periodically for new messages and stores them
% in a variable

data.u = ttGetMsg;
```

```
exectime = -1;
```

Periodic receiving function for node 2

```
function [exectime, data] = rec2(seg, data)

% Receiving function, checks periodically for new messages and stores them
% in a variable

data.u = ttGetMsg;
exectime = -1;
```

Periodic receiving function for node 3

```
function [exectime, data] = rec3(seg, data)

% Receiving function, checks periodically for new messages and stores them
% in a variable

data.u = ttGetMsg;
exectime = -1;
```

Periodic receiving function for node 4

```
function [exectime, data] = rec4(seg, data)

% Receiving function, checks periodically for new messages and stores them
% in a variable

data.u = ttGetMsg;
exectime = -1;
```

Interrupt function of node 1

```
function [exectime, data] = interrupt(seg, data)

%interrupt handler shows the received message with the corresponding time
%stamp

disp('node 1 received a message');
msg = ttGetMsg;
time = ttCurrentTime;
time
```

```
msg
exectime = -1;
```

Interrupt function of node 2

```
function [exectime, data] = interrupt2(seg, data)

%interrupt handler shows the received message with the corresponding time
%stamp

message = ttGetMsg;
disp('node 2 received a message');
time = ttCurrentTime;
time
message
exectime = -1;
```

Interrupt function of node 3

```
function [exectime, data] = interrupt3(seg, data)

%interrupt handler shows the received message with the corresponding time
%stamp

disp('node 3 received a message');
msg3 = ttGetMsg;
time = ttCurrentTime;
time
msg3
exectime = -1;
```

Interrupt function of node 4

```
function [exectime, data] = interrupt4(seg, data)

%interrupt handler shows the received message with the corresponding time
%stamp

disp('node 4 received a message');
msg4 = ttGetMsg;
time = ttCurrentTime;
time
msg4
exectime = -1;
```

Output on the MATLAB command window of the Simulation

| Time | Output | Message (value) |
|---|---|---|
| 0 | node 1 sending to node 2 | |
| 0 | node 2 sending to node 1 | |
| 0 | node 3 sending to node 4 | |
| 0 | node 4 sending to node 3 | |
| 0.0010 | node 1 received a message | 21 |
| 0.0010 | node 2 received a message | 12 |
| 0.0010 | node 3 received a message | 43 |
| 0.0010 | node 4 received a message | 34 |
| 0.0100 | node 1 sending to node 3 | |
| 0.0100 | node 2 sending to node 4 | |
| 0.0100 | node 3 sending to node 2 | |
| 0.0100 | node 4 sending to node 1 | |
| 0.0110 | node 1 received a message | 41 |
| 0.0110 | node 2 received a message | 32 |
| 0.0110 | node 3 received a message | 13 |
| 0.0110 | node 4 received a message | 24 |
| 0.0200 | node 1 sending to node 2 | |
| 0.0200 | node 2 sending to node 1 | |
| 0.0200 | node 3 sending to node 4 | |
| 0.0200 | node 4 sending to node 3 | |
| 0.0210 | node 1 received a message | 21 |
| 0.0210 | node 2 received a message | 12 |
| 0.0210 | node 3 received a message | 43 |
| 0.0210 | node 4 received a message | 34 |
| 0.0300 | node 1 sending to node 3 | |
| 0.0300 | node 2 sending to node 4 | |
| 0.0300 | node 3 sending to node 2 | |
| 0.0300 | node 4 sending to node 1 | |
| 0.0310 | node 1 received a message | 41 |
| 0.0310 | node 2 received a message | 32 |
| 0.0310 | node 3 received a message | 13 |
| 0.0310 | node 4 received a message | 24 |
| 0.0400 | node 1 sending to node 2 | |
| 0.0400 | node 2 sending to node 1 | |
| 0.0400 | node 3 sending to node 4 | |
| 0.0400 | node 4 sending to node 3 | |
| 0.0410 | node 1 received a message | 21 |
| 0.0410 | node 2 received a message | 12 |
| 0.0410 | node 3 received a message | 43 |

| | | |
|---|---|---|
| 0.0410 | node 4 received a message | 34 |
| 0.0500 | node 1 sending to node 3 | |
| 0.0500 | node 2 sending to node 4 | |
| 0.0500 | node 3 sending to node 2 | |
| 0.0500 | node 4 sending to node 1 | |
| 0.0510 | node 1 received a message | 41 |
| 0.0510 | node 2 received a message | 32 |
| 0.0510 | node 3 received a message | 13 |
| 0.0510 | node 4 received a message | 24 |
| 0.0600 | node 1 sending to node 2 | |
| 0.0600 | node 2 sending to node 1 | |
| 0.0600 | node 3 sending to node 4 | |
| 0.0600 | node 4 sending to node 3 | |
| 0.0610 | node 1 received a message | 21 |
| 0.0610 | node 2 received a message | 12 |
| 0.0610 | node 3 received a message | 43 |
| 0.0610 | node 4 received a message | 34 |
| 0.0700 | node 1 sending to node 3 | |
| 0.0700 | node 2 sending to node 4 | |
| 0.0700 | node 3 sending to node 2 | |
| 0.0700 | node 4 sending to node 1 | |
| 0.0710 | node 1 received a message | 41 |
| 0.0710 | node 2 received a message | 32 |
| 0.0710 | node 3 received a message | 13 |
| 0.0710 | node 4 received a message | 24 |
| 0.0800 | node 1 sending to node 2 | |
| 0.0800 | node 2 sending to node 1 | |
| 0.0800 | node 3 sending to node 4 | |
| 0.0800 | node 4 sending to node 3 | |
| 0.0810 | node 1 received a message | 21 |
| 0.0810 | node 2 received a message | 12 |
| 0.0810 | node 3 received a message | 43 |
| 0.0810 | node 4 received a message | 34 |
| 0.0900 | node 1 sending to node 3 | |
| 0.0900 | node 2 sending to node 4 | |
| 0.0900 | node 3 sending to node 2 | |
| 0.0900 | node 4 sending to node 1 | |
| 0.0910 | node 1 received a message | 41 |
| 0.0910 | node 2 received a message | 32 |
| 0.0910 | node 3 received a message | 13 |

| | | |
|---|---|---|
| 0.0910 | node 4 received a message | 24 |
| 0.1000 | node 1 sending to node 2 | |
| 0.1000 | node 2 sending to node 1 | |
| 0.1000 | node 3 sending to node 4 | |
| 0.1000 | node 4 sending to node 3 | |
| 0.1010 | node 1 received a message | 21 |
| 0.1010 | node 2 received a message | 12 |
| 0.1010 | node 3 received a message | 43 |
| 0.1010 | node 4 received a message | 34 |
| 0.1100 | node 1 sending to node 3 | |
| 0.1100 | node 2 sending to node 4 | |
| 0.1100 | node 3 sending to node 2 | |
| 0.1100 | node 4 sending to node 1 | |
| 0.1110 | node 1 received a message | 41 |
| 0.1110 | node 2 received a message | 32 |
| 0.1110 | node 3 received a message | 13 |
| 0.1110 | node 4 received a message | 24 |
| 0.1200 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.1210 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |
| | node 4 received a message | 34 |
| 0.1300 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.1310 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |
| | node 4 received a message | 24 |
| 0.1400 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.1410 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |

| | | |
|---|---|---|
| | node 4 received a message | 34 |
| 0.1500 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.1510 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |
| | node 4 received a message | 24 |
| 0.1600 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.1610 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |
| | node 4 received a message | 34 |
| 0.1700 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.1710 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |
| | node 4 received a message | 24 |
| 0.1800 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.1810 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |
| | node 4 received a message | 34 |
| 0.1900 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.1910 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |

| | | |
|---|---|---|
| | node 4 received a message | 24 |
| 0.2000 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.2010 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |
| | node 4 received a message | 34 |
| 0.2100 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.2110 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |
| | node 4 received a message | 24 |
| 0.2200 | node 1 sending to node 2 | |
| | node 2 sending to node 1 | |
| | node 3 sending to node 4 | |
| | node 4 sending to node 3 | |
| 0.2210 | node 1 received a message | 21 |
| | node 2 received a message | 12 |
| | node 3 received a message | 43 |
| | node 4 received a message | 34 |
| 0.2300 | node 1 sending to node 3 | |
| | node 2 sending to node 4 | |
| | node 3 sending to node 2 | |
| | node 4 sending to node 1 | |
| 0.2310 | node 1 received a message | 41 |
| | node 2 received a message | 32 |
| | node 3 received a message | 13 |
| | node 4 received a message | 24 |