EtherSim: A Simulation of the Ethernet MAC Sublayer using C#

Arnout Swinnen

A dissertation presented for the degree of

Licenciaat in de Toegepaste Informatica

Promotor: Prof Dr Ir Marnix Goossens

Advisor: Ir Bart Haagdorens

TELE Research Group

Faculty of Sciences – Department of Computer Sciences

Vrije Universiteit Brussel

Brussels, Belgium, 2003-2004

Acknowledgements

Many thanks go to Ir. Bart Haagdorens for his support and indications and also to Aldatillian for alpha and beta testing.

Abstract (English)

Computer Networking involves some principles that are hard to explain to other people. To make it easier to explain some basic concepts, we developed EtherSim. As EtherSim illustrates the working of the MAC sublayer of Ethernet, we'll discuss the most important features of the MAC sublayer of Ethernet and how these features are implemented in EtherSim. The program can also be used to illustrate what could go wrong if the constraints specified in the IEEE standards are not obeyed. Because EtherSim has been built using C#, we compare C# to Java by highlighting the most important differences. As performance is always an issue, we also compare the performance of C# to the performance of Java via benchmarking.

Abstract (Nederlands)

Hoewel computer netwerken veel mogelijkheden bieden, zijn een aantal basis concepten moeilijk uit te leggen. Daarom hebben we applicatie EtherSim ontwikkeld: deze applicatie illustreert de werking van de MAC sublayer in Ethernet. Dit document bevat dan ook een studie van de Ethernet MAC sublayer. Daarnaast belichten we ook hoe deze concepten geimplementeerd zijn in EtherSim. Daar EtherSim geprogammeerd is in C#, analyseren we ook C# en plaatsen het in een kader ten opzichte van Java. Dit wordt in grote mate verwezenlijkt door de verschillen tussen beide programmeertalen te benadrukken. In een laatste fase wordt de performantie van C# getoest aan de performantie van Java, z'n direct concurrent, via benchmarking.

Table of Contents

Chapter 1 Introduction 1
Chapter 2 Network Concepts 5
2.1 Contention 5
2.1.1 Contention in Wireless LANs 8
2.2 Ethernet 12
2.2.2 Manchester Encoding 17
2.2.3 Ethernet in EtherSim 18
2.3 Repeaters 19
2.3.1 Repeater 19
2.4 Twisted Pair Cables 21
2.4.1 Encoding Schemes 22
2.5 Bridging 26
2.5.1 Spanning Tree Bridges 27
2.5.2 Bridge Protocol Data Units (BPDU) 37
2.5.3 Changing the State of a port 38
Chapter 3 Structure of EtherSim 41
3.1 Overall Structure 41
3.2 The time-triggered architecture 46
3.3 User Interface 50
Chapter 4 Microsoft .NET 52
4.1 Java versus C# 52
4.1.1 Important Differences 53
4.2 The C in C# 56
4.3 Memory Management in .NET 57
4.3.1 Garbage Collection 58
4.3.2 Finalization 58
4.3.3 Improvements to the Garbage Collector 59
4.4 Memory Management in Java 60
4.5 Benchmarking 61
4.5.1 Arithmetic Benchmark 62
4.5.2 Scimark 2.0 Benchmark 64

4.6 Conclusion 70
Chapter 5 Conclusion 72
Appendix A The Mark and Compact Algorithm 79
Appendix B Benchmarks: Detailed Results 81
Christopher W. Cowell-Shah Benchmark 81
Scimark 2.0 detailed results 82

List of Figures

Fig 1: The 7 layers of the OSI Reference model 3
Fig 2: Collision Detection 7
Fig 3: The Hidden Station Problem 10
Fig 4: The bounds on the number of slottimes to back off after a collision 13
Fig 5: Calculation of the minimal frame length in Thick Ethernet 13
Fig 6: An IEEE 802.3 Ethernet frame (IEEE 802.3, 2002, p. 38) 14
Fig 7: Manchester Encoding Waveform Examples (IEEE 802.3, 2002, p 122) 17
Fig 8: 2 Hubs connected with a Full Duplex Link 19
Fig 9: The Physical Layer for 100baseT4 and 100baseTx (IEEE 802.3, 2002) 21
Fig 10: NRZI Encoding of the bits 1111011100 (IEEE 802.3, 2002) 23
Fig 11: NRZI-3 Encoding for 1111011100 (IEEE 802.3, 2002) 24
Fig 12: The operation level of the Bridge (IEEE 802.1D, 1998) 27
Fig 13: A Redundant Bridge in a Network 28
Fig 14: An example Network Topology 29
Fig 15: The Active topology for the given example topology 30
Fig 16: The Configuration BPDU (IEEE 802.1D, 1998) 36
Fig 17: Topology Change BPDU (IEEE 802.1D, 1998) 38
Fig 18: State Diagram for port changes of a bridge 39
Fig 19: Hierarchy of the Virtual Lab 42
Fig 20: Structure of the Transceivers 44
Fig 21a: Polymorphism in C# 54
Fig 21b: Polymorphism in C# 55
Fig 22: Average results of the Arithmetic Benchmark 62
Fig 23: Results of the small variant of the Scimark 2.0 Benchmark 66
Fig 24: SciMark 2.0 Results for the large variant 69
Fig 25: Average Results for the Scimark 2.0 Benchmark 71
Fig 26: The Mark Phase 80
Fig 27: The Compact phase 80

List of Tables

Table 1: MAU maximal delay allowed in Bittimes 20
Table 2: The nibbles used in 4B/5B (IEEE 802.3, 2002) 25
Table 3: C# Results for the Arithmetic Benchmark (in milliseconds) 61
Table 4: Java Results for the Arithmetic Benchmark (in milliseconds) 61
Table 5: Paired t-test results for the Arithmetic Benchmark (in Milliseconds) 63
Table 6: The .NET results of the SciMark 2.0 Benchmark in the small variant (MFLOPS) 64
Table 7: The Java results of the SciMark 2.0 Benchmark in the small variant (MFLOPS) 64
Table 8: Paired t-test results for the SciMark 2.0 benchmark between Java and C# (in MFLOPS) 65
Table 9: Results of the Large Scirmark 2.0 Benchmark in C# (in MFLOPS) 67
Table 10: Results of the Large Scimark 2.0 Benchmark in Java (in MFLOPS) 67
Table 11: Paired Student-t tests for the complex variant of the SciMark 2.0 Benchmark (MFLOPS) 68
Table 12: C# Results for the Arithmetic Benchmark (ms) 82
Table 13: Detailed results for Java for the Arithmetic Benchmark (ms) 82
Table 14: Java Results for the Arithmetic Benchmark (ms) 83
Table 15: The Java results of the SciMark 2.0 Benchmark in the small variant (MFLOPS) 83
Table 16: The .NET results of the SciMark 2.0 Benchmark in the small variant (MFLOPS) 83
Table 17: Java Benchmark results of the small variant of the SciMark 2.0 Benchmark (MFLOPS) 83
Table 18: Detailed results for .NET of the small variant of the SciMark 2.0 benchmark (MFLOPS) 84
Table 19: Detailed results for Java of the large variant of the SciMark 2.0 benchmark (MFLOPS) 84
Table 20: Detailed results for .NET of the large variant of the SciMark 2.0 benchmark (MFLOPS) 85
Table 21: Detailed results for .NET of the large variant of the SciMark 2.0 benchmark (MFLOPS) 85

Chapter 1
Introduction

The computer was one of the first machines that didn't help man with hard labor like the plough did, but it helped man with a process that was typical for mankind: thinking. Those machines were originally developed for calculations, but they quickly evolved and got used for several tasks, such as planning, information storage and management in several domains. As computers became more and more popular, the need for interconnecting them also grew. The benefits of interconnected computers are legacy: it becomes possible to share expensive hardware, access a computer without having to move physically and transportation of information could be done completely electronically: Because of the development of computer networks, there is nowadays hardly a difference between sending an order for your sandwiches to the shop around the corner and sending a copy of a trade contract to the government of Chile. An important factor that helped in the evolution of computer networks is entertainment factor. While ten years ago computer networks were reserved for business, they found their way into people's home. This has had important implications, such as the development of broadband routers for domestic use.

Although there are large benefits on computer networks, it may be quite difficult to understand the functioning and all the aspects of a computer network. When speaking about computer networks, it is desirable to organize the computer networks as a stack of layers (Tanenbaum, 2003, pp. 26-49). Such a layer offers some services to the higher layers. The use of layers gives the possibility to make an abstraction between the implementation details and its interface. When using these layers, we can state that a layer communicates with the same layer on another machine. The rules and assumptions made for this communication, also called the protocol, are encapsulated in this layer and thus known by both communicating machines. In reality, no layer immediately communicates with the corresponding layer on some other machine, but the information to be exchanged is passed to a lower layer. This implies that the layers are hierarchically organized. The lowest layer in the hierarchy then places the information to be communicated on the physical medium. This physical medium can then transport the information encoded as electrical pulses, optical light signals or whatever else that crosses your mind. This encoding on the physical medium is not of any importance for the higher layers. The most common used stack of levels and protocol, also called network architecture, is the International Standards Organization (ISO) Open Systems Interconnection (OSI) reference model. This model was developed by Day and Zimmerman in 1983 (Day and Zimmerman, 1983) and has been revised by Day in 1995 (Day, 1995). Improvements to

the OSI Reference model are still possible, recent work from Bauer B. and Patrick A.S. added 3 layers to the 7 layered OSI reference model (Bauer and Partick, 2004). These three layers are then used for Human-Computer Interaction. These extra layers hide the technology for the user, and are quite important in recent high tech devices, such as the modern mobile phones. Because these extra layers don't really concern protocols, we don't consider them any further. The higher layers of the OSI Reference open numerous perspectives, but because of their diversity, it is hard to consider them in detail. Each type of application such as FTP, HTTP, SMTP and numerous others have their own specifications concerning the implementation of the Application Layer. The next layer, the Presentation layer is concerned with the semantics of the information transmitted. The concept of Electronic Data Interchange (EDI) middleware used in enterprises is located in this layer (Rapaport, 2004). The session layer is especially useful in environments where multiple users or processes work on the same item, for example in groupware. It concerns access to the items and synchronization.

The three layers mentioned above can be considered as layers that relate rather to the application semantics than to the network technology. The lower layers really consider the network technology. The highest of these layers is the Transport Layer. The Transport Layer splits the data to be transmitted into pieces and re-assembles the pieces at the other end. The transport layer also determines what type of quality of service has to provide: must all parts arrive, may the parts be reordered? The transport layer is the lowest layer that has point-to-point communication between the source and destination machine. In other layers, intermediate machines may be introduced to establish the communication.

The next lower layer is the network layer. The major concern of the network layer is routing individual packets. In correspondence with routing of packets, the network layer is also responsible for coping with congestion or overloaded locations in the network. The most important aspects of quality of service that are affected with the network layer are jitter, transit time and delay.

One level lower in the OSI model, the Data Link Layer is located. This layer splits up the input data into frames and transmits these frames. Transmitting a frame considers when and how to put a frame on the line. When multiple machines can start transmitting on the same medium, the problem of which transmitter can transmit when is controlled by the Medium Access Control (MAC) (Kleinrock and Tobagi, 1975, Cited in Tanenbaum, 2003, pp 255-271) sublayer of the Data Link Protocol. This layer is discussed in more detail later on and has also been modeled into EtherSim application.

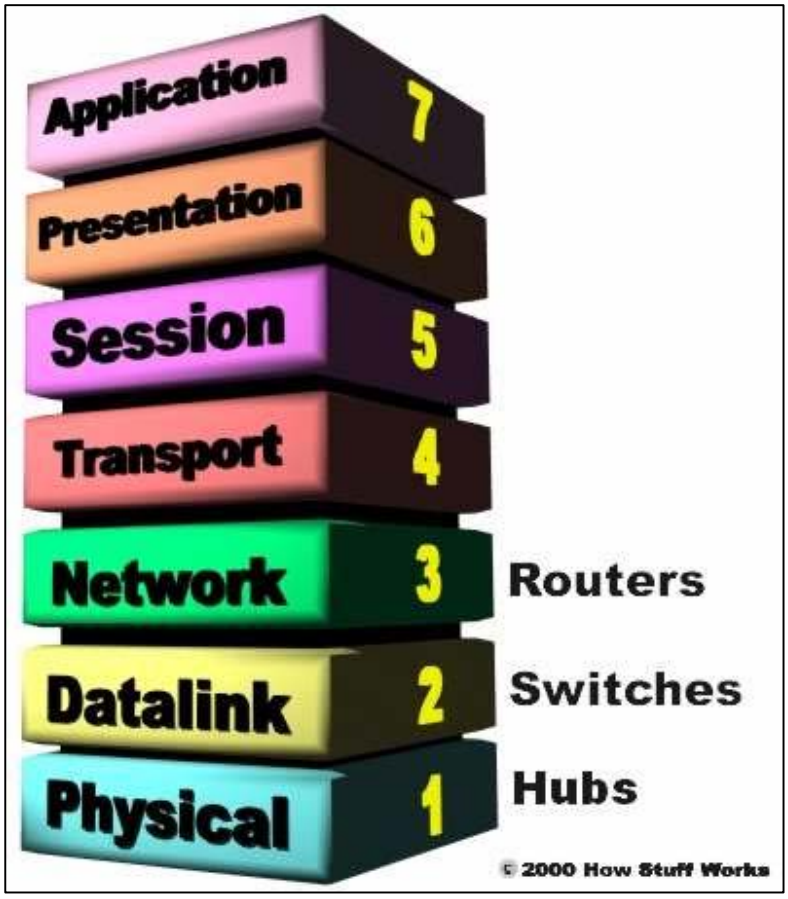

Fig 1: The 7 layers of the OSI Reference model

Image Source: http://computer.howstuffworks.com/nat3.htm

The bottom layer in the OSI Reference model is the Physical layer. The Physical Layer is concerned with transmitting bits over the communication channel. It concerns the power and frequency needed to transmit a particular bit.

Although a strict separation between the higher discussed layers is not always present in an implementation of some network architecture, the distinction can roughly be made for each implementation. For example, a bridge covers only the Physical Layer and the MAC sublayer of the data link layer, nevertheless it also determines on which outgoing port to forward frames. This obviously is a kind of routing, which in fact belong to the networking layer.

The area of networking has much evolved since the invention of computer networks in the 1970s. Apart from higher speeds, such as the evolution from 10Mbps LAN networks to 10Gbps networks, the technology itself has also evolved. The medium used to transmit data has evolved from a coaxial cable

over Unshielded Twisted Pair (UTP) cables to optical fibers and even radio waves. Optical fibers are used for high speed networks over longer distances while radio waves are used for wireless networks. While changing from coax to optical fiber only asks for changes in the physical layer of the OSI reference model, the increase in network speed also induced other problems, such as collision detection in high speed LANs. These problems are discussed in detail in Chapter 1.

The introduction of Wireless LANs on the other hand induced more thorough modifications. It is self-evident that the use of radio waves asks for another implementation in the Physical Layer. Apart from the fact that the bits should be encoded into radio waves instead of electromagnetic or optical signals, the encoding scheme should also use a lot more redundancy. This redundancy is necessary because radio transmissions are more susceptible to external distortions than transmissions over cables. Because of its different nature, Wireless LANs also use another MAC protocol. The IEEE 802.11 standard specifies two modes of operation, one with a central polling mechanism, and a distributed solution based on the Medium Access with Collision Avoidance for Wireless (MACAW) protocol (IEEE, 1999) (Karn, 1990).

Associated with the network concepts, a program has been to illustrate the functioning of the MAC layer in Ethernet and Fast Ethernet networks. The program, called EtherSim, illustrates the working of Ethernet at several speeds, going from some Megabits per second to Gigabits per second. It is also possible to illustrate what could go wrong if the IEEE 802.3 standards aren't obeyed. Also a part of the IEEE 802.1D standard, which is concerned with bridges has been modeled into the program.

An important aspect of the program is that it is largely developed as time-triggered architecture (Kopetz, 1997, pp. 1-67), which means that actions are executed as a reaction to the occurrence of a clock tick. Nevertheless, also the event-triggered aspect is discussed. The discussion about the structure of the program is given in Chapter 3. This chapter highlights what's possible with EtherSim and how this is implemented.

The last chapter is concerned with the language in which the program has been written. The program has been written in C#, a part of Microsoft's .NET. As C# and .NET more in general can be seen as an immediate rival of Sun's Java, a comparison between both languages is included in Chapter 4. Because both languages resemble very well to each other, the differences are highlighted. We also did some benchmarking to compare the performance of C# to the performance of Java. Because the name C# makes us thinking of the programming language C, the analogies between C# and C are also highlighted. We also have a quick look under the hood of the virtual machine used by C#, in particular we'll consider the garbage collector and memory management of C# more in general.

Chapter 2
Network Concepts

After the invention of the computer, it would be nice to interconnect multiple computers with each other. The most important benefit of interconnecting computers is the possibility to share expensive hardware among several computers. Other benefits are the possibility to access data remotely, provide backups and communication.

2.1 Contention

An important issue in computer communication is the allocation of the medium. The problem is that multiple senders might want to use the same medium simultaneously. Protocols that regulate the access control to the medium make part of the Medium Access Control (MAC) sublayer, a sublayer of the data link layer in the OSI reference model. Multiplexing allows multiple signals or streams to be transmitted over the same medium. The concept of multiplexing already existed in radio transmission and the telephone system. It has been adopted afterwards in computer networks. The two most important variant of multiplexing are Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM).

FDM divides the frequency into frequency bands where each separate signal has exclusive possession of at least one of these bands during transmission. Each signal is then shifted in the frequency domain such that the different multiplexed signals don't overlap with each other. A variant of FDM is Wave Division Multiplexing (WDM), which uses the same principle but at very high frequencies. WDM has been developed in the 90's in the context of optical fibers. WDM multiplexes several optical signals by shifting them in the frequency domain such that each signal can use a part of the complete frequency domain available on the shared fiber. The major difference between electrical FDM and optical WDM is that the multiplexing and de-multiplexing hardware for WDM can be built as a completely passive device; a kind of prism can do the job. Because these devices operate on a complete passive base, they are highly reliable. Although this is an important benefit of WDM and optical fibers in general over FDM with electrical signals, there also a huge drawback. At the moment it is still impossible to change the frequency of optical waves without converting them first to electrical signals. So when using optical links over multiple hops, two possibilities exist. Either the optical signal is converted into an electrical signal and

then retransmitted as an optical frequency on some other frequency. It is evident that this solution is expensive, both the hardware needed is expensive and the conversion also extends the delay of the signal propagation. Another solution is to use the same frequency from begin to end. The major concern is then that the same band of the spectrum must be used on all intermediate links. This solution is equivalent to the graph colouring problem (Guthrie, 1850, Cited in Grimaldi, 1999, pp. 532-538), which NP-hard (Cook, 1971, Cited in Norvig and Russel, 1995) to solve. Nevertheless, this approach seems to be most promising, as it is possible to introduce algorithms based on Artificial Intelligence which try to solve this problem.

With TDM, in contrast to FDM, the complete bandwidth of channel is periodically allocated to one of the input streams. While FDM can only be applied in the analogue domain, TDM opens the possibility for digital applications. The TDM multiplexer divides the input stream into packets, and places these packets into the composite (multiplexed) output stream. Two variants of TDM exist, static TDM which locates the incoming parts of the different input streams in a round robin fashion on the output stream. Please note that the parts from the different input streams don't have to be equal in size. This means that some input streams can be favoured in relation to others. A second variant of TDM is statistical TDM. Statistical TDM places the parts of the input streams on the composite stream in a round robin fashion, but only considers the input streams that are active. Because in the static variant a part of each input stream is placed on the composite stream whether the stream is sending data or not, the de-multiplexer can de-multiplex the incoming data from the moment that its clock is synchronized with the clock of the transmitter. In the statistical variant, this is no longer possible. To be able to de-multiplex the composite stream, headers must be used. It could be possible to use a header at the beginning of each round, indicating which input streams will be affected and where they are located. Another possibility would be to add a header to each part that is placed on the composite stream.

With TDM, it is impossible to have more than one device transmitting on the same medium simultaneously. If more than one signal is on the same medium, all signals on the medium become unusable. As in a traditional Ethernet LAN, there is no central multiplexer or de-multiplexer, the individual devices must agree on when they can start sending and what to do if their signal gets corrupted with another signal or how to avoid collisions from happening. Because multiple devices contend with each other for –temporary exclusive- access to the medium, such a protocol is called a contention protocol. The most common used contention protocol is Carrier Sense Medium Access Collision Detection (CSMA/CD). Using the CSMA/CD protocol, a transmitter first listens the medium before

t

f it detects that a transmission is in progr
dium is free. This is the CSMA part in CS
ening at the sender's side but this doesn't he
ver's side, not at the sender's side. To be able
smission when it detects that a collision occurre
diately aborting the transmission saves bandwidt
lided, it waits a random time and retries then to trans

We can recognize three periods when using CSMA/
going on that cannot collide any more, a contention pe

the medium, and an idle period if no device is trying to transmit anything (Tanenbaum, 2003,pp. 257-279). It is crucial to know the delay between the occurrence of a collision and realization by the sender that collision occurred by one of the transmitters. Suppose a Device1 in **Fig 2** starts transmitting, illustrated in **Fig 2a** and suppose it takes a time τ for a signal to propagate between the two farthest ends of the network. If now Device2 starts transmitting just before the signal reaches the other end of the network, a collision occurs at time τ-ε with ε being small (illustrated in **Fig2b**), Device2 detects immediately that its transmission has failed, Device1 doesn't notice that this collision happened before 2τ-ε. This means that a transmitter cannot be sure that its transmission will not collide before an interval with duration 2τ after the start of the transmission has expired. The delay τ depends on the transmission medium used and the distance of the network.

Another important issue is that the detection of a collision is an analog process. The transmitting device must monitor the medium by listening to it. Listening to the medium happens through the receiver. It is then checked if the received signal differs from the transmitted signal. This has important implications on the physical encoding of the data: if a 0 is for example encoded as 0 Volts, and this bit collides with another 0, also represented by 0 Volts, it is impossible to detect that a collision occurred. Another important implication is that the receiver must monitor the medium during transmission. This implies that the receiver part is in use during transmission and thus that it is impossible for a device to receive during transmission. As it is impossible for a device to receive and transmit simultaneously, a CSMA/CD system is a half duplex system.

2.1.1 Contention in Wireless LANs

Because of its nature, CSMA/CD doesn't fit well on Wireless LANs. Several reasons exist why this is a problem. The first reason is that a transmitter might not hear that another device is transmitting, but that a device that wants to start a transmission cannot hear this transmission because it is not in the radio range of the transmitter. This problem is referred to as the hidden station problem (Tanenbaum, 2003, pp 269), illustrated in **Fig 3**. A second reason is the inverse problem of the hidden station problem: the exposed station problem. This problem implies that a sender wanting to transmit can hear that another device is transmitting, and decides to not transmit because of this reason, while the receiver cannot hear this transmission. The device wanting to transmit thus doesn't start transmitting while it could start its transmission without any problem. A third reason is the fact that most radios are half duplex, which means that they can be either transmitting, or either receiving but not both simultaneous.

To cope with those problems, the IEEE 802.11, the standard for Wireless LANs, provides two alternatives: Distributed Coordination Function (DCF) (IEEE, 1999, pp. 72-86) and Point Coordination Function (PCF) (IEEE, 1999, 86-93). As can be expected from the name, DCF implements a distributed access protocol, i.e. without a central controller while PCF implements an access protocol with a central control point.

With DCF, Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is used. A mode of CSMA/CA built on top of Multiple Access Control with Collision Avoidance for Wireless (MACAW). MACAW is an extension of Medium Access Control with Collision Avoidance (MACA), optimized for wireless communication.

2.1.1.1 MACA and MACAW

The basic idea behind MACA is to have the receiver sen to ae i r. r t

tt(i)8.4no ittha orbp thc ian -5.6(a)-5.4(t-284( )-5.6it)8.4(h)-5.4her(e)-5.4(ce)-5.4(i)8.4vei .-6.1 To let(it)8.4(h)116e( )5.6

a-ni2453sision i21.5ll. tt .25.5( )-5.6(tn)-5.7(st)-5.7(sion)-574 wanostar(.)8.1( )-11.3a( )]TJ-

e-0867542198(i7546li7546e)4882nt

CTS

Important improvements to MACAW use were done by Bhaghavan et al. in 1994. The first improvement was the introduction of an Acknowledgement frame (ACK). This was introduced because lost frames are not retransmitted before the transport layer noticed they were lost. As with wireless communications the chance of loosing a frame is rather high, the ACK frame was introduced in the data link layer such that a lost frame can be detected and retransmitted much earlier. Another important

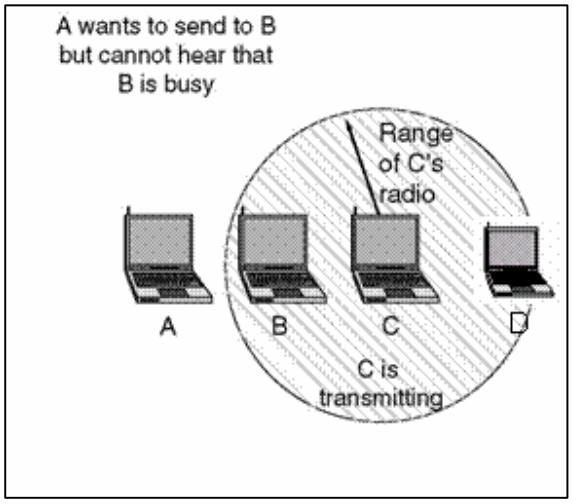

Fig 3: The Hidden Station Problem

Picture Source: http://www.mcs.vuw.ac.nz/courses/COMP204/2003T1/Handouts/COMP204_Review_Q_and_A.htm

improvement is the introduction of sensing the medium. If a device wanting to transmit detects that a RTS frame is sent to the same destination as it wants to send an RTS frame to, it waits until that transmission is over. The reason is simple: its RTS would have collided with the RTS that is currently being sent anyway. Two other improvements are introduced in MACAW: The Back-Off algorithm, described in the section about Ethernet, is run for each source-destination pair instead of running it for each station and a mechanism that allows stations to communicate about overloaded parts of the network (congestion) was also introduced. (Bhaghavan, 1994, Cited in Tanenbaum, 2003, pp. 269-270)

2.1.1.2 The MAC sublayer in the IEEE 802.11 standard

As said earlier, the DCF mode of MAC sublayer used in the IEEE 802.11 standard is based on the MACAW protocol. Although MACAW allows a station to transmit when it has heard an RTS but when it doesn't hear the CTS, the IEEE 802.11 standard doesn't allow this. A device hearing an RTS keeps silent until is expects the transmission to be over, and the transmission is over when the ACK frame has been received by the transmitter. Of course a device hearing a CTS frame also remains silent until the

transmitter has received the ACK frame. The time to remain silent can be deduced from the frame length included in the CTS/RTS frames.

Another important modification is the fact that a data frame is fragmented into fragments which are each acknowledged separately. The reason for this is that the longer the frame is, the more likely it is that the frame will get corrupted by external interferences. The reason for this is simply that the radio waves used for wireless LANs are in a frequency band that is very sensitive to external interferences. So fragmentation of the data frame reduces retransmission from the complete data frame to the corrupted fragments. The time that other devices which heard the CTS or RTS wait is limited until the transmitting device has received the ACK frame for the first fragment. Collisions afterwards are avoided with an intelligent timing scheme discussed later.

As mentioned earlier, two modes of operation are allowed in the IEEE 802.11 standard. The first mode, DCF is mandatory, while the second mode, PCF is optional. This PCF mode uses a central base station that polls the devices in, asking them if they have something to send. Because the control to the medium is centrally controlled, it is impossible for collisions to occur. It is important to note that not all devices need to be equally serviced. The base station periodically broadcasts a beacon frame, containing system information and inviting new stations to sign up for the polling service. The station can specify the rate at which it wants to make use of the polling service. The station is guaranteed to get the requested amount of bandwidth, which opens the possibility to give guarantees about quality of service.

Another remarkable feature of PCF is the integrated power management. A base station can order a device to go into sleep mode. If a base station does this, it queues up all frames to be transmitted to the device. The base station or the user can than wake up the device again, and the queued frames can be collected from the base station. This feature has been introduced to reduce the battery consumption and to extend lifetime of the device.

It is also remarkable that PCF and DCF can coexist in one cell. This is also accomplished via the timing scheme. The timing scheme is concerned with the amount of dead time after an acknowledgement before a frame or fragment can be transmitted. This interval depends on what is going to be sent. The shortest interval is the Short Inter-Frame Space (SIFS). After this time interval, only parties that already were in a communication can communicate: the next fragment after an ACK frame or a CTS frame in response to a CTS frame can be sent during this interval. The next interval is the PCF Inter-Frame (PIFS) interval. When the PIFS interval has expired, the base station can transmit a beacon or polling frame during this interval. After the next interval, the DCF Inter-Frame (DIFS) interval, any station operating in DCF mode

can try to acquire the channel. After the last interval, the Extended Inter-Frame Interval (EIFS) is used by a station to report a bad or unknown frame. This reporting has the lowest priority because a receiver isn't aware of what is going on in the network. If it had a high priority, it could possibly interfere with an ongoing dialog. (Tanenbaum, 2003, 295-302)

Currently, Wireless LANs aren't provided in EtherSim application, but they can be introduced in a next version.

2.2 Ethernet

The first local area network (LAN) was developed by Xerox in 1976. The reason for developing this was to make it possible to use the expensive printers by multiple computers. The system was called Ethernet and based on the ALOHA system, a satellite communication system developed by Norman Abramson in the 1970s. The system ran at a speed of 2.94Mbps. As the Xerox Ethernet became successful, DEC, Intel and Xerox established the DIX standard in 1978. The DIX standard was a standard for a 10Mbps Ethernet. In 1983 the DIX standard became the IEEE 802.3 standard with two minor changes which are discussed later in this paragraph. More detailed information is available in Tanenbaum (2003, pp. 271-291) and the IEEE 802.3 standard (2002).

2.2.1.1 The Functioning of Ethernet

The transmission medium for Ethernet is a coaxial cable, with segments of a maximal length of 500 meters. By making use of repeaters, the length could be extended to 2.5km. Ethernet uses CSMA/CD for medium access control. If a device detects that a collision has occurred, it doesn't stop transmitting immediately, but instead it transmits a jam signal. This jam signal ensures that the collision is detected by all other transmitting devices. The content of this jam signal is not specified in any standard, but it should not be designed as such that it corresponds to the 32 bit CRC check of the partially transmitted signal.

If the devices all retransmit immediately after the jam signal, exactly the same collision occurs again. To avoid this, the transmitting devices wait some time after receiving a jam signal before they retransmit the collided data. This time is not equal for all transmitting devices but each device determines this time to wait before retransmission locally by the exponential back-off algorithm. This time to wait takes into account the number of successive failed transmissions for some packet. The time to wait is always an integral multiple of the slottime: the time needed by one bit to get from one end of the Ethernet-LAN to

$$0 \leq r < 2^k \text{ with } k = \min(n, 10)$$

Fig 4: The bounds on the number of slottimes to back off after a collison

the other end. The number of slottimes r to wait always obeys to the inequality given in **Fig 4**, where n indicates the number of failed transmission for the current packet. Within those limits, the number of slottimes is chosen at random and such that the correlation between different r from different devices is minimal. The number of times a device tries to retransmit is also limited. If the number of retries to send some packet exceeds a predefined limit, the event of transmitting the given packet is reported as failed.

It is important to note that neither CSMA/CD nor Ethernet provides any acknowledgements. This means that although CSMA/CD can cope with interference of multiple devices trying to transmit simultaneously, it cannot correct errors caused by interferences. If the data transmitted by some device is garbled with external noise, such as an electromagnetic field next to the network cable, there are no provisions in the CSMA/CD or in the Ethernet protocol to correct this, although the Cyclic Redundancy Check (CRC) is used to detect the occurrence of such errors. If the need exists to send back an acknowledgement, this acknowledgement needs to compete with all other data waiting to be sent for channel time. Tokoro and Tamaru gave in 1977 a simple solution that allows speedy confirmation. Although this solution concerns only a simple modification to the protocol, this modification hasn't been adopted in the IEEE 802.3 standard (Tokoro and Tamaru, 1977, Cited in Tanenbaum, 2003, pp. 279).

Ethernet specifies both an upper and a lower limit on the size of the frames. The maximal size is 1518 bytes with the preamble excluded or 1526 with preamble included. This limit has been chosen arbitrarily. Larger frames could have been allowed, but they are limited because when the DIX Ethernet standard was

- At 10Mbps, 1 bit time is 1/10485760 s = 0.095 μs
- Propagation speed of Thick Coax is 231 m/μs
- The maximal delay induced by the repeaters is: 18*4* 0.095 = 6.84
- The total maximal delay in μs is, $\tau = 6.84 + 2500/231 = 17.66\mu s$
- $2\tau = 2*17.66\mu s = 35.33\ \mu s$
- #bits minimum required: 35.33/0.095 = 371bits.

Fig 5: Calculation of the minimal frame length in Thick Ethernet

developed the idea was that a Transceiver should be able to store a complete frame in its memory and that RAM was expensive at the time the DIX standard was established. The lower limit has another reason. As said earlier, Ethernet is a CSMA/CD system. This means that a transmitter can only be sure that its collision hasn't occurred after a time 2τ after the start of its transmission. As the IEEE 802.3 standard specifies a maximal network diameter of 2500 meters for a 10Mbps LAN and 4 repeaters, we can calculate the maximal round trip delay in a 10Mbps Ethernet LAN compliant to the IEEE 802.3 standard. A repeater adds 18 more bit times to this delay. This gives us the calculation given in **Fig 5**. So the very minimum is 371 bits, when using thick Ethernet. To have a safety margin, the minimal number of bits has been set to 512bits, which corresponds to 64bytes.

2.2.1.2 Structure of an Ethernet frame

The structure of the IEEE 802.3 Ethernet frames is given in **Fig 6**. It is important to mention that the structure of those frames hasn't changed during the evolution of Ethernet. This means that 100Mbps and even 1Gbps Ethernet frames still have the same structure as the original Ethernet frames. The reason for this is simply because organisations are in general not enthusiast when newer versions imply that those become incompatible with their existing infrastructure. For this reason, it was important for the 100Mbps standard to be compliant with the existing 10Mbps standard.

The Preamble field of the Ethernet frame is used to synchronize the receiver's clock with the sender's clock. The next field, Start of Frame Delimiter (SFD) is only available in the IEEE 802.3 specification, and doesn't exist in the DIX standard. The DIX standard has a preamble of 8 octets instead of 7 octets.

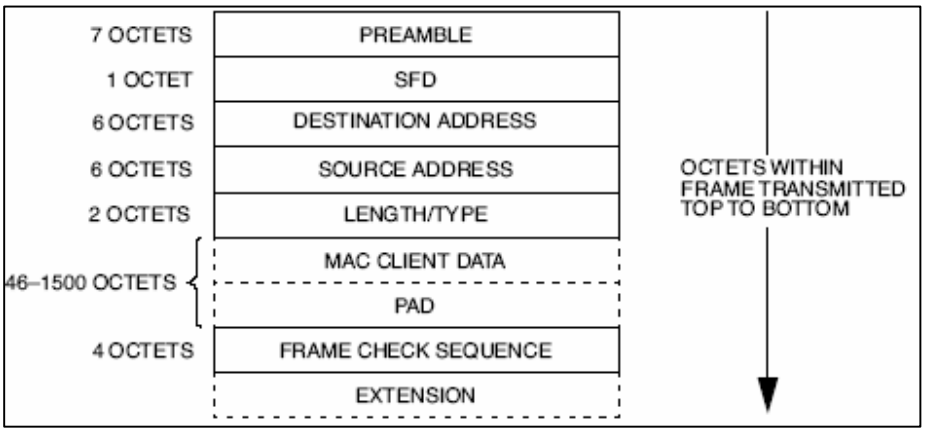

Fig 6: An IEEE 802.3 Ethernet frame (IEEE 802.3, 2002, p. 38)

This field was introduced for compatibility with the IEEE 802.4 and IEEE 802.5 standards. The SFD field is the first difference between the DIX standard and the IEEE 802.3 standard.

The destination address and source address fields indicate the source and destination devices. Those addresses are unique for each device and are assigned by the IEEE organisation. The addresses have a length of 48 bits. The highest order bit of the addresses is used to indicate whether the address is an ordinary address or a group address. Group addresses are both multicast addresses and broadcast addresses. Multicasting means that the frame should be delivered to multiple devices, but not all devices. This incorporates group management to determine where the frame should where it shouldn't be delivered. When a frame is broadcasted, it is delivered to every device connected to the network. The second highest order bit is used to differentiate between locally and globally administered addresses. The source address field will not be interpreted by the CSMA/CD sublayer.

The Type/Length field has a double function. In the DIX standard only a type field was provided, while the first versions of the IEEE 802.3 standard specified a length field. Because the DIX standard was already widely adopted by manufacturers and customers, IEEE threw the towel into the ring and allowed both. The Length field indicates the number of data octets that will follow in the data field while the type field indicates the nature of the MAC protocol, i.e. what the receiver should do with the frame. As both fields are specified to use the same octets, it is only possible to determine whether the octets contain a length field or a type field on their content: if the value of the field is less or equal than 1500 (decimal), the field is considered as a length field, else it is considered to be a type field. This field uses 2 octets.

The next field is the Data field. The data field is split up into the Mac Client Data field and the Pad field. The Mac Client Data field holds the data that is sent in the frame. But as explained earlier, a minimum frame length is specified, but it might be possible that less than 46 bytes of data have to be sent. If that happens, the Pad field is used to ensure that the Data field contains 46 bytes. The total length of the Data field should always be between 46 and 1500 octets.

The Frame Check Sequence field contains a Cyclic Redundancy Check. Both the receiver and transmitter calculate a value in function of the Source Address, Destination Address, Type/Length and Data Field. The transmitter places this value in the Frame Check Sequence field and the receiver checks the value in this field against the value it calculated itself. If these values don't match, the frame is considered as corrupted. The encoding is calculated as follows:

- The n bits of the frame are considered as the coefficients of a polynomial M(x) of degree n-1. The first bit of the destination field is considered as the coefficient of the x^{n-1} term and the last bit of the data field is considered as the coefficient of the x^0 term.
- The polynomial M(x) is multiplied by x^{32} and divided by the polynomial G(x) = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$. This division produces the remainder R(x) which has a degree r<32.
- The coefficients of R(x) are considered as a 32-bit sequence.
- The bit sequence is complemented and the result is the CRC value.

The last field, the extensional field, is a field that is not always present. It has a length between zero and the minimal frame size. This field exists because at speeds above 100Mbps the minimal frame length of 64bytes becomes insufficient. This can easily be seen from the calculation in **Fig 5**. If the bit time is reduced, which is exactly what happens if the speed is increased, the number of necessary bits increases. To be able to cope with this problem, it is possible to reduce the maximal network diameter, but this can't be reduced infinitely. So this extension field extends the frame to a length that can be used in a CSMA/CD context. This extension field makes it possible to use Ethernet frame specified in the IEEE 802.3 standard in Ethernet at any speed.

2.2.1.3 Ethernet Cabling

The cabling of Ethernet has evolved over time. After the expensive high quality Thick Ethernet variant, cheaper solutions came up. A cheaper coax variant, 10Base2 or commonly called "Thin Ethernet", was developed. It uses a coax cable of lower quality and allows runs, without repeaters or other amplifiers, of only 200 meters, while the thick variant allows runs of 500 meters without repeaters or amplifiers. For this reason the Thick Ethernet variant is labelled 10Base5. A third variant, that became the most popular variant, uses twisted pair cabling. The reason for its popularity is that the existing CAT3 twisted pair cabling of the telephone system can be used. The twisted pair cable connects 2 devices, so to build a network of more than 2 computers, a special distributor is necessary. The CAT3 cabling allows runs of up to 100 meters, and the high-end CAT5 cabling allows runs of up to 200 meters. Ethernet using twisted pair cabling is referred to as 10BaseT. A last variant of the 10Mbps Ethernet uses optical fiber cabling. Although connectors for this type of cabling are expensive, optical fibers allow long runs, of up two 2 kilometres, without need for a repeaters or other amplifiers. Because of its characteristics, this variant of

Ethernet is typically used for runs between buildings. To 10Mbps Ethernet over optical fibers is referred as 10BaseF.

2.2.2 Manchester Encoding

This paragraph addresses the encoding of the Ethernet frames. This data encoding doesn't make part of the MAC Sublayer, but makes part of the Physical layer. It indicates how bits are transformed into electromagnetic signals. Manchester encoding is described in detail in the IEEE 802.3 standard (IEEE, 2002, pp. 121-123).

A naive encoding scheme would to encode a 0 with zero volts and a 1 with some other voltage. Although this encoding scheme is simple and straightforward, it is useless. The major problem with it is that it is impossible for the receiver to distinguish between the signal encoding a 0 and the medium being idle. An improvement would be to encode a 0 as some non-zero voltage, a 1 as some other non-zero voltage and that a voltage of 0 Volts means that the medium is idle. These voltages are sent out during some time interval. It is evident that the sender and the receiver must agree on this time interval. With this encoding scheme, another problem shows up: when there is a clock drift between the sender's and the

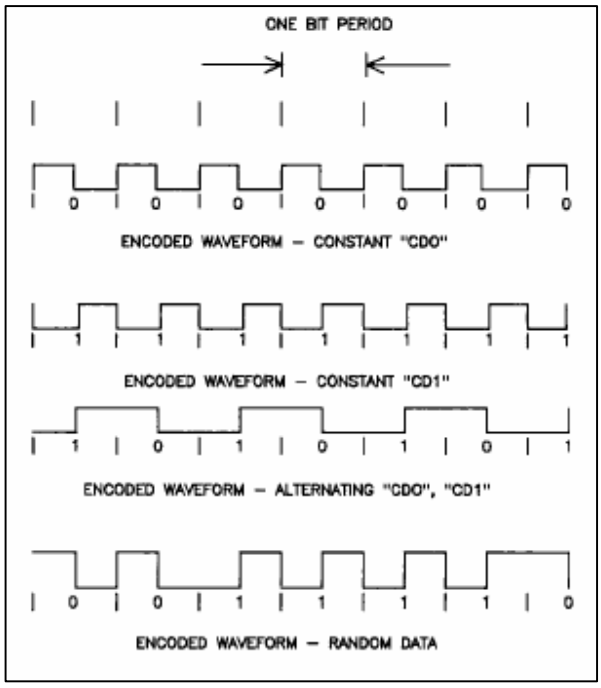

Fig 7: Manchester Encoding Waveform Examples (IEEE 802.3-2002, p122)

receiver's clock, the sender and the receiver get out of synchronization. If this happens, the receiver may receive bits that were not sent out if the receiver's clock runs faster than the transmitter's clock or not receive bits that were sent out if the receiver's clock runs slower than the transmitter's clock.

To overcome with these problems, the Manchester Encoding scheme is used. This encoding scheme encodes both the clock and the data into one signal. Manchester encoding makes use of 3 levels of voltage: a high and a low voltage, where the low voltage is the inverse of the high voltage and the idle voltage which is 0 volts. We define a bit-symbol as the signal that encodes a 1 or a 0. To encode the clock into the signal, there must be a transition in the middle of each bit-symbol. The data is encoded as follows: the first half of the bit-symbol is the logical complement of the data bit to be encoded, for a example a 1 begins has a high voltage in the first half, and the second half is the logical uncomplemented value of the data bit encoded, so the second half of the second bit-symbol representing a one has a low voltage. It is important to note that to send a sequence of the same bits, a sequence of 1's or a sequence of 0's, two transitions in voltage are needed for one bit. In the Ethernet IEEE 802.3 standard, the high voltage is +0.85 Volt and the low voltage is -0.85 Volt. The Manchester encoding is used for all 10Mbps variants of Ethernet, including the Coax variants, the twisted pair variant and the optical fiber variant.

2.2.3 Ethernet in EtherSim

The concept of traditional Ethernet is an important feature of EtherSim. EtherSim provides coaxial cables under the form of half duplex links. Although the Ethernet standard allows two variants of coaxial cable to be used: 10Base5 and 10Base2, this difference hasn't been modelled in EtherSim. The 10Base5 Ethernet, commonly called Thick Ethernet, makes use of a high quality, usually yellow, thick coaxial cable. Segments of up to 500 meters at 10Mbps without repeaters or amplifiers are allowed with this type of cable. 10Base2 Ethernet, commonly called Thin Ethernet, uses a lower end coaxial cable. With 10Base2, segments of 200 meters at 10Mbps are allowed without any repeaters or amplifiers. The reason for these limits is located in the electromagnetic properties of the coaxial cable. As those properties aren't modelled in EtherSim, it is impossible to model the difference between 10Base2 and 10Base5 Ethernet.

A special connector was needed to connect devices to the coaxial cable. For 10Base5 Ethernet, this is done with so-called vampire taps which have a pin that has to be pushed into the coaxial cable's core. For 10Base2 Ethernet, BNC connectors are used to for T-junctions with the coaxial cable. The length of the cable from the connector to the device is limited. The most important difference between both connectors is that the transceiver is integrated into the vampire tap. With the BNC connectors the transceiver is

integrated into the device. This makes the BNC connector just a passive connector. In EtherSim, connectors to devices are also provided. A connector is implemented as an ordinary half duplex segment, with this difference that it allows more than two items to be connected to it. In fact such a connector can be seen as a multiple-end half duplex segment. An important limit on the connector is that a limit is imposed on the number of ends where another half duplex segment can connect. As it is not possible to split a coaxial cable, the number of half duplex link segments that can be connected is limited to 2. By this measure, the semantics of a multi-end half duplex segment is limited to a connector on a coaxial cable.

2.3 Repeaters

The limit of 500 metres for Thick Ethernet comes from the fact that the signal emitted by some device connected to the Ethernet LAN attenuates, and another device on the LAN must hear the difference between the signal it emits itself and the emitted signal in addition with some signal that arrives from the other end of the LAN. If we would increase the power of a signal when emitted, the signal that arrives from the other end on the LAN would be blasted away. The requirements and functioning of repeaters is discussed in detail in the IEEE 802.3 standard (IEEE, 2002, pp. 178-207).

2.3.1 Repeater

A simple solution to the limit of 500 meters would be to use multiple parts of 500 meters, connected by an amplifier. Such an amplifier then amplifies the incoming signal back to its original power, and emits it on the other segment of 500 meters. The drawback of amplifiers is that they just amplify. This means that

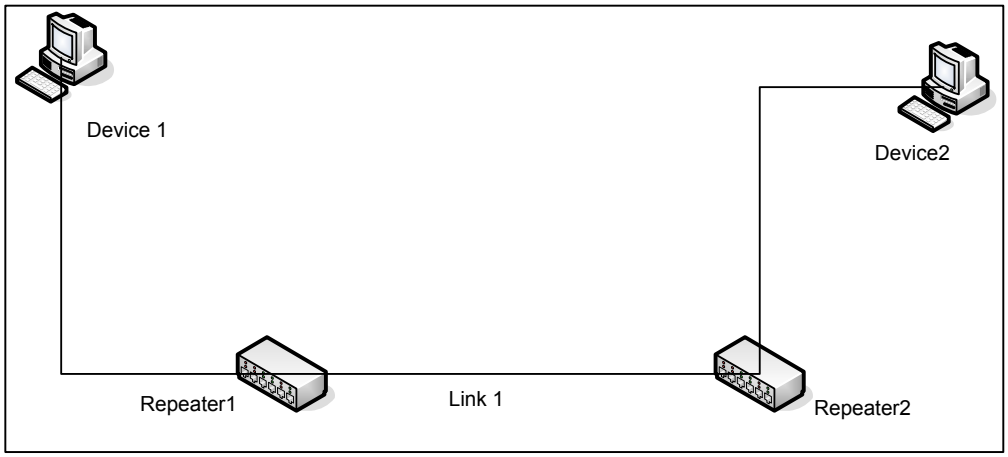

Fig 8: 2 Hubs connected with a Full Duplex Link

distortions during transmission are also amplified. The solution to this problem is to convert the incoming electromagnetic signal back to bits. In a next step, the received bits are retransmitted on the other connected LANs. This is exactly what a repeater does: It receives and decodes data from any segment connected to the repeater under the worst conditions concerning noise, timing and signal amplitude. In the next phase it retransmits this data on all other segments.

The retransmitted signal has amplitude as specified in the standard for the connected segment. The retransmitted signal is also retransmitted within the by the outgoing segment allowed jitter interval. The forwarded data packet is forwarded unaltered, with one exception: If a collision occurs, the repeater detects this and transmits a jam signal on all connected segments. This jam signal has the same properties as the jam signal emitted by any other device.

Of course receiving and forwarding the data consumes some time. The time for receiving and handling is limited to maximally 18 bit times. To this time, the time to put and get the data from the medium by the Medium Attachment Unit (MAU) must be added. The maximal time allowed by the most common 10Mbit MAUs is given in **Table 1**.

In EtherSim, the delay invoked by a repeater can be set in clockticks. Of course this delay is seriously exaggerated, but when working with data, EtherSim wouldn't have any illustrative value. Suppose during

Another problem with repeaters is illustrated in **Fig 8**. Assume this network is built up with Unshielded Twisted Pair (UTP) cables, which are able to transport data in 2 directions simultaneously. Suppose Device1 sends out some data. Repeater1 will forward this data on all of it ports. Repeater2 receives this data via Link1. Repeater2 forwards this data on to Device2, but also on the other pair of the Link1. Suppose the port on Repeater1 connected to Link1 goes immediately back to the receiving state after transmission. If that happens, a part of the data that was just transmitted is considered and treated as newly received data. To prevent this from happening, the port is not monitored for input for a certain time after the transmission. This time must be larger than the sum of the delays on the transmit- and receive-path for the port, and the time must be smaller than 10 bit times for a 10Mbps repeater. This time is called the Transmit Recovery Time.

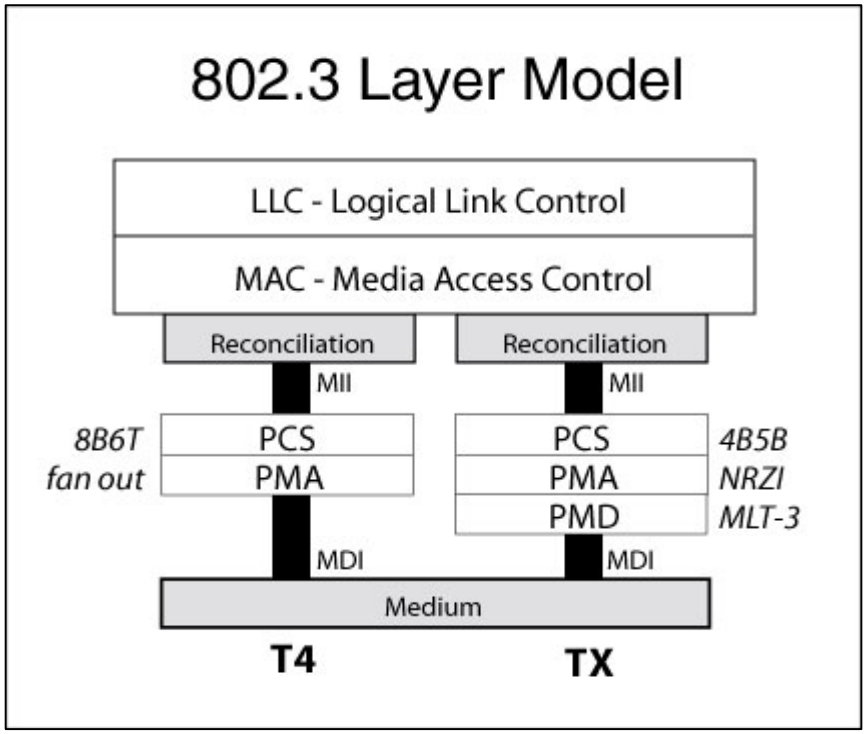

Fig 9: The Physical Layer for 100baseT4 and 100baseTx (IEEE 802.3-2002, 2000 edition)

2.4 Twisted Pair Cables

Because most offices were already wired with UTP cables, it was desirable to use the same cabling for networking too. With UTP cabling, there is no longer any shared coaxial cable, but one Twisted Pair cable running between two bridges, switches or other devices. Twisted Pair cables quickly became the dominant

cabling method for Ethernet networks. The reasons for this are the numerous benefits of Twisted Pair cabling over Multidrop cabling. With twisted pair cabling, it is far more easy to add or remove devices to the network, instead of having to use Vampire taps or BNC connectors, adding a device is done by simply plugging in the cable into the device at one end and into the repeater, bridge or other device at the other end. Also cabling of buildings is much easier when using Twisted Pair cables because they aren't that rigorous as the Coaxial cables. Nevertheless those practical benefits of Twisted Pair cabling over Coaxial cabling, twisted pair cabling has also drawbacks over coaxial cabling. The most important drawback is that because of the lower quality of the twisted pair cables the maximal end-to-end distance is limited to 100m for CAT3 cables and 200m for high-end CAT5 cabling.

Because the evolution of technology never stops, it isn't surprising that the day came that 10Mbps wasn't fast enough any longer. To allow higher speeds, the IEEE committee worked out a faster variant of Ethernet specified in the addendum 802.3u of the 802.3 standard. The common name for this fast variant is "Fast Ethernet". It specifies an operational speed op 100Mbps. Because there was no need for it, Fast Ethernet does not support any multidrop cables, vampire taps or BNC connectors. This means that Fast Ethernet is only allowed over Twisted Pair cables and Optical Fibres. Aside Optical Fibres, both the cheap CAT3 and high-end CAT5 twisted pair cables can be used to build a 100Mbps network. 100Mbps over CAT3 twisted pair cabling is referenced to as 100Base-T4, 100Mbps over CAT5 is referenced to as 100Base-TX and 100Mbps over Optical Fibres is referenced to as 100Base-FX.

2.4.1 Encoding Schemes

An important difference between the 10Mbps and the 100Mbps variants is that Manchester Encoding is used for in all 10Mbps variants while other encoding schemes are used on the 100Mbps variants. The encoding schemes depend on the type of cable used. The reason why Manchester encoding can't be used when working with high-frequency clocks is that, in worst case, it needs 2 transitions of amplitude per bit. If not impossible to produce, such constraints would result in very expensive cabling which would prevent Fast Ethernet from being successful. To avoid this problem, different encoding schemes are used. 100Base-FX (100Mbps over optical fibres) uses the Non-Return-to-Zero Invert-on-one (NRZI) encoding scheme. To decrease the frequency on CAT5 UTP, it uses a variation of NRZI, called Multi-Level-Transition 3 (MLT-3). MLT-3 is also often called NRZI-3. A more detailed description of those encoding schemes can be found in IEEE 802.3 (2002, pp. 56-179) (WildPackets Inc., no date).

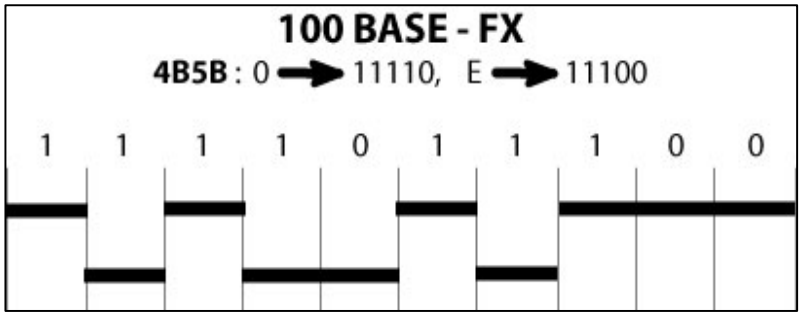

Fig 10: NRZI Encoding of the bits 1111011100 (IEEE 802.3, 2002)

To support these different encoding schemes, a few functional layers are added to the physical layer of the OSI Reference Model. In the 10Mbps variant of Ethernet, the data is immediately handed over from the MAC layer to the PMA (Physical Medium Attachment) sublayer. The PMA sublayer then immediately passes the data to the Medium. In Fast Ethernet, the MAC layer either passes the data to the left Reconciliation when using 100Base-T4 or either to the right Reconciliation when using 100Base-TX. The data goes through the Physical Coding Sub-Layer (PCS) and PMA. Eventually it goes through the Physical Medium Dependent sub-layer before it is put on the medium.

A first measure taken to support 100Mbps networks is NRZI. To encode data using NRZI either one or either no transition is used to encode one bit. While Manchester encoding uses one transition per bit, with NRZI a transition is only present to encode a one. The absence of a transition encodes a zero. An example of the encoding of bitstring "1111011100" using NRZI is shown in **Fig 10**.

NRZI needs significantly less transitions than Manchester Encoding. While in worst case Manchester Encoding needs 2 transitions per bit, NRZI needs only 1 transition per bit. Although fewer transitions are needed per bit when using NRZI, this introduces a new problem. When a large number of subsequent zeros are sent, the clock of the receiver can easily get out of synchronization with the clock of the sender. To avoid this problem, the PCS sublayer encodes the data using the 4B5B translation. 4B5B translation means that every 4 bits of data are replaced with nibbles of 5 bits. A complete overview of the nibbles that are used is given in **Table2**. From this table can be seen that only 16 of the total of 32 available nibbles are used for data. 5 more nibbles are used for control purposes and the remaining 11 nibbles are invalid. The 21 used nibbles all contain at least two transitions, because the 5 nibbles containing only one 1 are all invalid nibbles. By using this encoding scheme, the PCS ensures that there will be enough transitions when the nibble is encoded using NRZI. Another curiosity in the table is the IDLE signal. The idle signal is a sequence of five 1's, which gives the maximal number transitions when encoded using NRZI. The

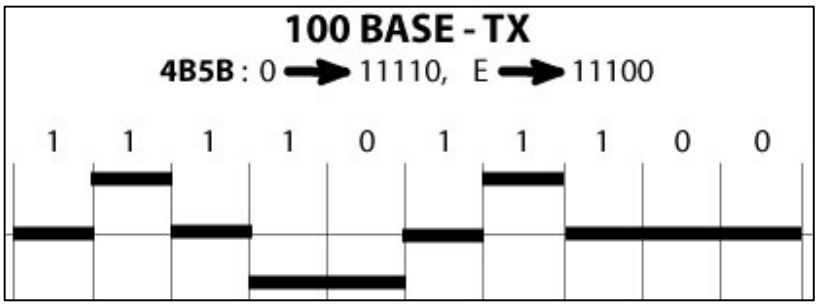

Fig 11: NRZI-3 Encoding for 1111011100 (IEEE 802.3, 2002)

idle signal is continuously transmitted when there is no data to transmit. This idle signal ensures that the clocks of the transmitter and the receiver remain synchronized. For 100Base-FX, the encoding system is implemented as described over here. Now we'll have a closer look at the bandwidth needed to be able to have a 100Mbps network. As the 4B5B data encoding is used, an extra overhead of 25% is needed, this introduces the need for a medium to be able to carry data at the speed of 125Mbps. As NRZI needs only half a wave to send one bit, the optical fibers need to support 62.5MHz to be able to carry data at the rate of 100Mbps.

For transmitting at 100Mbps over CAT5 cables, NRZI-3 or MLT-3 is used. Just like NRZI, MLT-3 represents a bit as the presence or the lack of transition. Instead of alternating between two levels, like in NRZI, NRZI-3 uses 3 levels. By doing this, it becomes possible to encode four bits in one complete sine wave. We can think of MLT-3 as a Stop & Go sine wave, where a zero indicates a "Stop", and a 1 indicates a "Go". The "Go" indicates that the sine pattern should continue. If we assume -1, 0 and 1 as possible values, this means that the output should always obey the following regular expression ((-1)*0*1*)*. The MLT-3 encoding happens in the PMD sublayer, and is added to the NRZI encoding. It is important to note that the third level in MTL-3 is only introduced after the encoding of a bitstring using NRZI. Because more than one twisted pair is available, it is possible to use one pair for transmitting at 100Mbps, while another can be used for receiving at 100Mbps, in other words, it is possible for a network to operate in full duplex mode over 100Base-TX.

For 100Base-TX the same 4B5B scheme is used as in the encoding scheme used for the 100Base-FX scheme. So 100Base-TX also needs an operational speed of 125Mbps, from which 20% is used for the 5^{th} bit in the 4B5B encoding scheme. As it is possible to encode 4 bits on one sine wave, the twisted pair cables used to implement a 100Base-FX network must support clock rates of 31.25 MHz.

	PCS code-group [4:0] 4 3 2 1 0	Name	MII (TXD/RXD) <3:0> 3 2 1 0	Interpretation
DATA	1 1 1 1 0	0	0 0 0 0	Data 0
	0 1 0 0 1	1	0 0 0 1	Data 1
	1 0 1 0 0	2	0 0 1 0	Data 2
	1 0 1 0 1	3	0 0 1 1	Data 3
	0 1 0 1 0	4	0 1 0 0	Data 4
	0 1 0 1 1	5	0 1 0 1	Data 5
	0 1 1 1 0	6	0 1 1 0	Data 6
	0 1 1 1 1	7	0 1 1 1	Data 7
	1 0 0 1 0	8	1 0 0 0	Data 8
	1 0 0 1 1	9	1 0 0 1	Data 9
	1 0 1 1 0	A	1 0 1 0	Data A
	1 0 1 1 1	B	1 0 1 1	Data B
	1 1 0 1 0	C	1 1 0 0	Data C
	1 1 0 1 1	D	1 1 0 1	Data D
	1 1 1 0 0	E	1 1 1 0	Data E
	1 1 1 0 1	F	1 1 1 1	Data F
	1 1 1 1 1	I	undefined	IDLE; used as inter-stream fill code
CONTROL	1 1 0 0 0	J	0 1 0 1	Start-of-Stream Delimiter, Part 1 of 2; always used in pairs with K
	1 0 0 0 1	K	0 1 0 1	Start-of-Stream Delimiter, Part 2 of 2; always used in pairs with J
	0 1 1 0 1	T	undefined	End-of-Stream Delimiter, Part 1 of 2; always used in pairs with R
	0 0 1 1 1	R	undefined	End-of-Stream Delimiter, Part 2 of 2; always used in pairs with T
INV	0 0 1 0 0	H	Undefined	Transmit Error; used to force signaling errors
	0 0 0 0 0	V	Undefined	Invalid code
	0 0 0 0 1	V	Undefined	Invalid code
	0 0 0 1 0	V	Undefined	Invalid code
	0 0 0 1 1	V	Undefined	Invalid code
	0 0 1 0 1	V	Undefined	Invalid code
	0 0 1 1 0	V	Undefined	Invalid code
	0 1 0 0 0	V	Undefined	Invalid code
	0 1 1 0 0	V	Undefined	Invalid code
	1 0 0 0 0	V	Undefined	Invalid code
	1 1 0 0 1	V	Undefined	Invalid code

Table 2: The nibbles used in 4B/5B (IEEE 802.3, 2002)

As CAT3 cabling is only certified for 16MHz, even the MTL-3 combined with 4B5B encoding used for 100Base-TX is not usable for 100Base-T4. To be able to transmit data at the speed of 100Mbps over CAT3, another encoding scheme is used. This scheme combines the 4B5B encoding scheme used in 100Base-X and the MTL-3 encoding used in 100Base-TX. Like MTL-3, 8B6T uses three states on the output line. 8B6T converts 8 bits into 6 tri-state symbols. Unlike the MTL-3 encoding, the subsequent symbols don't have to obey any pattern. Because this constraint is removed, there are 729 possible outputs for 256 possible inputs. It is important to note that 8B6T reduces the effective needed speed, while 4B5B increases the effective needed speed. As 8B6T reduces the number of symbols needed from 8 to 6, to be able to have a speed of 100Mbps, the underlying medium must only be able to carry 75% of the 100Mbps. The fasted waveform required is required when the sine wave must alter between the two extremes. When this happens, only 2 symbols are encoded in one sine wave. This results in a need of a minimal required clock rate of 37.5 MHz. This is still far too high for CAT3 cables.

As CAT3 cables have 4 twisted pairs, it is possible to de-multiplex the outgoing signal onto 3 twisted pairs. This de-multiplexing process is called fanning. As the data needed to be sent is now split into 3 equal parts, each part needs only to support a clock rate of 12.5 MHz. The fourth pair of the CAT3 twisted pairs cable is used to listen for collisions, while the data is transmitted over the three other pairs. As there are only four twisted pairs available, it is impossible for a network to operate at the speed of 100Mbps in full duplex mode over CAT3 cabling.

How are the words generated by 8B6T transmitted? As the minimal Ethernet Packet length is 64 bytes, there are at least 8 words of 6 symbols to transmit. Then 3 pairs are used to transmit, and a word is sent on each pair in a round-robin fashion. The use of the pre-amble is extended in 100Base-T4 networks. It doesn't only synchronize the clocks of the transmitter and receiver, but also indicates the pair access scheme that is going to be used. If there was no agreement on the pair access scheme, it would be impossible to re-assemble the frame correctly at the receiving side.

Although 100Mbit is not allowed over coaxial multidrop cables, EtherSim nevertheless provides 100Mbit half duplex links. Half duplex links at 100Mbit even make sense, as they can be thought of as CAT3 cables. EtherSim also provides full duplex links, which can be thought of as CAT5 twisted pair cables or optical fibres. Although it could have been possible to build a 100Mbit bus network, this has never been standardized. Nevertheless, in EtherSim it is possible to simulate a 100Mbit bus-network.

2.5 Bridging

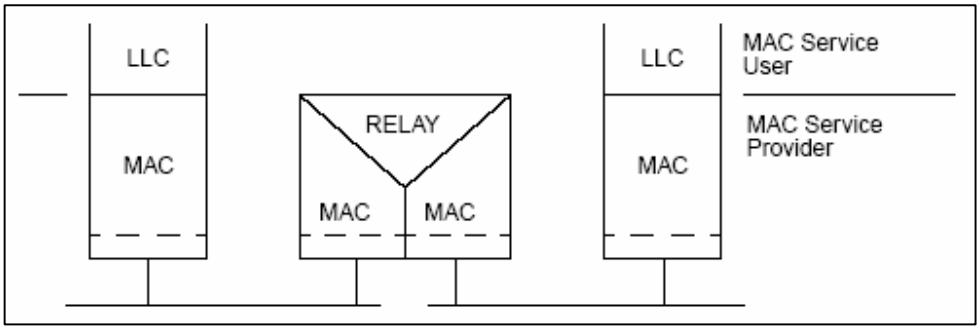

Fig 12: The operation level of the Bridge (IEEE 802.1D, 1998)

A bridge can be used to connect MAC networks of all types. These networks don't need to use the same MAC, but it is the task of the bridge to interconnect the different LANs transparently. For two or more devices on separate LANs, which are interconnected through a bridge, it should seem as they were both connected to the same LAN. A bridge operates under MAC sublayer of the Data Link Layer. Although the presence of bridges is nearly transparent to the higher level, it isn't completely transparent: Differences in Quality of Service provided by the MAC sublayer can arise when switches are used.

It is the task of the Bridge to relay and filter frames between the separate MACs connected to the Bridge. As can be seen from the image **Fig 12**, the switch operates completely in the MAC sublayer of the OSI Reference model.

2.5.1 Spanning Tree Bridges

In many organisations, it is common to interconnect LANs using two ore more bridges in parallel. This is often done for reliability issues: If one bridge or one component from a particular to a bridge fails, the other bridge can take over. Although this construction increases the reliability of the network, it has also a drawback. This issue is illustrated with picture **Fig 13**: Suppose Workstation1 sends a frame to Workstation2, but neither Bridge1, nor Bridge2 know about WorkStation2. Brdige1 will receive the frame forwarded by Bridge2 and vice versa. So Bridge1 and Bridge2 receive a datagram for Workstation2 on its port connected to LAN2. As each of these doesn't know about Workstation2, they each forward the frame onto LAN1. So Bridge1 and Bridge2 will continue forwarding the same frame indefinitely.

The ro is
lowest se l n
Cost. Th oc
root via le
for that L N a

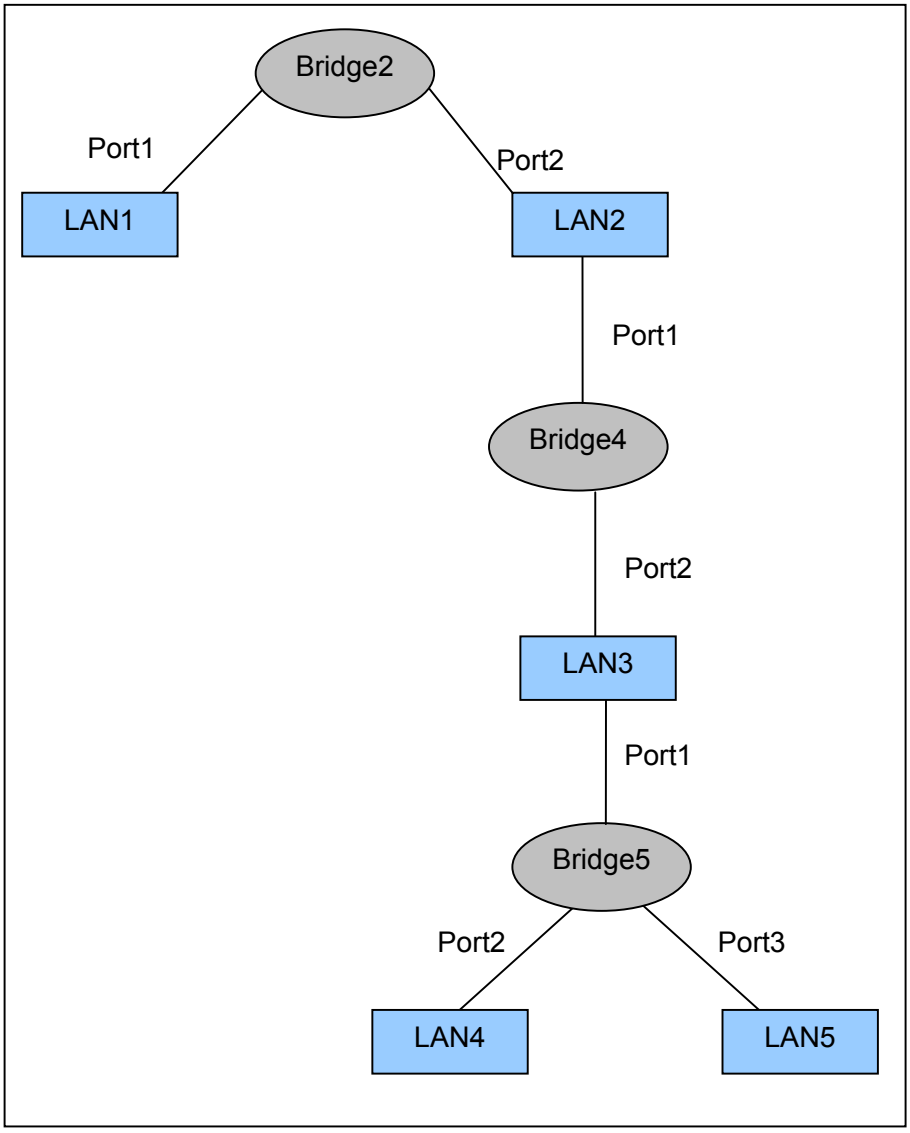

Fig 15: The Active topology for the given example topology

is then the port via which the configuration Messages arrive at the bridge.

To select the next bridges in the tree, for each LAN the Bridge Port with the lowest Root Path Cost is chosen. In case more than one Bridge Port has the same Root Path Cost for the same LAN, the same mechanism for selecting the root is chosen: The one with the lowest serial number is selected. Internally the bridge performs the same algorithm, as it might be necessary to select one of the ports of one particular bridge as the designated port for a LAN connected to the bridge. As the Path Cost, Bridge

2.5.1.3 Changes In Topology

The Spanning Tree algorithm as it is now cannot cope with changes in network topology. If for example the designated port for a LAN is disabled or breaks down, no other port will take over the function of the Designated Port. The same holds for the root bridge, or any other bridge on the least cost path between a particular LAN and the root: if it breaks down, no other bridge will take over its function, although it might be possible to rebuild the spanning tree in such a way that the network remains fully functional. With the current configuration, placing redundant bridges is only a waste of money, as the desired gain in reliability isn't reached.

To be able to cope with changes in topology, the topology information propagate through the network has a limited lifetime. This can be achieved by transmitting along with the information the age of the information. So each bridge stores the information from the designated port of LAN connected to each of its ports, together with the age of that information. As the root emits Configuration messages at regular intervals, the age is regularly updated and should never expire. If the information times out on a certain port, not being the designated port for that LAN, the given port will attempt to become the designated port for that LAN. As the port assumes to be the designated port for that LAN, it will send a Configuration Message containing the root information of the bridge to which the port belongs onto that LAN. The normal selection procedure for selecting the designated port for a LAN is started.

If the root-port times out, another port will be selected as root port. The information for LANs for which the current bridge is the designated bridge will then be updated with new root information, and if necessary, a new designated bridge for a particular LAN will be selected. In case no information from the current root remains, for example when it has become unreachable, the bridge will claim to the root itself, and thus act as the root. If the root has effectively changed, the other bridges' root information will also time-out and the best successor will be seen as the root. It might on the other hand also be the case that the root didn't break down effectively, but that the because of a reconfiguration, the path cost increased. When this happens, the new configuration messages will be discarded as inferior. When the timed-out bridge then claims to be the root, the neighbour bridges will immediately react and the situation is rectified quickly.

So in either case, the changes in the active topology are quickly propagated throughout the network. When the root bridge is effectively down, the network is reconfigured as in the original algorithm when the time-out expires. In case a bridge falsely assumes the root is down, at least one of its neighbours will

respond to the configuration message emitted by that bridge with a configuration message containing the correct information, so the correct information is quickly propagated through the network.

As all bridges should share a common understanding when information should time out, the time-out value is transmitted in all configuration messages originated at the root. To minimize the chance that the network is reconfigured because of the loss of configuration messages, the time-out value includes multiple times the interval between the emissions of configuration messages. Because of this, a false reconfiguration of the network will only happen if several subsequent configuration messages are lost. Besides this, also the time needed for propagation of information throughout the network, i.e. the propagation delay of the BPDU between sending and receiving, is taken into account.

In practice a root bridge usually sends out a configuration message every 4 seconds. The aging time for information in Cisco bridges for example is 5 minutes. As 5 minutes is quite long in rapidly changing environments, an important improvement to reduce the reaction delay has been developed. When a bridge detects a topology change, for example a link that goes down, or a port that gets enabled, it sends out a Topology Change Notification on its root port. A Topology Change Notification is a special BPDU, used to notify the root about changes in the Network Topology. When a bridge receives such a Topology Change Notification, it sends a Topology Change Notification itself on its root port. When the Topology Change Notification hits the root bridge, the root bridge sends out a configuration message on all its ports, with the Topology Change bit set. So all bridges in the network know the topology has changed, and they reduce their time-out value. For Cisco bridges, the time-out is reduced to 15 seconds, instead of 5 minutes. Thanks to these measures, the network can react faster to topology changes, than if it waited until the information has timed out.

2.5.1.4 Implementation in EtherSim

This algorithm has been implemented in EtherSim project. It is used by the switches, which are just multi-port bridges. As the interval between 2 successive broadcasts of Configuration Messages performed by the root bridges isn't specified in any standard, this is configurable by the user. The delay can be set in clockticks, where one clocktick corresponds to the time needed by an electromagnetic signal to travel the distance of one 1 LinkSegment. As the length of one LinkSegment in meters is also configurable by the user, and the material of the LinkSegment is unspecified (copper, fiber …) the real time interval can be calculated as follows:

- The signal propagation speed in copper is about *200 000 000 meters/second*.

- The length of a LinkSegment is *l meters*.
- One clocktick corresponds to *s seconds.*

$$l/s = 200\ 000\ 000$$

This tells us that with the default settings of *l=200m*, one clocktick corresponds roughly to 1/1 000 000 seconds. If we would simulate an interval between two broadcasts of configuration message from the root to correspond to 5 real seconds, we should set the interval to 5 000 000 clockticks. With the duration of one clocktick set to one second, this results in a simulation time of *57 days 20 hours 53 minutes 20 seconds*. In order to give EtherSim any illustrative value, the interval between to broadcasts is set by default to 60 clockticks. So the ratio of Configuration Messages is about 83000 times higher than in reality. This is the reason why it is possible to turn off the option to build the spanning tree in EtherSim. Care must be taken to not use an interval that is smaller than the time needed to transmit a BPDU, because if this would be the case, the network would get congested. It must also be taken into account that it might be needed to transmit more than one BPDU during one interval, for example as a reply on a Configuration Message with worse information. If the broadcast interval is too small, such a reply will double, or maybe even triple the time needed for a BPDU to arrive at some other remote bridge. In the meanwhile, the rootport of the remote bridge might have timed-out, resulting in even more BPDU on the network. A rule of the thumb is to set the delay at least four or five times the time needed to transmit a BPDU.

To build the spanning tree, special messages are sent over the network. These messages are sent using BPDU, which means they can be received by blocked ports of bridges. The messages carry the cost to access the root from the previous bridge, the cost to receive data from the LAN onto which the configuration message was sent and the port through which the Configuration Message was sent. As the BPDU are sent over an Ethernet network, they also follow the Ethernet standards concerning packet length etc.

Although in reality the bridges broadcast immediately when they are powered up, the bridges in EtherSim don't. This is done by design: The user is given a chance to build the –complex- network topology before the bridges begin configuring themselves. When a bridge receives a Configuration Message there are some possibilities:

- The Configuration Message contains a better root, so the rootport, root and root path cost are changed accordingly in the receiving bridge. The Total Root Path cost in the Configuration Message is updated and then flooded on all ports for which the bridge thinks it has the designated ports.

- The Configuration Message contains a root with a higher Priority ID. As a reply the bridge sends a Configuration Message containing its own root and the related information.
- The Configuration Message contains a lower cost to the root. So the root port is changed to the port on which this message was received and the cost to the root is updated. The Total Root Path Cost in the Configuration Message is also updated and flooded on the designated ports of the bridge.

The bridges in EtherSim are also able to cope with changes in topology. Each blocked port must hear a Configuration Message within a certain time interval, expressed in ClockTicks. This Configuration Message indicates that there is a designated port for the LAN connected to the given port. If a blocked port times out, the timed out port will take over the rule of designated port. If some other port connected to that LAN hears the Configuration Message sent out by the timed out port, it will reply with its own parameters.

Beside a port, the complete bridge can also time out. This happens if it didn't receive any information on its root port before the old information timed out. As the shortest path to the root can't be through a designated port, a blocked port is selected as rootport. As the IEEE 802.1D standard does not specify which port should take over the role of rootport, we decided to take the last blocked port on which a BPDU arrived. The reason for taking this port is as follows: as a time-out indicates a topology change, the LAN on which the most recent information was received has the highest chance to be either not involved in the topology change, or to have already adapted to the new situation.

To have a common understanding about when a port or bridge should time out, the time-out value is included in each Configuration Message sent by the root. So it is the root who decides when the information is invalidated. Of course the time-out of bridges should be larger than the interval between the broadcast of Configuration Messages. To be able to cope with lost messages too, the time-out interval should be at least three times the broadcast interval. As the time-out of blocked port is dependent on the same parameters, we use the same mechanism to time-out blocked ports.

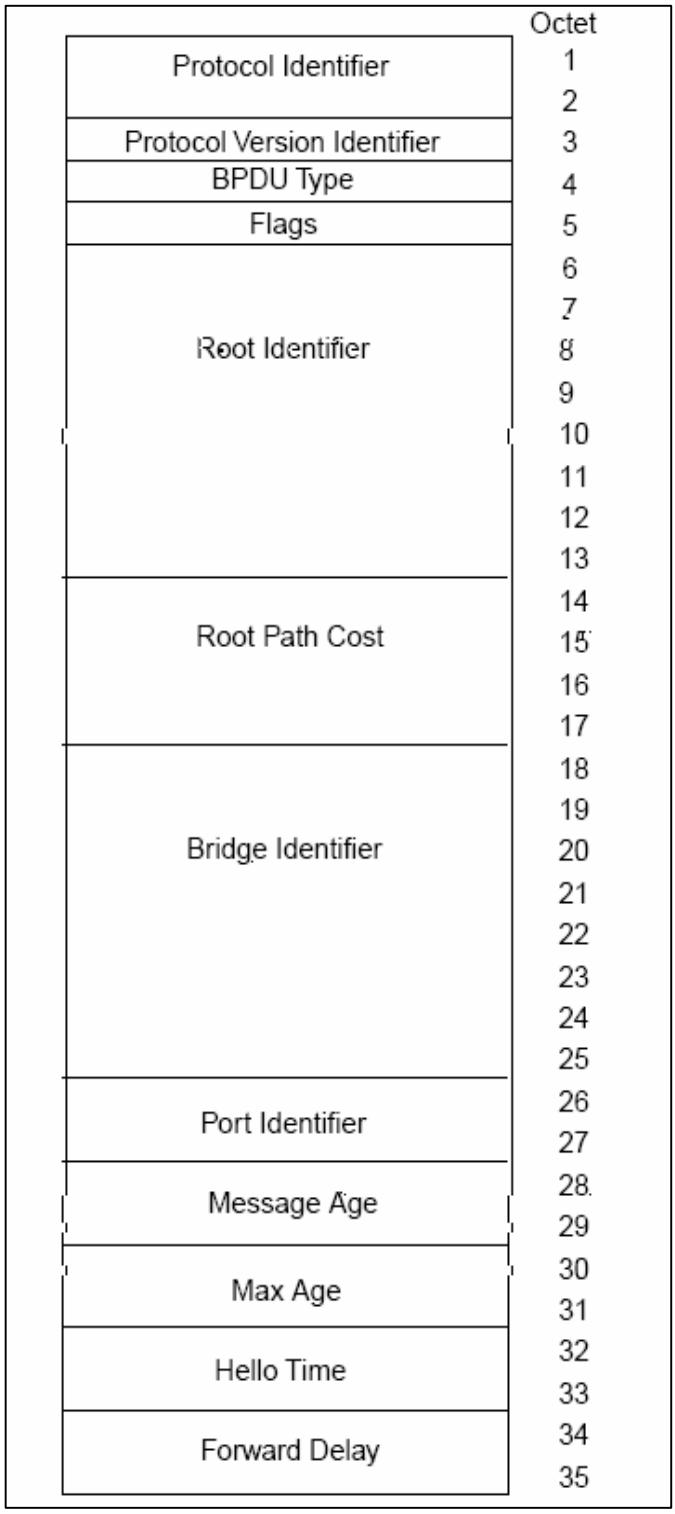

Fig 16: The Configuration BPDU (IEEE 802.1D, 1998)

2.5.2 Bridge Protocol Data Units (BPDU)

Communication between bridges happens through Bridge Protocol Data Units. Bridge Protocol Data Units contain information to configure bridges in a bridged network. They are especially used to remove loops in a network topology. Like other frames, BPDU are not immediately forwarded when they arrive at a bridge. Instead the bridge might use the information provided in the BPDU to build its own BPDU. Care must be taken to not confuse a Configuration BPDU with a Configuration Message: A Configuration Message expresses the propagation of configuration information throughout the bridged network, while a Configuration BPDU only exists between ports of bridges, attached to the same LAN.

It is required for a BPDU to contain an integral number of octets and the first two octets contain a Protocol Identifier. If the octets are ordered increasingly and if the first bit in an octet is the lowest-order bit, no further restrictions are placed on the use, encoding or structure of the BPDU with different values for the Protocol Identifier.

2.5.2.1 Configuration BPDU

The Configuration BPDU is the BPDU used by the Spanning Tree Protocol (STP) to build the spanning tree of bridges in a bridged network. The structure of the Configuration BPDU is given in **Fig 17.** BPDU Type indicates the type of BPDU, for example “Topology Change”.

The Flags field contains several flags for a certain BPDU. A particular flag corresponds to a particular bit-position in the Flags field. Which position maps to which field, depends on the type of BPDU.

Timer values are encoded in units with a length of 1/256 seconds. This gives an accuracy of 1/256 s and a range of 0 to 256 seconds, the 256^{th} second excluded. The fields Message Age, Max Age, Hello Time and Forward Delay in the Configuration BPDU are such timer values.

The Max Age filed contains the maximal age received protocol information can have. If its age exceeds this maximal age, the information is discarded. This parameter is set by the bridge that constructs the BPDU.

The Hello Time is the time interval between two broadcasts of a Configuration Message by the root bridge. This parameter is dependent on the bridge that is currently considered as the root bridge.

The Forward Delay is the time spent by a port between the Blocked and the Forwarding State or the time a port spends in the Listening and Learning state.

2.5.2.2 Topology Change BPDU

The Topology Change BPDU is used to propagate the notion of an altered topology through the network. These BPDU are sent from the bottom up: from the bridge that detects the change in topology to the root bridge in the active topology. As the BPDU should only contain the message "The topology has

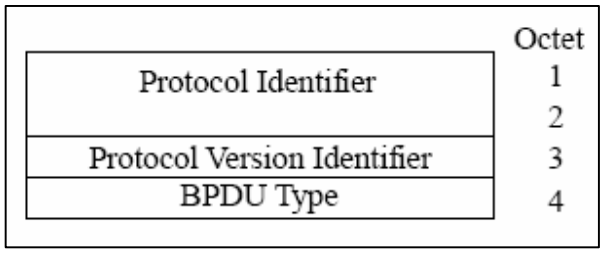

Fig 17: Topology Change BPDU (IEEE 802.1D, 1998)

changed", the Topology Change BPDU is a very simple BPDU. It's structure is given in **Fig 12**.

The Topology Change Notification BPDU has a very simple structure. As each BPDU it has a Protocol Identifier. Next it has an identifier for the version of the protocol used and a BPDU type field. This type field contains the value for a Topology Change Notification, so that when a bridge receives such a BPDU, it knows that the topology has changed and some action must be taken in response. No more information is needed to signal a change in topology.

2.5.3 Changing the State of a port

Of course the propagation of the Configuration Message through the network involves some delay. So a quick transition of a port state could cause some problems: If a port was not involved the Active Topology, and it changes to the Forwarding State immediately, this can cause temporary loops. As the BPDU are also sent over Ethernet, the BPDU cannot be exchanged when some data is sent over the LAN. For this reason, it is desirable to wait with forwarding data onto a certain LAN until all ports of bridges connected to that LAN are configured properly. In concrete this means that a bridge should wait to forward data until the topology information has propagated through the network. In addition, all old topology information should be timed out also. As the topology has changed, there is a high probability that the information concerning the stations isn't valid any longer either. This means that before a port changes into the Forwarding State, it waits for better information, which indicates that the port should return into the blocking state. After this, the port already listens for data information, and learns about

which stations are accessible through the port, but still doesn't forward any data. During this period, the port is in the "Learning State". After the expiry of some time-out, the port changes its state from "Learning", to "Forwarding". In the forwarding state, forwarding of frames and station location learning are both enabled. For simplicity, EtherSim does not use these changes, but changes the state of some port immediately.

A variant of the STP protocol, called Rapid STP (RSTP) also exist. It is specified in the IEEE standard

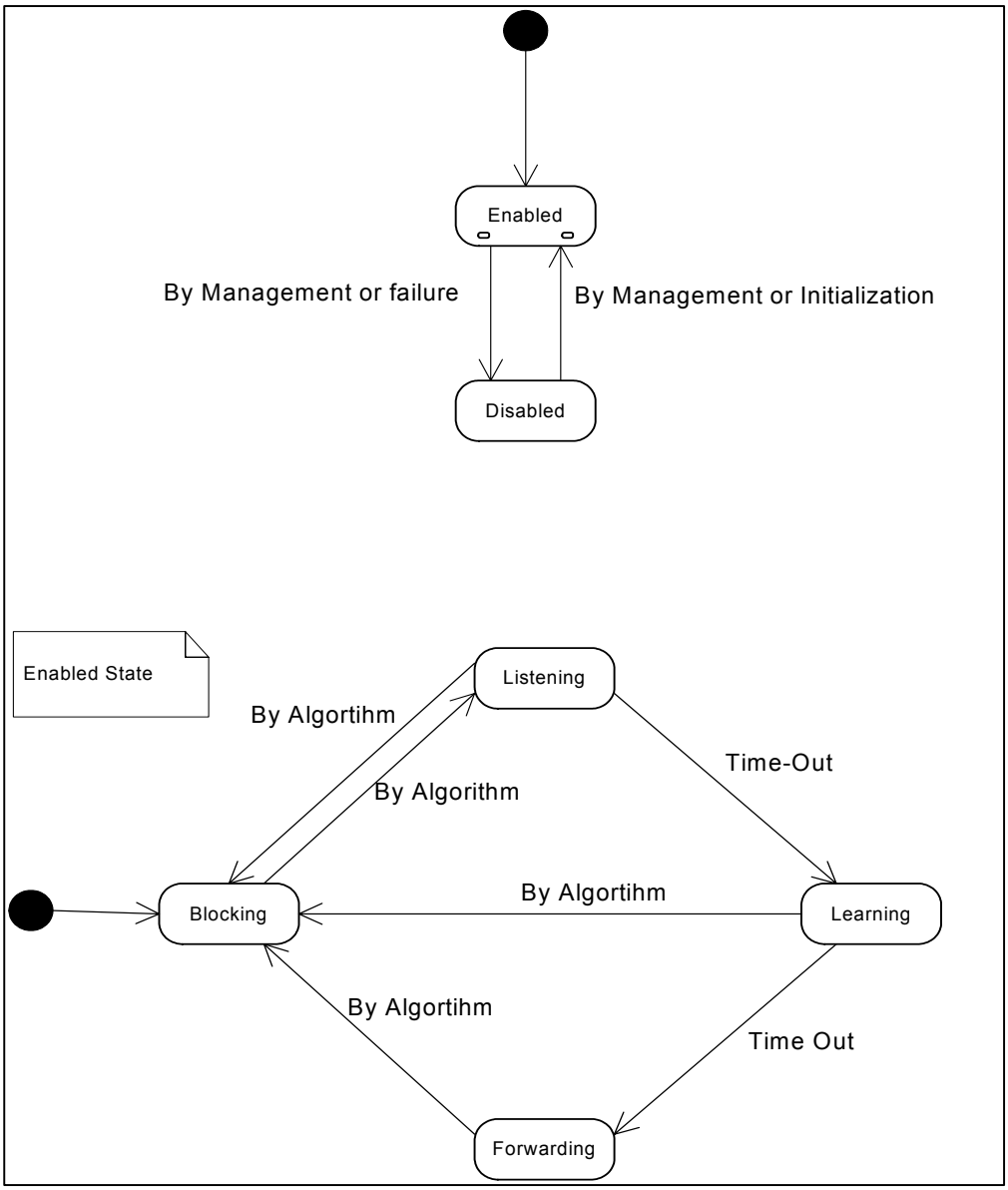

Fig 18: State Diagram for port changes of a bridge

802.1W and allows faster convergence. The gain in speed can be achieved because in some cases it is allowed to change the port state from blocked immediately to forwarding. A major constraint is that the bridges must be connected by point-to-point links. Also reaction to topology happens faster, because the Topology Change BPDU is forwarded on all forwarding ports. By this, all bridges are faster notified of the change in topology. The RSTP protocol is specified in the IEEE 802.1W standard.

Chapter 3
Structure of EtherSim

EtherSim is a program written in C#, it has been designed to make it possible to illustrate rather difficult concepts on a graphical manner. To represent the progression of the data over a link, we built a system based on discrete time. The propagation of data through the different components is implemented as an event-based system. This chapter describes the architecture of EtherSim in detail.

3.1 Overall Structure

Although the purpose of the program is to provide a graphical representation of network operation on the Medium Access Control Sublayer of the Data Link Layer in the OSI Layered Reference Model, the interface part is separated from the underlying logics for each component available. The components are for instance a Full Duplex Link Segment, a Half Duplex Link Segment, a Hub, a Switch and a Computer. The operational part of these components is strictly separated from the graphical representation. This makes it possible to easily build another representation on top of the underlying components. As the link between the components and their representation is made using the Observer design pattern (Gamma et al., 1995, pp.293-303), it is even possible to have multiple representations running simultaneously for the same network configuration.

The hierarchy of the Switch, Workstation and Hub is given in **Fig 19**. Changes in the subjects themselves are recorded in the State object. The State object then raises an event when one of its variables is changed. This event is kept by the AbstractSubject, which then orders the Observers to update themselves according to the new state. When an observers orders something, this immediately applied to the object, for example the WorkStationObserver will immediately notify the WorkStation to start sending when the user used the send-packet procedure. Those orders don't go via the State object because they don't affect the state, it are just orders which have to be executed.

3.1.1.1 Transceivers

The Hierarchy encapsulates related functionality in the same class. Although there is no other subclass of NetworkComponent than PacketReceiver, they are still in different classes. The reason for this approach is that this approach makes it possible to create new NetworkComponents that aren't PacketReceivers. To make this clearer, we'll have a word about the NetworkComponent and the

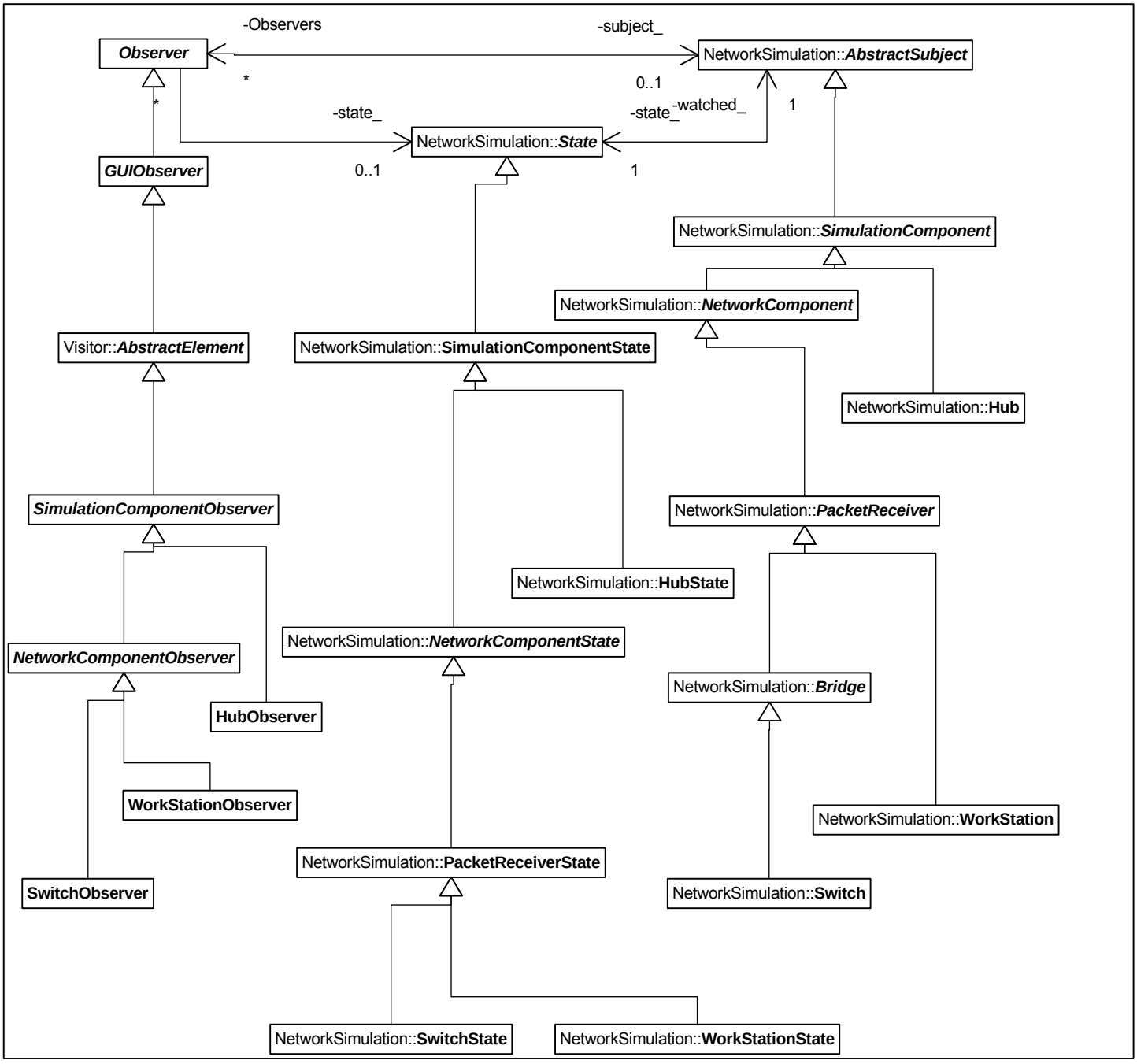

Fig 19: Hierarchy of EtherSim

PacketReceiver. The NetworkComponent provides functionality to manage Transceivers. A Transceiver is the interface between a LinkSegment and a device. It implements the notion of Transceivers in the original Ethernet, but also corresponds to a port in hubs and switches. When a LinkSegment is connected

to a NetworkComponent, a new Transceiver is created and associated with that LinkSegment. When the LinkSegment is disconnected, the Transceiver is removed. It is also important to be aware of the fact that two types of Transceivers exist. The HalfDuplexTransceiver listens on the medium before transmitting. If the HalfDuplexTransceiver hears that a transmission is already going on, it waits until the Transmission is over. The FullDuplexTransceiver on the other hand, doesn't sense the medium first but transmits immediately.

The HalfDuplexTransceiver should be used in combination with HalfDuplexLinks and with full duplex links that are connected to hubs. FullDuplexTransceivers should only be used in a switched or point-to-point topology with FullDuplexLinks. FullDuplexTransceivers don't sense the medium before sending because they assume that the network operates in full duplex mode and thus they assume it is possible to send and receive simultaneously. In reality, HalfDuplexTransceivers should be used with CAT3 cabling at 100Mbps, multidrop cables and any cabling in combination with hubs. FullDuplexTransceivers should only be used with a switched network using CAT5 or optical fiber cabling. An overview of the Transceiver is given in **Fig 20**.

Apart from the HalfDuplexTransceiver and the FullDuplexTransceiver, a third type of Transceiver also exists. This transceiver is the RepeaterTransceiver. The RepeaterTransceiver is a transceiver designed to be used with repeaters, present in EtherSim under the name of Hubs. The RepeaterTransceiver is always a HalfDuplexTransceiver in this sense that the RepeaterTransceiver always first senses the medium before it begins transmission. The reason for this is clear: with multi-port repeaters the network operates always in half duplex mode. So far, it we could have used the already available HalfDuplex-Transceiver, but there important differences. The RepeaterTransceiver for instance emits a noise jam when it detects a collision, but it doesn't retransmit the data for the simple reason that it doesn't have access to the data to be retransmitted. The Repeater simply forwards the bits that come in whithout interpreting what it forwards. Another feature that is specific for the RepeaterTransceiver is that the RepeaterTransceiver blocks after a transmission. The reason for this is explained in the section about repeaters.

3.1.1.2 PacketParts

When talking about data transmitted in the Data Link Layer of the OSI Reference model, we talk about frames. To be compliant to the IEEE 802.3 standard, an Ethernet frame must have a length between 64 bytes and 1518 bytes with the preamble excluded. Such a frame is represented by several PacketParts, which all make part of some Packet. The number of PacketParts to represent one frame depends on the

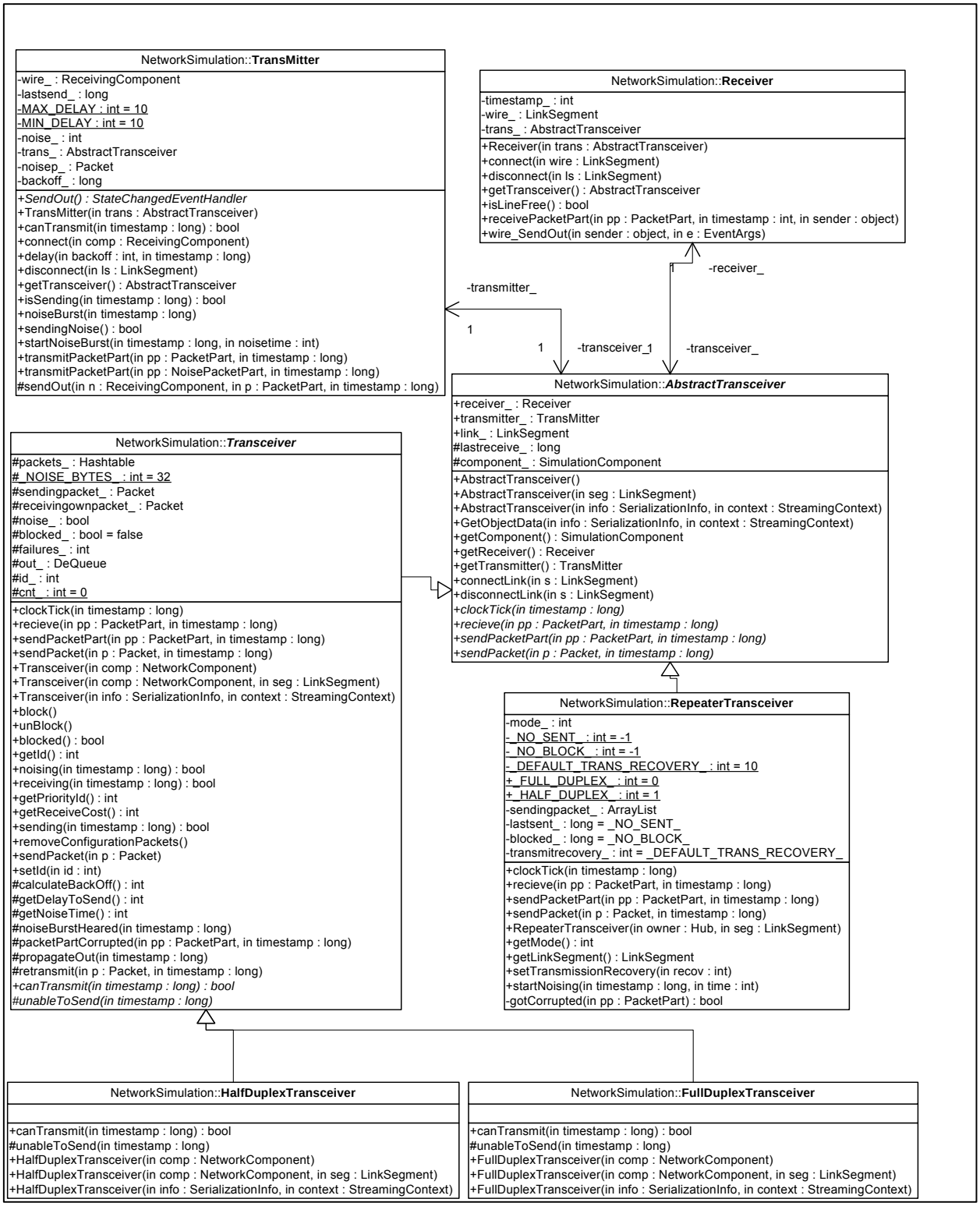

Fig 20: Structure of the Transceivers

current speed of the simulated network, the length represented by one LinkSegment and the length of the frame to be transmitted. All those PacketParts are sent one by one, while the first PacketPart is marked as a header. By using this header flag, the receiving side can identify separate frames from the incoming stream of data. The PacketParts all carry the color of the device that sent the frame, the destination of the frame, the source of the frame and the Packet to which the PacketPart belongs. In normal circumstances the PacketPart belongs to one Packet, but if the PacketPart got corrupted with some other PacketPart, the PacketPart acts as it belongs both to the Original Packet and to the Packet with which the current PacketPart got corrupted. So if the PacketPart belongs to more than one Packet, the receiving side knows that the PacketPart is corrupted. When a Transceiver detects that it received a corrupted PacketPart, the Transceiver checks if it is still transmitting one of the Packets to which the current PacketPart belongs. If this is the case, the transceiver sends out a noise jam. Sending out a noise jam is done by the FullDuplexTransceiver, the HalfDuplexTransceiver and the RepeaterTransceiver. Retransmitting the failed frame after this noise jam on the other hand is done by the FullDuplexTransceiver and the HalfDuplexTransceiver. Before the frame is retransmitted, the transceiver waits some random time, according to the Binary Exponential BackOff Algorithm, described in the section about Ethernet.

3.1.1.3 Transceivers

Transceivers are used to connect a device to a link. A Link is build up of parts, called LinkSegments. The LinkSegments exist in two variants, the HalfDuplexLinkSegment and the FullDuplexLinkSegment. As the name says, the HalfDuplexLinkSegment provides a half duplex link, such as a multidrop coaxial cable or a CAT3 cable at 100Mbps. The FullDuplexLinkSegment then provides a full duplex link such as a CAT5 cable or optical fibers. The difference between the FullDuplexLinkSegment and HalfDuplexLinkSegment is that the data on a FullDuplexLinkSegment only gets corrupted with other data on the FullDuplexLinkSegment if the data arrives on the FullDuplexLinkSegment from the same data as the data that is already on the FullDuplexLinkSegment arrives from. So if data arrives on a FullDuplexLinkSegment from opposing sides, no collision occurs. The FullDuplexLinkSegment can be thought of as a cable having a twisted pair dedicated for each end.

The HalfDuplexLinkSegment can’t contain more than one PacketPart simulateously whitout corrupting each other. So if one PacketPart arrives on the HalfDuplexLinkSegment from one end and simultaneously some other PacketPart arrives from another end onto the same HalfDuplexLinkSegment, both the PacketParts are corrupted. The HalfDuplexLinkSegment supports a variant which has more than 2 ends, namely a three-end HalfDuplexLinkSegment. This variant of the HalfDuplexLinkSegment represents the

notion of a vampire tap in the Thick Ethernet context or a BNC connector in the Thin Ethernet context. Of course the corrupted data must also be forwarded, so PackatParts can only be discarded if they corrupt each other and move in the same direction. This happens if more than one PacketParts are placed simultaneously onto the same LinkSegment.

3.2 The time-triggered architecture

'We can model the progression of time as a directed line extending from the past into the future. Events are then defined as occurrences that happen at some cut of this time line' (Kopetz, 1997, pp.45-69). We can now define a trigger as an event that causes the start of some action. Now we have the notion of events, we can define the time-triggered and event-triggered approach. In an event-triggered system, any action is a response to the occurrence of some event other than the regular event of a clock tick. In a time triggered approach on the other hand, any action is performed at predetermined points in time.

3.2.1.1 ClockTicks

In some sense, EtherSim can be thought of as a Time Triggered system. Of course building a network topology is a completely event-triggered aspect, but the behaviour of the network can be seen as time-triggered aspect. EtherSim makes use of one centralized clock, which is included in a Simulation object. All SimulationComponents are then notified of the progression of time and carry out the necessary actions. The `clockTick(long timestamp)` method available in each SimulationComponent is used to notify the SimulationComponent about the occurrence of a microtick. Before we can continue, we need some definitions (Kopetz, 1997, pp. 45-69):

> *"A clock is a device for measuring time. It contains a counter, and a physical oscillation mechanism that periodically generates an event to increase the counter. The periodic event is called the microtick of the clock. The granularity of a clock is the duration between two microticks."*

> *"Assume an omniscient external observer who can observer all events that are of interest in a given Context. This observer possesses a unique reference clock z with frequency f^z which is in perfect agreement with the international standard of time."*

> "*The granularity g^k of a given clock k is given by the nominal number n^k of microticks of the reference clock z between two microticks of this clock k.*"

Just like almost all programming languages, C# provides a timer class. This class implements in fact a part of the clock. It provides the periodic microticks and has a modifiable granularity. It is this clock that we use to control the time-triggered aspect of EtherSim. It is possible to change the granularity of this clock, of course between certain boundaries, but these boundaries are largely sufficient for our purposes. The clock is used to make the frames moving through the network. If we would set the granularity of the clock to small, the next microtick would come before the action associated with the current microtick has been carried out. Although future, more powerful computers might be able to cope with this problem, another problem would remain: the frames would progress that fast through the network that a human could hardly observe anything, which would make the program useless. For this reason we placed even more constraining bounds on the minimal and maximal granularity.

The possibility to change the granularity of the clock is a useful feature. This feature gives us the possibility to make it possible to the user to change the speed of the simulation according to his needs. The granularity of the clock can be specified in milliseconds, thus we can say that the granularity of the reference clock g^z is equal to one millisecond. As one millisecond is far too fast for a human to observe a change, it is clear that the lower bound of one millisecond possible for the granularity a sufficient for our purpose.

3.2.1.2 Reactions To Clockticks

What are the actions that are executed at the event of a microtick? As mentioned earlier, all SimulationComponents are notified of the occurrence of the microtick by giving them the new value of the timer counter. The LinkSegments are the last SimulationComponents that are notified about the progression of time. The reason for this will become clear later on as we will discuss what action each SimulationComponent takes as a reaction on the occurrence of a microtick.

The first SimulationComponent we discuss is the WorkStation. The WorkStation can be used to transmit and to receive frames. The WorkStations are end-points of the network and don't really participate in the functioning of the network, they just use it. As the WorkStation is a NetworkComponent, it uses a Transceiver to interface with the network itself. When the WorkStation transmits a frame, a Packet object is created. This Packet object is passed to the Transceiver which queues up the PacketParts making part of the given Packet. At the occurrence of a microtick of the central clock, the WorkStation notifies the Transceiver of this event. If the outgoing queue of the Transceiver isn't empty, it tries to transmit the PacketPart. It isn't sure that the Transceiver will be able to transmit the PacketPart, because the Transceiver could be backing off or the medium might be busy.

A Hub receives an electromagnetic signal, converts this into bits and retransmits this. By doing this conversion, the noise induced during transmission from the previous device to the hub is filtered out. Because EtherSim doesn't model the physical layer, it is impossible that noise is induced during transmission, and electromagnetic signals don't exist either. So the function of the hub is to distribute a PacketPart that was received on some line over all outgoing lines. Till there, there is no need for an action in response to the occurrence of a microtick, but in reality, the electronics in a hub introduce some delay. This is the point where the occurrences of microticks play a role in the hub.

The Hub uses a Delayer object, which is a kind of queue to simulate the delay induced by electronics in real networks. It is possible to enqueue items on the Delayer, but these items can only be dequeued after a specified duration. When the Delayer is notified that a microtick occurred, it removes the objects that have been on the queue for the specified duration. It is possible to dequeue multiple objects at the same microtick. So at the occurrence of a microtick, a Hub notifies its Delayer object of this occurrence. The returned PacketParts are then forwarded on all ports, represented by Transceivers, except the incoming one if the incoming port is half duplex link.

The last device is the Switch. A Switch receives complete packets, just like WorkStations. This is the reason why they inherit from the PacketReceiver class. The PacketReceiver class provides the functionality to receive complete packets instead of PacketParts. The Switch also uses a Delayer object, because in reality a switch also induces some delay. This Delayer object is used like the Delayer object in the Hub with the logical difference that the Delayer object stores complete Packets instead of PacketParts. Because a Switch is quite intelligent, it only forwards the packets on the line where it knows that the receiver is located. If the Switch can't locate the receiver, the data is forwarded on all ports except the incoming one. To be able to locate the receivers; the Switch uses auto-learning, as explained in the section about bridges. So at the occurrence of a microtick, the Packets that have been delayed long enough are removed from the Delayer. Next those packets are forwarded on the appropriate lines, but the Transceivers used by Switches are exactly the same Transceivers as those used by WorkStations. By these, the Transceivers of the Switches implement the Carrier Sense, Collision Detection and Binary Exponential Back-Off exactly as the WorkStations do.

The Switch also carries out another action at the occurrence of a microtick. If the Spanning Tree Protocol is enabled, the occurrence of a microtick is used to check if the time-outs haven't expired. At each occurrence of a microtick of the central clock, the value indicating the time remaining before the time-out expires is decreased by one. If the value hits zero, the time-out has expired. As explained in the

section about bridges, a bridge has a time-out on the rootport and on the blocked ports. The bridges that assume they are root bridges also use a time-out to transmit a Configuration Message, announcing they are the root at regular intervals.

3.2.1.3 The Event-Triggered aspect of EtherSim

dvic

transmitting device will only receive data when the device is notified about the occurrence of the microtick, the graphical representation doesn't run smoothly, but that there will be temporarily gaps between the LinkSegments and the device if the LinkSegments are notified about the occurrence of a microtick. The result is that the data currently on the LinkSegments moves away from the location where the transmission started.

3.3 User Interface

Till now, we have discussed what's under the hood of EtherSim. Now it's time to have a look at the external part of EtherSim. EtherSim User Interface (UI) contains 3 zones: a Canvas, a WorkSpace and a command output. The Canvas contains Prototypes (Gamma et al., 1995, pp. 117-126) of the Observers to be used. When the program is loaded, the several Observers are constructed and placed on the Canvas. When the user wants to use one of the components for which an Observer is shown on the Canvas, he can just drag this Observer onto the WorkSpace. When such an observer is dragged onto the WorkSpace, it is cloned and an underlying object is constructed. For instance, if the user drags a WorkStationObserver from the Canvas onto the WorkSpace, the WorkStationObserver is cloned, a new WorkStation is constructed and the cloned WorkStationObserver is registered as an observer of the WorkStation.

Once the objects are located on the WorkSpace, they are notified about their neighbour objects. If there are SimulationComponents located next to a newly added SimulationComponent, they should be connected if possible. Because it is difficult to locate neighbour objects in a continuous plane, especially because they might all have different sizes, we made the space available on the WorkSpace discrete. Making the available space discrete has been done by turning the available space into a discrete grid. A position on the grid has fixed dimensions and each object located on the grid covers one or more cells of the grid. The WorkSpace then keeps track of the positions covered by the objects located on it, and also of the inverse: which object is located on which position. This makes it possible for any object located on the WorkSpace to get its neighbour objects. The user can choose whether the grid is drawn on the WorkSpace.

Now we can locate the neighbour objects, it becomes possible to connect items. To connect items, we make use of a Visitor (Gamma et al., 1995, pp. 331-344) pattern. Whee1e1-3(m)7.8( )-5.6(grpkeeps tram)8.c1toy

observers are ordered to connect the underlying objects. Of course this is no guarantee that the SimulationComponents will get connected. It is for example possible that the Observers don't have the right orientation or that the maximal number of connections for a SimulationComponent has been reached.

Apart from the Canvas and the Workspace, a third component that is available in EtherSim, the ConsoleOutput. This item only gives some informative feedback to the user of what's happening. If a device needs to Back-Off for example because a collision occurred during the transmission of a frame, the number of slot times the device waits before retransmitting is shown in this ConsoleOutput.

All these observers inherit from the GUIObserver object, which inherits itself from a PictureBox. The PictureBox is a class that is already provided in the .NET framework. A PictureBox is an element meant to be used in combination with a Form. The PictureBox can hold images and shows them to the user. This feature exactly fulfils our needs: all components are represented by an image. Some images are stored in files, while others are created at run time. Some of these images loaded from a file are also modified at run time, this gives us the possibility to show for example the name of a Workstation on the image representing it. So the PictureBox has some nice and useful features, but it has some drawbacks also. As all elements of a form in .NET inherit from the class Control, the PictureBox also does. The major drawback is that it is impossible to do a binary serialization of a Control. To be able to cope with this problem, we had to define a custom serialization in the GUIObserver in order to be able to serialize the GUIObservers. This Serialization was needed to save the current network configuration.

Chapter 4
Microsoft .NET

EtherSim has been completely programmed in C#, which is a part of Microsoft's .NET platform. C# is an object oriented programming language, like Java also is. Although the C in C# might make users think that C# relates closely to the programming language C or C++, but the truth is that is distance to Java is much smaller than its distance to C or C++. In business environments for instance, most companies have to make the choice whether they develop their new applications on Sun's Java platform, or on the Microsoft .NET platform. This makes the .NET platform, with C# included a real included, a real concurrent for Java, but not for C or C++. As C# is an immediate rival for Java, we will mainly focus on the differences between Java and C#.

4.1 Java versus C#

As almost the same objectives can be accomplished with both programming languages, and corporations aren't willing to support two products with nearly the same functionality, we can state that C# is an immediate rival of Java. The reason for this is that they need developers who know both programming languages, or they need two types of programmers, one for each programming language. Apart from development, also deployment and support becomes more costly if more than one platform needs to be supported. For this reason, it is in business common to choose one platform to build all in-house applications. In this section, we will have a closer look to the pros and contras of each platform.

> *"When we look at the languages themselves, they're extremely close. But the environments they run in are very different, and the knowledge required to interface to the APIs is very different. "* [1]

This illustrates that the differences on the level of the programming language are rather limited. A Java developer will immediately understand C# code, and vice versa, but as the largest difference is located in the API's, it will take some time before a developer who changes from one language to the other will be as experienced in the other language. Tom Barnaby, lead instructor from Intertec Inc. states the following concerning this topic:

[1] Kevin Jubera, manager of application development services (ADS) centers of excellence in the IT department of Ford Motor Co.

"The amount of time to learn the other language is one to two weeks. But learning the underlying platform is a much harder task and can take a couple months."

Programs are in both cases compiled to some kind of intermediate code, and evaluated in a later stage. Java programs are compiled to some byte-code and are next run in the Java Virtual Machine. C# is –just as Visual Basic .NET- compiled the Microsoft Intermediate Language (MSIL). The result is evaluated in Microsoft's Common Language Runtime (CLR), although it is also possible to evaluate the MSIL using third party runtime environment. This makes it possible to run .NET applications on other platforms than Microsoft Windows. Examples of such third party alternatives are DotGNU[2] and the Mono project[3]. The evaluation of the intermediate code is called in both cases Just In Time (JIT) compilation.

As applications run in a virtual machine, the virtual machine is in both cases responsible for the memory management. In both cases, the keyword new is used to instantiate new objects. The virtual machine is responsible for garbage collection. The algorithm used by the .NET framework is the Mark and Compact Garbage Collection Algorithm.

4.1.1 Important Differences

A number of differences between Java and C# exist, although the languages seem at first view rather similar. One of these differences concerns nested classes. Nested classes exist both in C# and Java, but their meaning is quite different. A nested class is a class defined within another class. Java makes a distinction between a Static Nested Class, which is a class declared within another class and which has access to the static methods and variables of the enclosing class. Another type of nested classes in Java is the Inner Nested Classes, which are classes that can be declared within any code-block (including methods). For each enclosing class, there exist than a corresponding inner class. This inner class can't have any static methods or variables, but has access to all methods and variables of the enclosing class. C# also supports nested classes, but these always correspond to Java's Static Nested Classes.

Another important difference is located in the domain of multi-threading. In Java, a thread is created by defining a class that inherits from the special java.lang.Thread class, and next calling the run method to launch a new thread. In C# on the other hand, a System.Threading.Thread object should be created. This Thread object gets a System.Threading.ThreadStart object as parameter of the constructor. This

[2] DotGNU is available under the GNU public license at http://www.gnu.org/projects/dotgnu/
[3] Mono provides an open source alternative for the .NET framework. It is available at http://www.go-mono.com/

ThreadStart object contains a delegate (pointer to a method) to indicate where the thread should start running. It is important to note that in C# any method can be used as a starting point for a new thread, while in Java a special class must be defined and the thread starts always with the run method.

```
using System;

namespace PolyMorfism
{
         class Parent
        {
                public Parent()
                {
                }

                public virtual void callOverride() //Virtual Method, can be overridden
                {
                        System.Console.Out.WriteLine("Parent Override");
                }

                public void callNotOverride() //Non-virtual method, cannot be overridden
                {
                        System.Console.Out.WriteLine("Parent Not Overridden");
                }
        }
        class Child: Parent {

                public Child()
                {
                }

                public override void callOverride() //Override the base method
                {
                        System.Console.Out.WriteLine("Child Override");
                }

                public new void callNotOverride() //Do not override the base method
                {
                        System.Console.Out.WriteLine("Child Not Overridden");
                }
```

Fig 21a: Polymorphism in C#

```
            [STAThread]
            static void Main(string[] args)
            {
                    Child c = new Child();
                    Parent p = c;
                    p.callNotOverride();
                    p.callOverride();
                    c.callNotOverride();
                    c.callOverride();
            }
      }
}
```

Output :

>Parent Not Overridden

>Child Override

>Child Not Overridden

>Child Override

Fig 21b: Polymorphism in C#

In C# the switch statement allows string literals. Fall-through through different options is prohibited in C#, unless the label does not contain a statement. This in contrary to Java.

As said earlier, the major difference between Java and C# is located in the frameworks. One important part of the frameworks is the part responsible for collections. On this field Java exceeds .NET framework with its possibilities to access collections both in a thread-safe and a thread-unsafe manner. Java provides also some more types of collections and provides a larger set of algorithms for the manipulation of collections.

Just like Java, C# supports Serialization to write an object to a stream. While Java allows objects to be written to a binary stream, C# supports both a binary and an XML stream. By implementing the ISerializable interface in C#, it is possible to define a custom serialization. The same functionality is available in Java through the java.io.Externalizable interface. The serialization to XML is an important benefit, because the current trends are more and more to store data in XML format and data stored in XML format can be read with any other software. The serialization of objects to XML has also a great value when building distributed applications: it is possible to serialize objects to an XML stream conform to the Simple Objects Access Protocol (SOAP), specified in section 5 of the World Wide Web

Consortium (World Wide Web Consortium, 2003). Nevertheless those benefits, the save function in EtherSim uses the binary Serialization for saving.

4.2 The C in C#

The C in C# isn't there because there is no large difference between C or C++ and C#, but, according to Microsoft, because C# is a language that makes it easier for C programmers to build COM+ applications. To achieve this objective, some features from C and C++ are integrated into C# to make the C# language more feel like C or C++. Maybe the resemblance between C# and the other C-languages makes it easier to adapt C and C++ programmers to move to C#, but for the rest the C in C# can be called a marketing trick. The most important analogies between C# and C/C++ are the syntax for inheritance, the main returns an integer as status indicator, the syntax for namespaces and also the syntax of destructors is the same, although the semantics of the destructor are the same as these of "`finalize`" in Java. The semantics of the keyword `protected` on the other hand are the same as in C++. C# supports operator overloading, but contrary to C/C++ it doesn't allow the operators `new, ( ), ||, &&, =` to be overloaded. Contrary to Java, but similar to C++, methods are not by default made virtual. The programmer should declare them explicitly. Applications written in C# are –just like Java- ran in a virtual machine, the Common Runtime Environment (CLR). This CLR contains a garbage collector, which is responsible for removing unreferenced object from memory. Although a garbage collector is present in the CLR, it is still possible to use explicit pointers. Nevertheless, code using C-style pointers must be marked `unsafe`, and can only be run in fully trusted environments. The reason for this is that is possible to break type safety and even possible to access private members of a class. This incorporates possible security flaws, and for this reason it is not possible to execute unsafe code in an unsafe environment, such as over the Internet for example. Because unsafe code also removes bound checking on arrays, unsafe code is more performing when working with arrays.

C# also supports pointers to functions, in the form of delegates. Delegates are used for instance for associating an action with a particular event. They can also be used to pass operations as a parameter to generic algorithms, just like in C/C++.

C# also provides the goto statement. This allows jumping from some point in the code to some label. In C# it isn't possible to jump into a statement block.

In Java all methods are by default virtual. If one wants a method not to be overridden, it must explicitly be marked as final, which disallows a child class to have a method with the same signature. In C#,

methods must be marked explicitly as `virtual` if needed. When a base class contains a method with the same signature as a child class, the method should be marked with the keyword `override` or with the keyword `new`. If it is marked with the keyword `override`, the method should be marked virtual in the base class, and the method of the child class replaces then the method defined in the base class with its own. If a method should not be overridden in C#, it suffices to not mark it as virtual. An illustration is given in **Fig 21**.

Another important analogy between C# and C is that both languages support both multidimensional arrays in the form of jagged arrays, or also called "arrays of arrays" and multidimensional or rectangular arrays. The jagged array has a number of benefits over the rectangular array: the arrays in some dimension are not necessarily of equal length and when talking in 2 dimensions for example, the rows are accessible as objects, namely arrays. The benefits of a rectangular array are that the access time to objects is lower.

4.3 Memory Management in .NET

In the Microsoft .NET CLR, all resources are located on the managed heap. Objects that aren't useful any longer are removed from the managed heap automatically. When a process starts, a contiguous region of space is reserved for it. The heap also contains a pointer, called NextObjPtr, which indicates where the next object should be located. The use of the new operator places the newly created object on the heap –if it fits- and returns the address of the newly created object. So adding an object to heap is just adding a value to the NextObjPtr and incrementing the pointer with the size of the new object. To be able to work this way, it is assumed that the amount of memory is infinite, which isn't of course the case. In order to cope with this constraint, a mechanism to allow the managed heap to make such an assumption is created. This mechanism is called the garbage collector.

One assumption made by design of the garbage collector is that it is more likely for young objects to be destroyed quicker than older objects. So objects are classified into generations, where a generation reflects the age of the object. This makes it possible to perform garbage collection over one generation only, instead of doing garbage collection over all objects. (Liebermann and Hewitt, 1983)

Each application has some roots. Roots are pointers to some object on the managed heap or null pointers. Such roots are for instance global and static variables, but also CPU registers containing pointers to objects in the managed heap. Richter describes the working of the garbage collector in .NET in two articles (2000) (2000b).

4.3.1 Garbage Collection

When the garbage collection starts, it assumes all objects are garbage. The garbage collector begins building a graph of all objects accessible from the roots. When the garbage collector encounters an object that is already in the graph, it can stop exploring that branch. There are two reasons for doing this: it is quite useless to do the same job twice, and it guarantees that the algorithm is finite, in worse case all objects will be included in the graph.

In a second step, the garbage collector walks down the heap linearly. If it encounters some contiguous blocks of free space, the object located after this garbage is moved tight behind the previous object in the heap. Of course all root pointers and pointers in other objects to this object must be modified to the new location. When the NextObjPtr is reached, it is made pointing right after the last object. After this operation, the garbage is collected, and the heap is defragmented. The mechanism described here, is just the Mark and Compact algorithm. More information on the Mark and Compact algorithm is available in Appendix A.

As the garbage collector moves objects, a special mechanism is needed to be able to handle objects to which an explicit pointer exists, because moving the object would invalidate the pointer. An object which explicitly pointed to remains pinned on the heap, and is marked as pinned. A drawback of the use of explicit pointers is that they may cause fragmentation of the heap.

4.3.2 Finalization

When an object that is considered as garbage is encountered during garbage collection, its `finalize` method is called. The problem with those `finalize` methods is that it is not known when they will be called, because it is unpredictable to know when the allocated memory will be full. Another drawback is that the more `finalize` methods to be executed, and the more resource demanding the finalize methods are, the longer the garbage collection process will take.

To be able to handle the finalization of objects, a pointer to those objects is placed in the Finalization Queue when the object is created. When objects are determined as garbage, the garbage collector scans the Finalization Queue. If the Finalization Queue contains a pointer to the object to be removed, the pointer is moved to the F-Reachable Queue and the object itself isn't removed from the heap yet. When pointers are added to the F-Reachable Queue, the thread dedicated to the handling the finalization of the objects is woken up. This thread then calls the `finalize` method of the objects to be removed. So the F-Reachable Queue can be seen as an additional root. When the thread dedicated to handling the `finalize`

methods has called the `finalize` method of some particular object, this object removed from the F-Reachable queue, and has become truly unreferenced by now. The next time the garbage collector runs, the object will be removed from the managed heap.

The fact that objects which have a `finalize` method live, then die when they become unreferenced, then live, when they are referenced from the F-Reachable Queue and finally die permanently when its `finalize` method has been called, is called resurrection. Another special form of resurrection is when the finalization method of some object places a pointer to the object in some root, for instance a global variable. If this happens, the object is again alive after its finalization, as it is reachable from some root. Care must be taken in such cases, as it might be possible that the parent object or some objects referenced by the parent object are already finalized and now resurrected. This might cause unpredictable results. C# provides a method, called `ReRegisterForFinalize`, which registers the object again for finalization, and thus adds a pointer to it in the Finalization Queue. If the `ReRegisterForFinalize` method is called multiple times, the `finalize` method of the object will be called multiple times.

4.3.3 Improvements to the Garbage Collector

As mentioned earlier, the Garbage Collector used in the .NET framework isn't a straightforward implementation of the Mark and Compact algorithm, but it is implemented as a generational or ephemeral garbage collector. A generational garbage collector makes the following assumptions (Richter, 2000):

- Newer objects are more likely to have a shorter lifetime.
- Older objects tend to have a longer lifetime.
- New objects are strongly related to each other and are accessed frequently around the same time.
- It is faster to compact a portion of the heap than the complete heap.

Lifetimes of objects can vary widely. Some objects, for example the form of the application, keep almost from beginning to termination of the program in memory. Other objects are just used to store temporal results. Traditional (non-generational) garbage collectors threat both types of objects in the same manner. So it is as costly to reclaim free space from inaccessible new objects as from inaccessible old objects.

In the original paper from Lieberman and Hewitt (1983) the amount of available memory is divided into relatively small regions. New objects are created in one region, the current region. If this region is full, a newly allocated region becomes the current region. These regions are organized into generations,

which are numbered to be able to keep track of the age of the regions. When the garbage collection process is instantiated, the region is condemned. When a region is condemned, all live objects in it are moved to a new region with the same generation number. The performance improvement now comes from the fact that the rate at which garbage collection happens changes with the age of the objects.

In the garbage collector used in the .NET framework, a slightly different approach is taken. Assume the heap is empty. Objects are created and added to the heap. When the garbage collection is performed, the remaining objects are considered as objects of generation 1. Objects are created again, and the garbage collector runs again. The objects from generation 1 that survive become now generation 2. The newly created objects of generation 0 that survive become generation 1. This process continues until the maximal number of regions has been reached. When the maximal number of generations has been reached, the objects in the oldest generation remain there. Newer objects that survive a collection move up one generation.

The gain in performance can be found in the fact that it is possible to only collect the most recent generation, generation 0. References to objects of older generations must not be further explored. Of course it is possible that some old object has changed and references a new object. But the time of write in the new object can easily be known, and the time of the last collection also. So if a write recently happened to an old object, the references of that object are checked nevertheless.

Another improvement exists for handling large objects. Large objects are located on a separate heap, which is also garbage collected. The problem with compaction of large objects is that moving large objects in the heap wastes too much CPU time. For this reason, the Large Objects Heap is never compacted.

4.4 Memory Management in Java

In Sun's Java, the Java virtual machine is just as Microsoft's .NET framework responsible for the memory management. Just like in .NET, newly created objects are placed on a heap which is virtually infinitely large. To free memory and remove unused objects from the heap, the Java Virtual Machine (JVM) uses a garbage collector. As there is no official specification for the garbage collector in the JVM, we can't describe the official garbage collector in this document. Because the garbage collector runs its own thread, garbage collection happens transparently for the user or the programmer.

In Java, all objects are located on the heap, while local variables reside on the Java stack. Every thread has its own stack containing its local variables. These local variables are either a reference to objects or a

primitive type. The set of roots in a JVM is implementation dependent, but should always include the local variables that are references to objects. Live objects are then the objects referenced by a local variable or the objects referenced by another live object. It is also implementation dependent if the garbage collector distinguishes genuine object references from primitive objects. Garbage collectors that don't make the difference are called Conservative Garbage Collectors. The drawback of Conservative Garbage Collectors is that it is possible that they consider variable containing a primitive value as a valid reference to an object. In that case some unreferenced objects can be considered as live, and will thus not be collected. This trade-off is made for a lower CPU consumption.

Java supports, just as C#, finalization. The finalization is performed by the garbage collector, when the garbage collector collects the object. So care must be taken when using finalization-methods in Java.

4.5 Benchmarking

As mentioned earlier, both Sun Microsystems and Microsoft profile their languages as immediate rivals. In this light, it might be useful to make a compare both languages also on the computational performance level. As C# is a semi-interpreted language, just like Java, we already know that none of these will break the supremacy of C/C++ concerning computational power. Also C, nor C++ are immediate rivals of C# or Java. For this reason, we have only included C# and Java in the benchmarks. As Visual Basic .NET compiles to the same Microsoft Intermediate Language, there won't be any difference in performance between C# and Visual Basic .NET. For this reason, Visual Basic .NET can also be ruled out of the benchmarks.

Operation	Max	Min	Average	Median	Std Dev	95% Confidence
Integers	19013	15101	16227.6	15113.5	1800.537	1176.328962
Doubles	33589	27048	28388.2	27073	2735.032	1786.854861
Longs	47246	38044	39898.1	38060.5	3869.579	2528.078568
Trigonometric	8127	6511	6839.7	6520	671.9276	438.9846214
IO	11221	6789	7882.6	7331.5	1409.35	920.758661
Total	119171	93631	99236.2	94123.5	10073.65	6581.331286

Table 3: C# Results for the Arithmetic Benchmark (in milliseconds)

Operation	Max	Min	Average	Median	Std Dev	95% Confidence
Integers	21307	21217	21238.3	21232.5	26.76254	17.48453557
Doubles	143298	133319	134431.4	133483.5	3116.77	2036.252448
Longs	52584	40527	41749.7	40547	3806.818	2487.075645
Trigonometric	112521	95129	96954.1	95214	5470.174	3573.781028
IO	12027	8771	10230.3	10149	1049.211	685.4720831
Total	341658	299122	304603.8	300486	13052.83	8527.69235

Table 4: Java Results for the Arithmetic Benchmark (in milliseconds)

4.5.1 Arithmetic Benchmark

A first benchmark we ran was the benchmark developed by C. W. Cowell-Shah. This benchmark tests mathematical operations on integers, doubles and longs. It also benchmarks IO operations. The mathematical test performs the standard +, -, *, / operators alternatively on increasing numbers. For integers, for example, the sequence begins as follows: 1-2+3*4/5-6+7*8/9…. For integers, the first number is one, and as the calculation continues, 1 billion operations are performed. The mathematical test for doubles does the same, but it starts at 10 billion and stops also after 1 billion operations. The test for longs is identical to the test for doubles, but the calculation happens with doubles instead of with longs. Also other mathematical operations are performed, for instance the cosine, tangent, sine, logarithm and square root are calculated for 10 Million subsequent numbers (the numbers from 1 to 10 Million). To benchmark the IO operations, 1 million lines containing the number of the current line are written to a file and next read from that file.

Our test system is a computer with an Intel Pentium 4 processor running at 1300MHz with 512MB RD-Ram clocked at 400MHz. The operating system was Windows Server 2003, without any other program

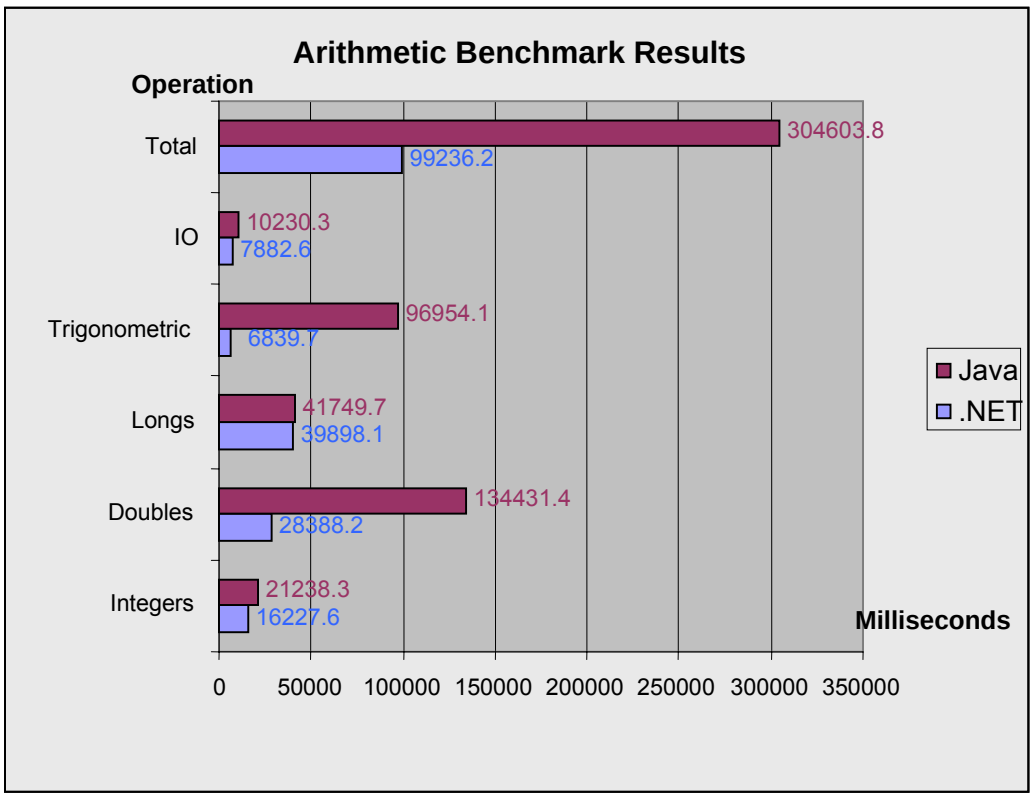

Fig 22: Average results of the Arithmetic Benchmark

running. Both the Java and .NET benchmark have been executed ten times each. Detailed results for those benchmarks are available in the Appendix B. We have modified the original code slightly to output the results into a comma separated file, in order to be able to process the results of the benchmarks afterwards. These modifications don't interfere nor change anything to the benchmark algorithm itself.

The minimal, maximal and average value together with the median for each sequence of experiments is given in the tables below. The complete set of results is available in Appendix B. The values give the time

Doubles	*.NET*	*Java*
Mean	27905.625	133462.1
Variance	5230874.839	8332.411
Observations	8	8
Pearson Correlation	0.60852979	
Hypothesized Mean Difference	0	
df	7	
t Stat	-133.7187179	
P(T<=t) one-tail	1.72533E-13	
t Critical one-tail	1.894578604	

Overall	*.NET*	*Java*
Mean	97429.625	300657.3
Variance	65677941.7	817351.1
Observations	8	8
Pearson Correlation	0.711109545	
Hypothesized Mean Difference	0	
df	7	
t Stat	-76.7614476	
P(T<=t) one-tail	8.37797E-12	
t Critical one-tail	1.894578604	

Integers	*.NET*	*Java*
Mean	16020.25	21232.38
Variance	2829393.643	141.6964
Observations	8	8
Pearson Correlation	0.238457785	
Hypothesized Mean Difference	0	
df	7	
t Stat	-8.77882376	
P(T<=t) one-tail	2.50681E-05	
t Critical one-tail	1.894578604	

Trigonometric	*.NET*	*Java*
Mean	6719.875	95236.38
Variance	311904.4107	5883.696
Observations	8	8
Pearson Correlation	-0.146702015	
Hypothesized Mean Difference	0	
df	7	
t Stat	-435.5894736	
P(T<=t) one-tail	4.43766E-17	
t Critical one-tail	1.894578604	

Longs	*.NET*	*Java*
Mean	39211.375	40548.25
Variance	10508675.98	269.6429
Observations	8	8
Pearson Correlation	-0.029557569	
Hypothesized Mean Difference	0	
df	7	
t Stat	-1.166248801	
P(T<=t) one-tail	0.140853163	
t Critical one-tail	1.894578604	

IO	*.NET*	*Java*
Mean	7602	10188.13
Variance	700807.1429	647954.4
Observations	8	8
Pearson Correlation	0.470199781	
Hypothesized Mean Difference	0	
df	7	
t Stat	-8.650134868	
P(T<=t) one-tail	2.7583E-05	
t Critical one-tail	1.894578604	

Table 5: Paired t-test results for the Arithmetic Benchmark (in Milliseconds)

needed in milliseconds for some particular sequence of operations.

Now we have the benchmark results, shown in **Table 3** and **Table 4**, it is desirable to test whether the difference in the results is significant. The results of the t-tests for each component of the benchmark are given in **Table 5**. As we'd like to know if the observed difference between the two sets of results is significant, we perform a two-tail paired t-test. The significance level is 0.05 and we excluded the minimal and maximal value for each component from the t-test.

From the results of the two-tailed paired t-tests for each component, we can conclude that the difference in the results is significant. The only exception is the arithmetic operations on longs. For longs, there is no significant difference between the results for each language. This means that for longs, Java and C# performed as well in our test environment while C# performed better than Java on the other operations.

4.5.2 Scimark 2.0 Benchmark

The results depend on a large set of parameters, for instance the implementation of the benchmark, and the environment in which the tests are run. To have a broader view on the performance we also tested run another benchmark, the SciMark 2.0[4] benchmark (Pozo and Miller, 2004). This benchmark is distributed

	Min	Max	Median	Average	Std Deviation	95% Confidence
Overall	148.6047	151.2814	150.5105	150.2033	0.804002034	0.406873691
FFT	100.4472	101.3816	100.9119	100.9433	0.214074965	0.112137264
SOR	183.8182	185.2332	185.2321	184.9014	0.453263209	0.24639216
Monte Carlo	18.89857	19.00572	18.95199	18.94493	0.039776698	0.022505372
Martix	207.9933	219.1232	216.2301	215.9348	2.922802761	1.727234315
LU	220.1539	232.1355	231.3481	230.2921	2.966150413	1.838405291

Table 6: The .NET results of the SciMark 2.0 Benchmark in the small variant (MFLOPS)

	Min	Max	Median	Average	Std Deviation	95% Confidence
Overall	99.46883	100.0734	99.74302	99.77677	0.152481011	0.082888099
FFT	40.57128	40.9677	40.73806	40.75972	0.122161319	0.066406429
SOR	181.3093	182.4768	182.063	181.8484	0.380888239	0.207049401
Monte Carlo	15.21743	15.32166	15.21743	15.25164	0.038936221	0.021165582
Martix Multiplication	68.38064	68.81142	68.40919	68.53352	0.157740442	0.085747105
LU	191.3304	193.4996	192.4089	192.4905	0.655314525	0.356226488

Table 7: The Java results of the SciMark 2.0 Benchmark in the small variant (MFLOPS)

[4] Available for free on http://math.nist.gov/scimark2/index.html. Developed by R. Pozo and B. Miller

Overall	*Java*	*.NET*
Mean	99,77677	150,2033
Variance	0,02325	0,646419
Observations	15	15
Pearson Correlation	0,348938	
Hypothesized Mean Difference	0	
df	14	
t Stat	-255,538	
P(T<=t) two-tail	4,36E-27	
t Critical two-tail	2,144787	

Monte Carlo Integration	*Java*	*.NET*
Mean	15,25164	18,94493
Variance	0,001516	0,001582
Observations	15	15
Pearson Correlation	-0,00242	
Hypothesized Mean Difference	0	
df	14	
t Stat	-256,672	
P(T<=t) two-tail	4,09E-27	
t Critical two-tail	0,692417	

FFT	*Java*	*.NET*
Mean	40,75972	100,9433
Variance	0,014923	0,045828
Observations	15	15
Pearson Correlation	0,360191	
Hypothesized Mean Difference	0	
df	14	
t Stat	-1138,55	
P(T<=t) two-tail	3,59E-36	
t Critical two-tail	0,692417	

LU	*Java*	*.NET*
Mean	192,4905	230,2921
Variance	0,429437	8,798048
Observations	15	15
Pearson Correlation	0,067988	
Hypothesized Mean Difference	0	
df	14	
t Stat	-48,9017	
P(T<=t) two-tail	4,75E-17	
t Critical two-tail	0,692417	

Sparse Matrix Mult	*Java*	*.NET*
Mean	68,53352	215,9348
Variance	0,024882	8,542776
Observations	15	15
Pearson Correlation	-0,10637	
Hypothesized Mean Difference	0	
df	14	
t Stat	-193,93	
P(T<=t) two-tail	2,07E-25	
t Critical two-tail	0,692417	

SOR	*Java*	*.NET*
Mean	181,8484	184,9014
Variance	0,145076	0,205448
Observations	15	15
Pearson Correlation	0,183438	
Hypothesized Mean Difference	0	
df	14	
t Stat	-22,0642	
P(T<=t) two-tail	2,83E-12	
t Critical two-tail	0,692417	

Table 8: Paired t-test results for the SciMark 2.0 benchmark between Java and C# (in MFLOPS)

by the US National Institute for Standards and Technology. It was originally written for Java only, but it has been ported to C#, C and J# [5] by W. Vogels and C. Re (no date) at Cornell University.

This SciMark 2.0 benchmark returns a result in Million of Floating Point Operations per Second (MFLOPS). To calculate the performance, it does some floating point operations. The mathematical methods that are used for those calculations are Fast Fourier Transformations, Jacobi Over-relaxation, Monte Carlo Integration, Sparse Matrix Multiplication and Dense LU Matrix Factorization. This SciMark 2.0 benchmark performs clearly less trivial calculations then the arithmetic benchmark from C. W.

Cowell-Shah. This makes the benchmark more valuable, but also increases the complexity. As the complexity increases, it is more likely that the code will be less optimal. Nevertheless, we performed the benchmark, on the same system as the other benchmark was performed. The detailed results are available in the Appendix B.

Four tests ware ran for the SciMark benchmark, because the benchmark supports two modes. In One mode (normal) the problems fit into the low level caches, in the other mode (large) they don't. The second mode also tests the subsystem's memory performance, while the first mode doesn't.

To be able to draw conclusions from these results, we must check if the differences between the results

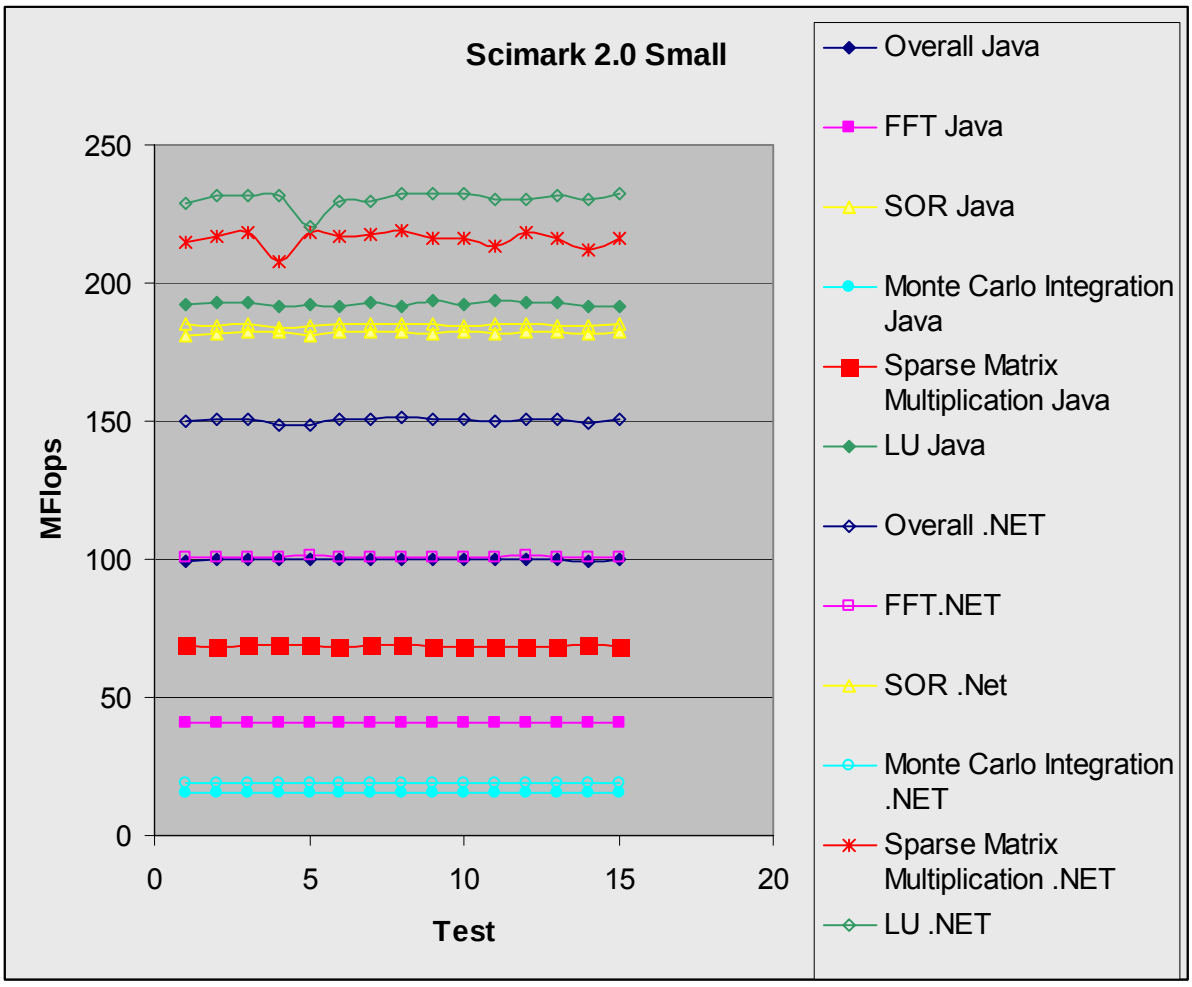

Fig 23: Results of the small variant of the Scimark 2.0 Benchmark

[5] Available from http://rotor.cs.cornell.edu/scimark/

are significant. We begin by checking if the overall results. As we want to check if one of the results is significantly better, we perform a paired t-test with the significance level set to 0.05. The results for the paired t-test are given in **Table 8**. As the two-tailed probability ($4.35539 * 10^{-27}$) is much smaller than 0.05, we can conclude there is a significant difference in the overall results for Java and .NET.

For each component, the two-tailed probability is lower than the significance level. This means that the differences for each individual component are significant. As the differences are significant for each component, in the small variant, and the average number of floating point operations per second is higher than the number of floating point operations in Java, it is clear that C# performs better for small problems in the domains tested by these calculations than Java.

Another variant of this SciMark 2.0 benchmark has also been run. The results are given in **Table 9** and **Table 10**. This variant uses more complex data, such that the used data structures don't fit any longer into caches. Detailed results are also available in Appendix B. In **Table 11** the results of the student-t tests for each component are given for this complex variant. From these results it can be seen that .NET performs significantly better on the tests included in the large variant of the SciMark 2.0 benchmark.

	Overall	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Max	110.4746	17.91035	185.0846	18.9519914	141.7886762	190.4222
Min	107.9222	17.44285	182.4394	18.8985685	137.2268381	182.5972
Average	109.7906	17.79388	184.6117	18.9164059	140.4980574	187.1332
Median	109.9167	17.82082	185.0846	18.8986497	140.6198694	187.2135
Std Deviation	1.109743	0.205535	1.263609	0.0251647	1.959390795	3.217625
95% Confidence	0.561597	0.104013	0.639463	0.01273486	0.991570334	1.628313

Table 9: Results of the Large Scirmark 2.0 Benchmark in C# (in MFLOPS)

	Overall	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Max	87.01158	15.66872	173.0169	22.5424468	71.95053223	152.8701
Min	85.12595	14.24543	172.1619	22.4669783	69.39550218	143.9574
Average	86.20782	15.08268	172.6799	22.5172906	71.54526372	149.214
Median	86.42853	15.15583	172.9388	22.5424468	71.72877545	150.4891
Std Deviation	0.788322	0.589309	0.386178	0.03557617	1.184655808	3.768226
95% Confidence	0.398939	0.298226	0.19543	0.01800369	0.59950754	1.906951

Table 10: Results of the Large Scimark 2.0 Benchmark in Java (in MFLOPS)

Monte Carlo Integration	*Java*	*.NET*
Mean	15.25164	18.94493
Variance	0.001516	0.001582
Observations	15	15
Pearson Correlation	-0.00242	
Hypothesized Mean Difference	0	
df	14	
t Stat	-256.672	
P(T<=t) two-tail	4.09E-27	
t Critical two-tail	0.692417	

SOR	*Java*	*.NET*
Mean	172.6799	184.6117
Variance	0.156833	0.542813
Observations	15	15
Pearson Correlation	-0.38185	
Hypothesized Mean Difference	0	
df	14	
t Stat	-48.1142	
P(T<=t) two-tail	5.96E-17	
t Critical two-tail	2.144787	

LU	*Java*	*.NET*
Mean	149.214	187.1332
Variance	7.812783	4.960486
Observations	15	15
Pearson Correlation	0.249528	
Hypothesized Mean Difference	0	
df	14	
t Stat	-47.2358	
P(T<=t) two-tail	7.70E-17	
t Critical two-tail	2.144787	

Overall	*Java*	*.NET*
Mean	86.20782	109.7906
Variance	0.348642	0.440082
Observations	15	15
Pearson Correlation	0.269947	
Hypothesized Mean Difference	0	
df	14	
t Stat	-120.216	
P(T<=t) two-tail	1.67E-22	
t Critical two-tail	2.144787	

FFT	*Java*	*.NET*
Mean	15.08268	17.79388
Variance	0.205625	0.014357
Observations	15	15
Pearson Correlation	-0.54727	
Hypothesized Mean Difference	0	
df	14	
t Stat	-19.8633	
P(T<=t) two-tail	1.18E-11	
t Critical two-tail	2.144787	

Sparse Matrix Mult	*Java*	*.NET*
Mean	71.54526	140.4981
Variance	0.366288	1.000374
Observations	15	15
Pearson Correlation	0.122897	
Hypothesized Mean Difference	0	
df	14	
t Stat	-241.989	
P(T<=t) two-tail	9.34E-27	
t Critical two-tail	2.144787	

Table 11: Paired Student-t tests for the complex variant of the SciMark 2.0 Benchmark (in

From these results, it seems that C# performs better than Java on the Fast Fourier Transformation, Jacobi Successive Over-Relaxation, Sparse Matrix Multiplication and on the Dense LU Matrix Factorization. On the other hand, Java performed significantly better on the Monte Carlo Numerical Integration. To be able to explain these differences, we'll have a closer look under the hood of each component. (Pozo and Miller, 2004)

The Fast Fourier Transformation does a one dimensional forward Fast Fourier transformation of complex numbers. This component tests array operations, complex arithmetic and trigonometric functions.

The SOR or Jacobi Successive Over-Relaxation component performs basic grid averaging. Each element in a matrix weighted with the average of its four neighbours. This component makes heavy use of array accesses and simple arithmetic. Because of the simple arithmetic operations, the numbers of Floating Point Operations for this component are significantly higher than those for other components.

The Monte Carlo Integration approximates the value of PI. This is done by calculating the integral of a quarter of a circle. The Monte Carlo Integration component tests especially on random number generation. This random number generator makes use of synchronized methods, and performs a number of rather simple arithmetic operations.

Spare Matrix Multiplication multiplies two matrices with about 0.5% of non-zero elements. Those non-

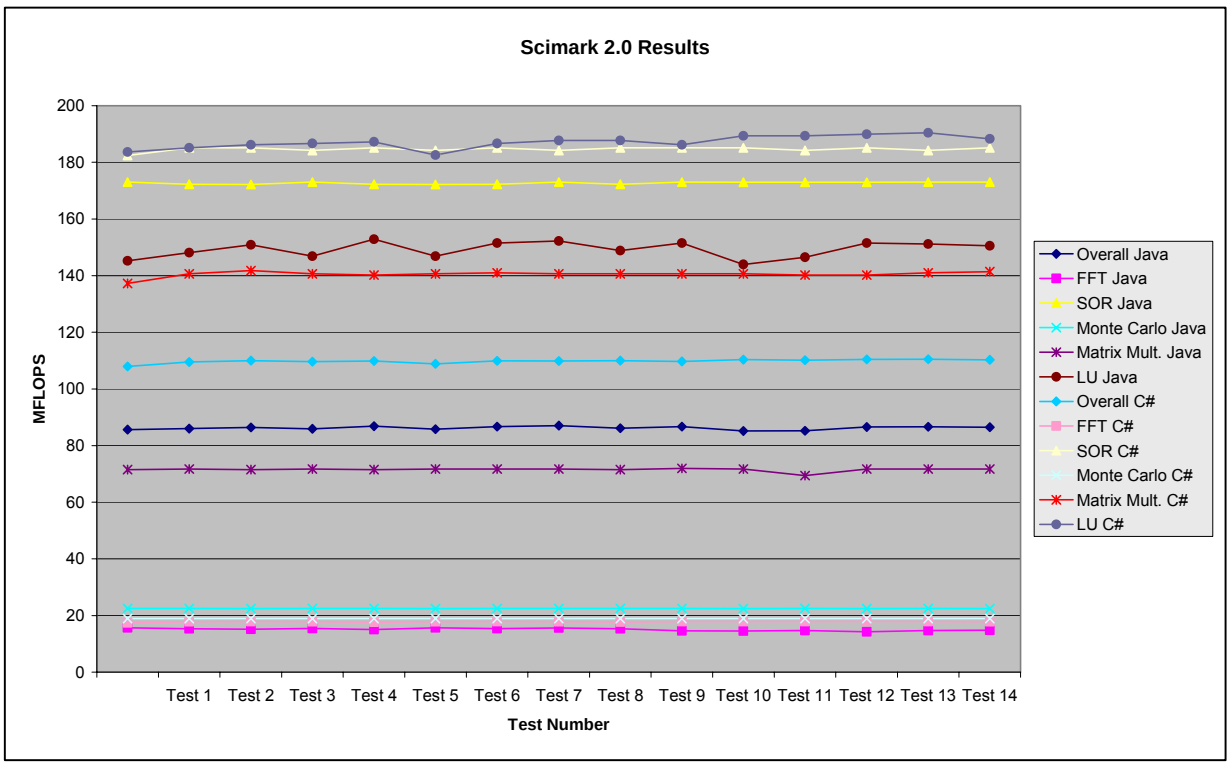

Fig 24 : SciMark 2.0 Results for the large variant

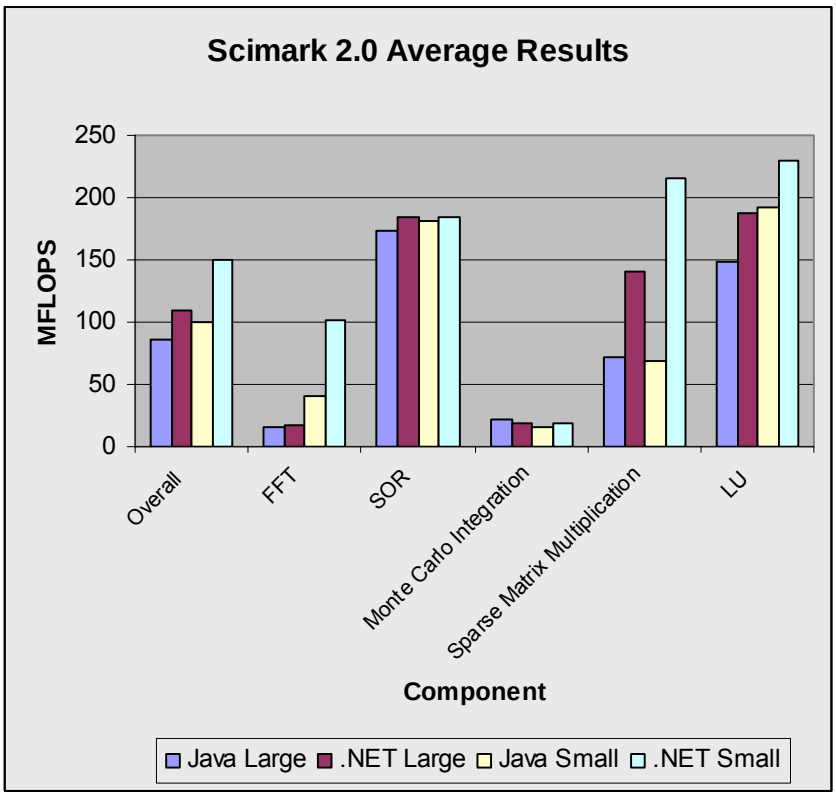

Fig 25: Average Results for the Scimark 2.0 Benchmark

zero elements are all located under the diagonal. This component heavily tests array access.

The last component, LU Matrix Factorization, computes the factorization of a dens matrix. To compute this, the program makes heavy use of dens matrix operations. As the dens matrices are represented by (multi dimensional) arrays, the most important aspect tested in this component is array operations.

A lot of these components make use of arrays, to represent vectors or matrices. Because C# performed better on all components that incorporated arrays, we could suspect C# to be more performing on array operations than Java. One might expect that reason for this is that in the C# implementation rectangular arrays, while Java uses jagged arrays, but as we checked the code, we found that both the C# and the Java version of the benchmark are implemented using jagged arrays. On the other hand, Java performed better on the Monte Carlo Integration component. This component makes use of synchronized methods, and we know that synchronized methods perform worse than other, non-synchronized, methods.

4.6 Conclusion

From the previous section, we can conclude that neither one of the languages is really better than the other. While Java provides a richer collection of containers, C# provides more possibilities concerning

interoperability between several languages. The current version of the .NET framework seems to perform significantly better concerning array operations than the current Java Virtual Machine, but as this factor is implementation dependent, it can't be really an issue while deciding which language to choose for a certain project. The next version of the Java Virtual Machine might overrule the benchmark results presented by the .NET framework today.

Concerning language complexity or syntax, there aren't much differences between Java and C#. Although C# incorporates some C-features, and can thus be considered as more powerful than Java, it is discouraged to use the most powerful features, such as explicit pointers. After all we can consider Java and C# as rivals of equal strength. The most important factors that will influence the decision are: the target operating system, the experience in one of the languages the programmers who should implement the project have and related to this the current in-house standards. The target operating system is a major issue, because the Sun's Java Virtual Machine supports nearly all operating systems, while, although third party solutions exist, such as Mono and DotGnu, Microsoft's .NET framework is only officially available for Microsoft Windows.

Chapter 5
Conclusion

Several objectives have been accomplished throughout this dissertation. The first objective was to build a program or a set of programs that could be used for explaining fundamental concepts of networking. The second objective was to give an analysis of C#, in particular in comparison to Java. To accomplish the first objective, first a study of the functioning of networking had to be done. In a next phase, we focused on some particular aspect of networking, namely the Medium Access Control (MAC). The program we built, EtherSim, illustrates the functioning of the MAC sublayer of the Data layer in the OSI Reference model for the most common network protocols: Classic and Fast Ethernet. It is important to be aware of the fact that the evolution of Ethernet is captured in EtherSim. It is for instance possible to simulate as well the classic 10Mbps (or even the original Xerox 2.94Mbps) Ethernet variant, as the more recent 1Gbps and even 10Gbps Ethernet variants. EtherSim illustrates the functioning of the main components available in the MAC sublayer for Ethernet: Computers, Half and Full Duplex links, Repeaters and Bridges.

There are numerous extensions possible. The most logical extension should be to allow simulation of other MAC protocols than Ethernet. A first MAC protocol that could be added is the Token Ring MAC protocol. A more interesting feature would be to add support for illustrating the MAC protocol used in Wireless LANs. It is clear that adding support for MAC Wireless Communication isn't that obvious. The most important reason that we go from guided transmission over some cable to unguided transmission: signals are transmitted as radio waves through the air. Modeling this requires a fundamentally different approach from modeling a system based on guided signal transmission.

Another important extension is implementation of more features of the MAC sublayer. EtherSim currently implements basic functionality of the MAC sublayer, but possibilities are left open to go more into the details. Interesting features to be added are for instance the Rapid Spanning Tree Protocol to allow quicker reconfiguration as a reaction on changes in topology. Another interesting feature that could be added to the Switch is support for Virtual LANs (IEEE 802.1Q).

As said before, EtherSim mainly focused on the MAC sublayer. Another nice extension could be to incorporate other levels of the OSI Reference model. This can be done in two directions: downwards we

could add the Physical Layer and model for instance attenuation of the signal over the medium, while in the upward direction we could model subnets and routing.

In the light of the analysis we made of C#, we situated C# in the world of programming languages. Although the name obviously refers to C/C++, we can state that the most important analogies between C# and C or C++ are located in the syntax of the language. Although the language C# incorporates some features of C/C#, conceptually the distance to Java is much closer. The most important difference between Java and C# are the APIs of the items available in the environments. It is important to note that there is no real best, Java has a somewhat richer collection of APIs while C# introduces some useful features from C. As there are no great technical differences, the choice between Java and C# is rather a choice based on business affairs, politics and personal preference.

In our benchmarks, C# performed significantly better than Java, but there could be numerous explanations for this. The implementation of the .NET runtime environment might for example simply perform better on Intel CPUs than Java. It is important to be aware of the fact that those results of the benchmarks highly depend on the implementation of the run-time environment. Apart from this the hardware has also its impact. So it isn't wise to decide to use a particular environment based on benchmarks: A new version of the virtual machine or new computer hardware can cause completely different results.

Acronyms and Definitions

4B5B: A translation scheme. A sequence of 4 bits is encoded as a nibble of 5 bits.

8B6T: An encoding scheme that encodes 8bits into a word of length 6, made up of different 3 tokens.

ACK: See *Acknowledgment Frame*

Acknowledgement Frame: A frame used in MACAW to indicate that the frame successfully reached receiver.

American National Standards Institute: A voluntary organization, founded in 1918, that creates standards for the computer industry. It has over 1300 members, including all the major computer companies.

ANSI: See *American National Standards Institute*

Blocked Port: A port on which a bridge doesn't receive or forward frames, as a result of running the STP protocol.

BPDU: See *Bridge Protocol Data Unit*

Bridge: A device connecting two LANs.

Bridge Protocol Data Unit: A frame used for communication between bridges.

Bytecode: The intermediate language to which Java code is compiled. This Bytecode is then interpreted in the JVM.

Carrier Sense Multiple Access: A technique where a device senses the medium to check if it is free to use before starting a transmission.

Clear to Send: A frame used in MACA and MACAW to indicate that a transmitter can start its transmission towards the device transmitting the Clear to Send frame.

CLR: See *Common Language Runtime.*

Collision: A situation where more than one device is trying to transmit data simultaneously over the same medium. The result is that all transmission are garbled.

COM: See *Component Object Model.*

COM+: An extension to the COM model.

Common Language Runtime: The part of the .NET framework that is responsible for the files containing MSIL.

Component Object Model: 'A software architecture that allows applications to be build from binary software components.' (Microsoft, 2002)

Computer Network: A collection of autonomous computers interconnected by a single technology.

Configuration Message: A message containing administrative information about bridges that propagates through a part of the network.

Congestion: Happens when the network gets (locally) overloaded.

Conservative Garbage Collector:

CRC: See *Cyclic Redundancy Check.*

CSMA: See *Carrier Sense Multiple Access.*

CSMA/CA: *CSMA* with collision avoidance.

CSMA/CD: *CSMA* with collision detection.

CTS: See *Clear to Send*

Cyclic Redundancy Check: A technique that calculates some checksum from the data. The result is transmitted or stored along with the data and can be used to check if the data got corrupted.

DCF: See *Distributed Coordination Function.*

Designated Port: The port of a bridge that is used to access the root in the active topology from the LAN connected to that port.

Distributed Coordination Function: A MAC mode specified in the IEEE 802.11 standard for Wireless LANs. This mode doesn't need a central control point.

Electronic Data Interchange: The transfer of data between different companies using networks. The by the ANSI approved standards for EDI are known under the name of the X12 standards.

EDI: See *Electronic Data Interchange*

Event: An occurrence that happens at some cut of the directed line modelling the flow of time.

Event-Triggered Architecture: Actions are carried out in response to the occurrence of some event, other than the occurrence of the regular event of a microtick.

Ethernet: A LAN architecture. Its specifications are used for the IEEE 802.3 standard.

Ephemeral Garbage Collector: See *Generational Garbage Collector.*

Extensible Markup Language: A system for organizing and tagging documents. A designer can also define its own tags.

FDM: See *Frequency Division Multiplexing*

File Transfer Protocol: A protocol for transferring files between electronic devices.

Frequency Division Multiplexing: A multiplexing technique where the available bandwidth is distributed among the several input streams.

FTP: See *File Transfer Protocol.*

Full Duplex: A situation where a device can be transmitting data and receiving other data simultaneously.

Generational Garbage Collector: A garbage collector that takes into account the age of the objects when collecting garbage. It can for example perform a garbage collect over a fraction of the objects currently in memory.

Granularity: The duration between two consecutive microticks.

Half Duplex: A mode where a device cannot be sending data and receiving some other data simultaneously.

Hyper Text Transfer Protocol: A protocol used to transfer formatted text between electronic devices.

HTTP: See *Hyper Text Transfer Protocol.*

International Standards Organization: ***"****A network of national standards institutes from 148 countries working in partnership with international organizations, governments, industry, business and consumer representatives. A bridge between public and private sectors."* (ISO, 2004)

ISO: See *International Standards Organization.*

Java Virtual Machine: Interprets bytecode.

JVM: See *Java Virtual Machine.*

LAN: See *Local Area Network.*

Local Area Network: A computer network that covers a relatively small area.

Manchester Encoding: An encoding scheme to convert bits into electromagnetic signals. This encoding scheme is used in Ethernet.

Medium Access Control: A mechanism that controls the access of multiple devices to a shared medium.

MAC: See *Medium Access Control.*

MACA: A MAC protocol with built in collision avoidance.

MACAW: A variant of MACA, optimized for wireless communication.

MFLOPS: Millions of Floating Point Operations per Second, used to measure the speed of a computer.

Microsoft Intermediate Language: The intermediate language to which C# and Visual Basic.NET code is compiled. This intermediate code is interpreted in the CLR.

Microtick: Periodic event generated by a clock.

MLT-3: See *Multi-Level-Transition 3.*

MSIL: See *Microsoft Intermediate Language*

Multi-Level-Transition 3: An encoding scheme to encode bits into electromagnetic signals using 3 levels of voltage. A transition in voltage or the absence of it encode a bit.

Non-Return-To-Zero: An encoding scheme to convert bits into electromagnetic signals. It uses a transition in amplitude or the absence of it to encode a bit.

NP-Hard: Non-Polynomially computable. This means that the required computation time and/or memory grow exponentially with the size of the problem to be solved.

NRZI: See *Non-Return-To-Zero.*

NRZI-3: An alternative name for *MLT-3.*

OSI: A model of 7 layers, it models a Network Architecture and each layer provides some services to the higher layers.

PCF: See *Point Coordination Function.*

Point Coordination Function: A medium access control mode specified in the IEEE 802.11 standard for Wireless LANs, based on a central coordination point which decides when which device can transmit.

Protocol: An agreement between devices when communicating.

Quality of Service: Requirements concerning jitter, throughput, loss of messages and delay.

QoS: See *Quality of Service.*

Reference Clock: "Assume an omniscient external observer who can observer all events that are of interest in a given Context. This observer possesses a unique reference clock z with frequency f^z which is in perfect agreement with the international standard of time." (Kopetz, 1997, pp. 45-65)

Repeater: A device used to regenerate and/or to replicate some signal on a LAN.

Request to Send: A frame used to indicate that a device wants to start a transmission. Used in MACA and MACAW.

Resurrection: A term to indicate that an object that was considered as destroyed is again accessible.

Root Port: Port of a bridge used to access the root bridge in the network with the least root path cost.

RTS: See *Request to Send.*

Simple Mail Transfer Protocol: A protocol for transferring email messages.

Simple Objects Access Protocol: A messaging protocol, based on XML. Request and reply messages are encoded to XML and can be transmitted over a wide variety of protocols.

SMTP: See *Simple Mail Transfer Protocol.*

SOAP: See *Simple Objects Access Protocol.*

Spanning Tree Protocol: A link management protocol used to build a spanning tree in a bridged network. This every LAN in the network appears exactly once as a node in the tree. The Spanning Tree Protocol is part of the IEEE 802.1D standard.

STP: See *Spanning Tree Protocol.*

RSTP: Rapid STP, a variant of STP that allows faster convergence.

TDM: See *Time Division Multiplexing*

Time Division Multiplexing: A multiplexing technique where each input stream gets periodically exclusive access to the medium.

Time-Triggered Architecture: All actions are carried out as a reaction to the occurrence of the regular event of a microtick.

Topology: The layout of a communication system.

Trigger: An Event that causes some action.

Type Safety: Programs cannot perform an operation on an object unless that operation is valid for that object. This prevents a program from doing inappropriate memory accesses.

Universal Twisted Pair: A cable containing pairs of cables where the wires of some pair are twisted around each other. Often used in LANs and the telephone system.

UTP: See *Universal Twisted Pair.*

XML: See *Extensible Markup Language.*

Appendix A

The Mark and Compact Algorithm

The Mark and Compact Algorithm consists of 2 phases. In first phase, it finds and marks all live objects. In the second phase, it compacts the heap by moving all live objects into contiguous memory locations.

The Mark and Compact Algorithm makes use of handles. The advantage of using handles is that it is

```
for each root variable r
    mark (r);



void mark (Object p)
    if (!handle[p].marked)
        handle[p].marked = true;
        for each Object q referenced by p
            mark (q);
```

Fig 26: The Mark Phase

possible to move an object in the heap without having to change all references to that object. It suffices to modify the handle. Another advantage, is that the algorithm can mark the handle instead of the object itself.

Once it is known which objects should be kept, and which should be removed, the heap should be defragmented, removing the unused spaces between the used data. The most straightforward method to do

```
void compact ()
    long offset = 0;
    for each Object p in the heap
        if (handle[p].marked)
            handle[p].object = heap.move (p, offset);
            handle[p].marked = false;
            offset += sizeof (p);
```

Fig 27: The Compact phase

this, is simply to move all data to one end, and mark the first location after the last object as the location to insert new objects.

This compact method moves all objects to one side of the heap. This movement process is done in one single pass through the live objects on the heap. It must also be noted that the relative positions of the objects on the heap remain unchanged. Another important feature is that the objects themselves don't need to be altered; modifications are performed on the handles only.

Appendix B
Benchmarks: Detailed Results

Christopher W. Cowell-Shah Benchmark

The results for the benchmark on mathematical and IO operations, developed by Cristopher W. Cowell-Shah, are given here. All values indicate the time in milliseconds to complete some sequence of operations. The benchmarks were run on a PC with an Intel Pentium 4 processor running at 1300MHz, with 512 MB RAM. The operating system was Windows Server 2003 with the .NET Framework 1.1 installed for the .NET benchmark and JSDK 1.4.2_02 for the Java benchmarks.

Test	Integers	Doubles	Longs	Trigonometric	IO	Total
Test 1	19013	33564	47246	8127	11221	119171
Test 2	18752	33589	47234	8102	9568	117245
Test 3	18739	27236	38067	6521	7032	97595
Test 4	15116	27068	38047	6531	7343	94105
Test 5	15106	27058	38047	6511	7032	93754
Test 6	15126	27065	38054	6519	7881	94645
Test 7	15101	27118	38104	6519	6789	93631
Test 8	15111	27058	38044	6519	7260	93992
Test 9	15111	27078	38044	6529	7320	94082
Test 10	15101	27048	38094	6519	7380	94142

Table 12: Detailed results for C# of the Arithmetic Benchmark (ms)

Operation	Max	Min	Average	Median
Integers	19013	15101	16227.6	15113.5
Doubles	33589	27048	28388.2	27073
Longs	47246	38044	39898.1	38060.5
Trigonometric	8127	6511	6839.7	6520
IO	11221	6789	7882.6	7331.5
Total	119171	93631	99236.2	94123.5

Table 13: C# Results for the Arithmetic Benchmark (ms)

Operation	Integers	Doubles	Longs	Trigonome	IO	Total
Test 1	21307	133598	40547	95209	11518	302179
Test 2	21217	133478	40527	95129	8771	299122
Test 3	21247	133319	40567	95219	10409	300761
Test 4	21227	133508	40547	95199	9520	300001
Test 5	21217	133489	40547	95209	10559	301021
Test 6	21248	133338	40537	95249	9021	299393
Test 7	21218	133518	40527	95169	9720	300152
Test 8	21237	133329	40537	95219	9889	300211
Test 9	21237	133439	40577	95418	10869	301540
Test 10	21228	143298	52584	112521	12027	341658

Table 14: Detailed results for Java for the Arithmetic Benchmark (ms)

Operation	Max	Min	Average	Median
Integers	21307	21217	21238	21233
Doubles	143298	133319	134431	133484
Longs	52584	40527	41750	40547
Trigonometric	112521	95129	96954	95214
IO	12027	8771	10230	10149
Total	341658	299122	304604	300486

Table 15: Java Results for the Arithmetic Benchmark (ms)

Scimark 2.0 detailed results

This section contains the detailed results of the SciMark 2.0 benchmark.

	Min	Max	Median	Average	Std Deviation	95% Confidence
Overall	99.46883	100.0734	99.74302	99.77677	0.152481011	0.082888099
FFT	40.57128	40.9677	40.73806	40.75972	0.122161319	0.066406429
SOR	181.3093	182.4768	182.063	181.8484	0.380888239	0.207049401
Monte Carlo	15.21743	15.32166	15.21743	15.25164	0.038936221	0.021165582
Martix Multiplication	68.38064	68.81142	68.40919	68.53352	0.157740442	0.085747105
LU	191.3304	193.4996	192.4089	192.4905	0.655314525	0.356226488

Table 16: The Java results of the SciMark 2.0 Benchmark in the small variant (MFLOPS)

	Min	Max	Median	Average	Std Deviation	95% Confidence
Overall	148.6047	151.2814	150.5105	150.2033	0.804002034	0.406873691
FFT	100.4472	101.3816	100.9119	100.9433	0.214074965	0.112137264
SOR	183.8182	185.2332	185.2321	184.9014	0.453263209	0.24639216
Monte Carlo	18.89857	19.00572	18.95199	18.94493	0.039776698	0.022505372
Martix	207.9933	219.1232	216.2301	215.9348	2.922802761	1.727234315
LU	220.1539	232.1355	231.3481	230.2921	2.966150413	1.838405291

Table 17: The .NET results of the SciMark 2.0 Benchmark in the small variant (MFLOPS)

Operations	Overall	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Test 1	99.64086	40.58639	181.3093	15.28675764	68.66722098	192.3547
Test 2	99.70865	40.58639	181.3775	15.21743	68.40918635	192.9527
Test 3	99.93851	40.72284	182.063	15.28675764	68.66722784	192.9527
Test 4	99.74302	40.89086	182.063	15.2797953	68.66722784	191.8142
Test 5	99.70176	40.89086	181.3093	15.28675598	68.66722784	192.3547
Test 6	99.65916	40.73806	182.063	15.21742836	68.40918635	191.8681
Test 7	100.0734	40.9677	182.4768	15.32165805	68.81142256	192.7892
Test 8	99.73054	40.89086	182.063	15.21743	68.66722098	191.8142
Test 9	99.87015	40.87553	181.3775	15.21742836	68.380637	193.4996
Test 10	99.78118	40.73806	182.063	15.28675598	68.40918635	192.4089
Test 11	99.96599	40.73806	181.9942	15.21742836	68.380637	193.4996
Test 12	99.85946	40.73806	182.063	15.21742836	68.380637	192.8982
Test 13	99.85088	40.57128	182.063	15.28675764	68.380637	192.9527
Test 14	99.46883	40.72284	181.3775	15.21743	68.69601705	191.3304
Test 15	99.65916	40.73806	182.063	15.21742836	68.40918635	191.8681

Table 18: Java Benchmark results of the small variant of the SciMark 2.0 Benchmark (MFLOPS)

Operations	Overall	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Test 1	149.6816	100.4472	185.2321	18.89856849	214.8117343	229.0186
Test 2	150.5252	100.9119	184.5219	18.89856849	216.9454802	231.3481
Test 3	150.9564	100.9112	185.2326	18.89860909	218.3916841	231.3481
Test 4	148.6047	100.9119	183.8182	18.95199143	207.9932636	231.3481
Test 5	148.6803	101.3808	184.5229	18.95199143	218.3916841	220.1539
Test 6	150.3663	100.9119	185.2321	18.95199143	216.9454802	229.7903
Test 7	150.5105	100.9119	185.2321	18.95199143	217.6667315	229.7897
Test 8	151.2814	100.9119	185.2321	19.00563513	219.1231587	232.1343
Test 9	150.6922	100.9115	185.2321	18.95199143	216.2300814	232.1355
Test 10	150.5607	100.9115	184.5224	19.00571726	216.2295371	232.1343
Test 11	149.8147	100.9112	185.2321	18.95199143	213.4124032	230.5659
Test 12	150.894	101.3816	185.2321	18.89860909	218.3916841	230.5659
Test 13	150.3929	100.9119	184.5229	18.95203226	216.2295371	231.3481
Test 14	149.3859	100.9119	184.5224	18.89860909	212.0306618	230.5659
Test 15	150.7029	100.9119	185.2332	19.00563513	216.2295371	232.1343

Table 19: Detailed results for .NET of the small variant of the SciMark 2.0 benchmark (MFLOPS)

	Overall	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Test 1	85.60394	15.66872	173.0169	22.54244677	71.54835443	145.2433
Test 2	85.99114	15.30986	172.2393	22.54244677	71.74888215	148.1152
Test 3	86.42853	15.15583	172.1619	22.46697833	71.52835995	150.8296
Test 4	85.91717	15.48722	173.0169	22.54244677	71.72877545	146.8105
Test 5	86.84566	15.02806	172.2393	22.54244677	71.54835443	152.8701
Test 6	85.7628	15.6458	172.1619	22.46697833	71.72877545	146.8105
Test 7	86.68631	15.42023	172.2393	22.54244677	71.74887735	151.4807
Test 8	87.01158	15.57744	173.0169	22.54244677	71.74887735	152.1723
Test 9	86.09428	15.33181	172.2393	22.54244677	71.54834966	148.8095
Test 10	86.72557	14.6373	173.0169	22.54244677	71.95053223	151.4807
Test 11	85.12595	14.53782	172.9388	22.46697833	71.72877545	143.9574
Test 12	85.20844	14.67747	172.9388	22.54244677	69.39550218	146.488
Test 13	86.57903	14.24543	172.9388	22.46697833	71.72877545	151.5152
Test 14	86.63268	14.71786	173.0169	22.54244677	71.74888215	151.1373
Test 15	86.50423	14.79932	173.0169	22.46697833	71.74888215	150.4891

Table 20: Detailed results for Java of the large variant of the SciMark 2.0 benchmark (MFLOPS)

	Total	FFT	SOR	Monte Carlo Integration	Sparse Matrix Multiplication	LU
Test 1	107.9222	17.44285	182.4394	18.8985685	137.2268381	183.6035
Test 2	109.5221	17.82082	185.0846	18.9519914	140.6198694	185.1331
Test 3	109.958	17.85053	185.0846	18.8985685	141.7886762	186.1677
Test 4	109.6389	17.79113	184.1955	18.8986497	140.6198694	186.6895
Test 5	109.815	17.64435	185.0839	18.8986497	140.2346353	187.2135
Test 6	108.8263	17.8208	184.1948	18.8986091	140.6198694	182.5972
Test 7	109.9167	17.85055	185.0846	18.9519914	141.0075222	186.6887
Test 8	109.8493	17.79115	184.1955	18.8985685	140.6201641	187.7412
Test 9	110.02	17.70279	185.0846	18.9519914	140.6198694	187.7408
Test 10	109.7302	17.8804	185.0846	18.8986497	140.6198694	186.1673
Test 11	110.3648	17.8804	185.0846	18.8986091	140.6198694	189.3408
Test 12	110.1206	17.88042	184.1948	18.9519914	140.2349283	189.3408
Test 13	110.4124	17.91035	185.0846	18.9519914	140.2346353	189.8806
Test 14	110.4746	17.85053	184.1942	18.8986497	141.0075222	190.4222
Test 15	110.2884	17.79115	185.0846	18.8986091	141.3967224	188.2711

Table 21: Detailed results for .NET of the large variant of the SciMark 2.0 benchmark (MFLOPS)

Bibliography

Ander, S. et al. (1999), *'Testing of Event-Triggered Real-Time Systems'*, online at http://www.artes.uu.se/project/9905/9905-11_his_990818.pdf [Accessed May 10, 2004]

Bauer, B., Patrick, A.S. (2004), "*A Human Factors Extension to the Seven-Layer OSI Reference Model*.", online at http://www.andrewpatrick.ca/OSI/10layer.html [Accessed April 29, 2004]

Budd, T. (2002), *"An Introduction to Object Oriented Programming"*, Third Edition, Addison Wesley, Boston, USA

Cisco Systems Inc. (2003), *"Understanding Spanning-Tree Topology Changes"*, online at http://www.cisco.com/warp/public/473/17.html [Accessed May 15, 2004]

Cowell-Shah, C. (2004), 'Nine Language Performance Round Up: Benchmarking Math and File IO', "*OSNews.com, Exploring the Future of Computing*", online at http://www.osnews.com/story.php?news_id=5602 [Accessed May 15, 2004]

Day, J.D., Zimmerman H. (1983), 'The OSI Reference Model', *Proc. Of IEEE*, vol. 71, pp. 1334-1340

Dix, A. et al. (1998), "*Human-Computer Interaction*", Second Edition, Prentice Hall Europe, Harlow, England

Flanagan, D. (2002), "*Java in a Nutshell, A Desktop Quick Reference*", Fourth Edition, O'Reilly, Sebastopol, USA

Fuchs, E. et al (1997), *"Time Triggered Architecture (TTA)"*, TU Vienna, Austria, online at http://www.vmars.tuwien.ac.at/projects/tta/papers/DBTU_emmsec97/DBTU_emmsec97.html [Accessed May 15, 2004]

Gamma, E. et al. (1995), "*Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison-Wesley, New York, USA

Grimaldi, R. (1999), "*Discrete and Combinatorial Mathematics*", Fourth Edition, Addison-Wesley Longman Inc., New York, USA

IEEE (1998), 'Part 3: Medium Access Control (MAC) Bridges', "*IEEE standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Common Specifications*", ANSI/IEEE standard 802.1D, Institute of Electrical and Electronics Engineers Inc., New York, USA

IEEE (1998), 'Virtual Bridged Local Area Networks', "*IEEE Standards for Local ad Metropolitan Area Networks*", IEEE standard 802.1Q, The Institute of Electrical and Electronics Engineers Inc., New York, USA

IEEE (2001), 'Part 3: Medium Access Control (MAC) Bridges – Amendment 2: Rapid Reconfiguration', "*IEEE standard for Local and Metropolitan Area Networks – Common Specifications*", IEEE standard 802.1W, Institute of Electrical and Electronics Engineers Inc., New York, USA

IEEE (2002), 'Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer', "*IEEE standard for Information Technology – Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks – Specific Requirements* ", IEEE standard 802.3, Institute of Electrical and Electronics Engineers Inc., New York, USA

ISO (2004), *"ISO – International Organization for Standardization"*, online at http://www.iso.org [Accessed May 18, 2004]

Jaber S., Obasanjo, D. (2002), *"C# versus Java"*, online at http://www.dotnetguru.org/articles/CSharpVsJava.htm [Accessed May 16, 2004]

Karn, P. (1990), *"MACA - A New Channel Access Method for Packet Radio",* Appeared in the proceedings of the 9th ARRL Computer Networking Conference, London, Canada and online at http://www.ka9q.net/papers/maca.html

Kopetz, H. (1997), *"Real-Time systems: Design Principles for Distributed Embedded Applications"*, Kluwer Academic Publishers, Vienna, Austria

Lieberman, H., Hewitt C., (1983), *"A Real Time Garbage Collector based on the Lifetime of objects"*, MIT Artificial Intelligence Laboratory, Massachusetts, USA

Lindholm, T, Yellin F (1999), 'The Structure of the Java Virtual Machine', in *"The Java Virtual Machine Specification"*, Second Edition, Addison-Wesley Pub Co, Palo Alto, USA

Mariani, R (2003), 'Garbage Collector and Performance Hints' in *MSDN*, online at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetgcbasics.asp [Accessed April 22, 2004]

Meier, R., Cahill, V. (2002), "*Taxonomy of Distributed Event-Based Programming Systems*", Dublin, Ireland

Microsoft (2002), 'Component Object Model – COM', "*Microsoft COM Technologies – Information and Resources for the Component Object Model based technologies*", online at http://www.microsoft.com/com/default.asp [Accessed May 18, 2004]

Norton, M. (2001), 'Understanding Spanning Tree Protocol – The fundamental bridging algorithm' in *O'Reilly Network* online at http://www.oreillynet.com/pub/a/network/2001/03/30/net_2nd_lang.html [Accessed April 18, 2004]

Pozo, R. Miller, B. (2004), *"Java Scimark 2.0"*, online at http://math.nist.gov/scimark2/ [Accessed April 24, 2004]

Preiss, B. (1999), *"Data Structures and Algorithms with Object-Oriented Design Patterns in Java"*, Waterloo, Canada, online at http://www.brpreiss.com/books/opus5/ [Accessed April 22, 2004]

Preiss, B (2001), *"Data Structures and Algorithms with Object-Oriented Design Patterns in C#"*, Waterloo, Canada, online at http://www.brpreiss.com/books/opus6/ [Accessed May 16, 2004]

Rapaport, M. (2004), "*EDI Architecture",* online at http://www.sonic.net/~quine/EDIArchitecture.html [Accessed May 11, 2004]

Re, C., Vogels, W. (2004), *"Scimark – C#"*, online at http://rotor.cs.cornell.edu/SciMark/ [Accessed April 27, 2004]

Richter, J. (2000), 'Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework', *MSDN Magazine*, November 2000, online at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetgcbasics.asp [Accessed April 22, 2004]

Richter, J. (2000), 'Garbage Collection – Part2: Automatic Memory Management in the Microsoft .NET Framework', *MSDN Magazine*, December 2000, online at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetgcbasics.asp [Accessed April 22, 2004]

Russel, S., Norvig, P. (1995), "*Artificial Intelligence, A Modern Approach*", Prentice Hall Inc., New Jersey, USA

Sweeney, T. (2003), 'Java versus C# -- no need for war, just understanding', *SearchVB.com, The Source for News and Tips on Visual Basic, .NET and Visual Studio*, online at http://searchvb.techtarget.com/vsnetTip/1,293823,sid8_gci880182_tax293034,00.html [Accessed April 20, 2004]

Tanenbaum, A. (2003), "*Computer Networks*", Fourth Edition, Prentice Hall, Amsterdam, The Netherlands

Vogt, C. (1999), *"Creating Long Documents using Microsoft Word"*, online at http://ist.uwaterloo.ca/ew/thesis/Thesis_word.html [Accessed May 18, 2004], Waterloo, Canada

WildPackets Inc. (no date), 'Specifications for Data Encoding', "*WildPackets Inc., Technical Compendium, Technical Engineering Networking*", online at http://www.wildpackets.com/compendium/FE/FE-Encod.html [Accessed May 15, 2004]

WildPackets Inc. (2004), 'Propagation Delay and its Relation to Maximum Cable Length' in *WildPackets – Technical Compendium, Technology Engineering Networking* online at http://www.wildpackets.com/compendium/EN/EN-Propa.html [Accessed April 20, 2004]

World Wide Web Consortium (2003), "*SOAP Specifications*", online at http://www.w3.org/TR/soap/ [Accessed May 18, 2004]