

MSc in Computer Science



Reconfiguration between the different congestion avoidance and control algorithms for 4G handsets

Student name: Zygouras Vasileios
Personal code: TL10NET002

Supervisor Professor: Dr Costas Chaikalis

Larissa 2012

Contents

Introduction	3
Chapter 1: Reconfigurability in 4G	6
1.1 4G.....	6
1.2 Reconfigurability.....	8
1.3 Mobivas platform.....	15
Chapter 2: Study and Analysis of the Reconfigurable Algorithms and Multiplexing Techniques	18
2.1 Media Access Control in Wireless Networks.....	18
2.1.1 Taxonomy of Wireless Networks.....	19
2.1.2 Taxonomy of Wireless Networks based on range.....	20
2.1.3 Channel Access Methods.....	21
2.2 Block Diagrams of the most common Reconfigurable Algorithms and Multiplexing Techniques.....	24
2.2.1 Time Division Multiple Access (TDMA) Block Diagram.....	24
2.2.2 Orthogonal Frequency Division Multiplexing (OFDM) Block Diagram.....	25
2.2.3 Code Division Multiple Access (CDMA) Block Diagram.....	28
2.2.4 Carrier Sense Multiple Access (CSMA) Block Diagram.....	30
2.2.5 Combined Algorithm Block Diagram.....	32
2.3 Reconfigurable Algorithms and Multiplexing Techniques.....	38
2.3.1 Time Division Multiple Access TDMA.....	38
2.3.2 Code Division Multiple Access CDMA.....	40
2.3.3 Orthogonal Frequency-Division Multiplexing OFDM.....	42
2.3.4 Carrier Sense Multiple Access CSMA.....	44
Chapter 3: Simulation	49
3.1 OMNEST Network Simulation Tool.....	49
3.2 Simulation of Time Division Multiple Access TDMA.....	51
3.3 Simulation of Code Division Multiple Access CDMA.....	58
3.4 Simulation of Orthogonal Frequency Division Multiplexing OFDM....	66
3.5 Simulation of Carrier Sense Multiple Access CSMA.....	75
3.6 Comparison of the Simulation Results.....	82
Conclusion	84
References	86
Appendix of the Code	93

Abstract

This is a project that involves the investigation of autonomic technology to enable reconfiguration of the Media Access Control (MAC) in future 4G networks. The main aim is to examine how contention and congestion methods work in modern wireless networks and create a reconfigurable standard for future 4G networks. In order this issue to be addressed, two types of techniques can be employed: contention-free or contention-based. Already experiencing 4th Generation in Wireless Communications, a comparative study of four major multiplexing algorithms that are the most primarily used has been conducted in order Reconfigurability in modern wireless networks to be achieved. These algorithms are the Time Division Multiple Access, TDMA, which uses division in Time, the Code Division Multiple Access, CDMA, which uses coding division and spectrum spreading, the Orthogonal Frequency Division Multiplexing, OFDM, where different frequency carriers are used and finally the Carrier Sense Multiple Access, CSMA, trying to sense the carrier before the message has been received. The Simulations that took place covered the implementation of different scenarios that correspond to the operation of each algorithm separately and gave us vital and important results. The parameters that were examined were the Throughput, the Performance over Latency and the Error Probability. TDMA has proven to be the fastest in terms of Performance. However CDMA and OFDM seem to manage better as far as Error Probability is concerned whereas CSMA and OFDM are proven to be more efficient in the Throughput case. The Simulations that took place were developed with the help of OMNEST++ Network Tool by Andras Varga.

Introduction

Global vision consensus on the next generation of wireless mobile communications, broadly termed 4G, sketches a heterogeneous infrastructure, comprising different wireless access systems in a complementary manner and vested with reconfiguration capabilities. These capabilities will facilitate a more flexible and dynamic adaptation of

the wireless network infrastructure. As a consequence the ever-changing service requirements will be met in a better and more effective way.

Fourth generation wireless 4G means different applications and services to different people. For some it is just higher capacity (e.g. 100 Mbps) while for others it is an interworking of cellular and WLAN technologies that employs a variant of mobile IPv6 mobility management (e.g. Hierarchical Mobile IPv6) for inter-system handoff and IETF AAA technologies for seamless roaming. However the most important issue as far as 4G is that 4G will be mainly characterised by a horizontal communication model, where multiple different access technologies, such as cellular, cordless, wireless LAN, short range connectivity and even wired systems will interface to a common platform over the IP protocol. Such an action will enable this variety of multiple access technologies and systems to be complemented in an optimal way for different service requirements and radio environments.

Over the last few years, a number of EU research projects in mobile communications (FIRST, SORT, TRUST, CAST, MOBIVAS, SCOUT) have addressed the thematic area of **reconfigurability**. The action of addressing the area of reconfigurability has not only to do with the identification of main functional requirements but also with the proposal of a number of reconfiguration-supporting architectures. All these functional requirements and architectures are each one separately tailored to fit the problem domain under study by each project.

More specifically, reconfigurability deals with the dynamic instantiation, parameterization and inter-connection of functional entities (e.g. protocols) within the user. Furthermore it deals with the control and management planes of a collection of operating communication systems in a manageable, consistency-preserving and transparent way.

In order for possible common (reconfigurable) operations to be found among the aforementioned algorithms the **Mobivas (Mobile Value-Added Services)** platform is

used. The Mobivas intelligent application platform [1] provides for the integration of basic entities, namely the contracted Value-Added Service Provider, the Operator and network infrastructure, the Access network and the end user. The overall objective of Mobivas (Mobile Value-Added Services) is to develop architectural approaches and prototypical implementations of integrated software platforms and systems, which enable the flexible provision of Value-Added Services (VAS) in mobile communication networks. These platforms are adaptable to different network services and technologies, such as CDMA, OFDM, TDMA and CSMA, using different processing blocks, and constitute new opportunities for third party Value-added Service Providers (VASPs). The objective of Mobivas is not only to define, design and develop but also to validate integrated application architecture for downloadable Software Defined Radio (SDR) Value-Added-Services (VAS) and for innovative and modular network components for the seamless and efficient service provision in converging existing network.

Several hardware choices or software ones are offered in order for reconfigurability to be achieved. Algorithms such as Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA), Orthogonal Frequency-Division Multiplexing (OFDM) and Carrier Sense Multiple Access (CSMA) are the most widely and effective multiplexing techniques that are used in 4G wireless communications.

These algorithms are developed and evaluated with the help of OMNEST Open Source Network Simulation Tool by Andras Varga.

Chapter 1: Reconfigurability in 4G

1.1 4G

Traditionally, wireless systems were considered as an auxiliary approach that was basically used in regions where the establishment of a wireline connection was difficult to be built. The 1st Generation wireless systems that deployed in the 1980s were mainly based on analog technique. The evolution of 1G successfully led to the solution of various fundamental and vital problems, such as cellular architecture adopting, non-interrupted communication and multiplexing frequency band in mobile circumstances. It has to be mentioned that speech was the only service that was offered by the **1G**.

2G was based on digital signal processing, which gained tremendous success during the 1990s. GSM is the most representative service in 2G. Its main contributions were basically the utilization of SIM (Subcarrier Identity Module) cards and the offering of capabilities for a large number of users.

2.5G added data service and packet switching methods in the existing ones, bringing as a consequence the Internet into mobile communications that were personally conducted.

The revolutions of **3G** were dominantly the introduction of higher data rates and broader bandwidths in mobile communications. A big variety of multimedia data communications services are transmitted by 3G networks using the digital signal processing method. However, some aspects of 3G are still need to be further improved.

High error rates, low capacity as well as Quality of service may be the undesirable results that arise with the continuous increase in bandwidth and data rate, together with the simultaneous existence of various services. Additionally, the spectrum that could be used was limited and it was not only even more difficult the allocation of frequency bands but also the roaming across distinct service environment. Furthermore, there was lack of end-to-end seamless support mechanism spanning a mobile subnetwork and a fixed-one [2].

A **4G** system will be able to give a comprehensive solution where voice, data and multimedia streaming can be successfully offered to users not only on an Anytime and Anywhere basis but it can also be given at higher data rates compared to those of previous generations. The main objective is that 4G is a fully IP-based integrated system, achieved by the convergence of wired and wireless technologies. Such a system will enable between 100 Mbps and 1 Gbps speed both indoors and outdoors, with high quality and high security at an affordable cost to be successfully provided [2]. The following Figure (Figure 1) shows in detail a 4G feature framework. As it can be easily observed, a 4G feature framework consists of the users, the terminals and the various applications that are offered to the users. The various diversities among the various parts that participate in the framework are overcome with the adaptability that is offered in a 4G framework.

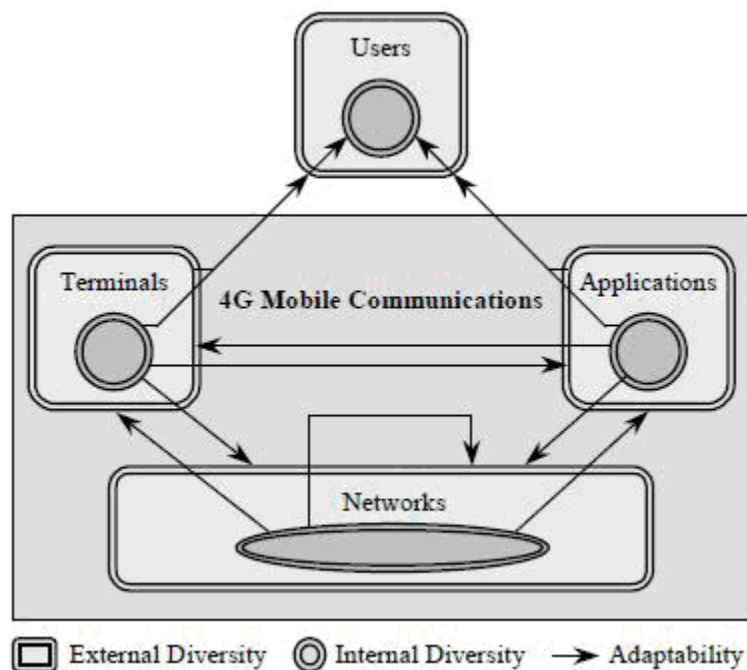


Figure 1: 4G feature framework

1.2 Reconfigurability

Optimization of different factors is required in wireless communication systems. These factors include real time performance, area, power, flexibility and time-to-market. The combination of the above factors cannot be optimized with instruction set and custom hardware processors. Furthermore, reconfigurable hardware offers a good balance between flexibility and implementation efficiency. Heterogeneous Systems-on-Chip including instruction set processors, reconfigurable blocks and custom hardware as well as multiplexing algorithms implementing software , as indicated in this work , are currently the state-of-the-art for the realization of future wireless communication systems. The critical design task in such platforms is the assignment of the target system's tasks on the different types of processing elements that are already available [3].

The designers of wireless communication systems make efforts in the field of extracting transmission capacity from a relatively limited amount of spectrum by placing increasing signal processing requirements. These computational requirements are outstripping the processing capabilities of conventional instruction set digital signal processors. The main and basic point is that wireless communication standards not only evolve and change in a fast way but also the various demands for new features, services and advanced functionality arise in a rapid way. Furthermore, competition in the field tends to be hard and it is both vital and substantial for a product to be available in the market before others appear and at a competitive price. Apart from the on-the-field upgradeability, there is also need for adaptation to more dynamic factors in the operational environment such as the traffic mix. The main issue is that the various post fabrication programmability and flexibility requirements cannot be covered by conventional ASICs [3].

A substantial increase in the fields of flexibility and implementation efficiency is required in wireless communication systems. Implementation platforms, new technologies and implementation hardware are used towards these successfully conveying messages from one end to another, on uplink and on downlink terms. Reconfigurability can be achieved through middleware architecture, adapted to our needs in every case. It is difficult to put

limits to wireless communication rates. Broadband communications rely on big data rates. The main goal is the achievement of high capacity and quality of service in the same channels and environments. Such an action enables the safe delivery of various messages at the same time, without errors and at proper power levels [3].

Several hardware or software choices are offered in order to achieve reconfigurability. The algorithms that are explained in the next step are the most widely and effective multiplexing techniques that are used in 4G wireless communications. In the following Figure (Figure 2) a 4th Generation, 4G, network is illustrated. As it can be easily observed the global roaming and the wide mobility range are dominant in 4G networks. There is a variety of distribution channels as well as multi and single point individual links. Furthermore, it has to be mentioned that the network access methods (algorithms), such as Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA), Orthogonal Frequency-Division Multiplexing (OFDM) and Carrier Sense Multiple Access (CSMA), which will be examined in the next section are widely used in 4G networks. Additionally, it has to be mentioned that voice, data and multimedia streaming are given to users on an anytime and anywhere basis and can also be provided at higher data rates than these compared to previous generations.

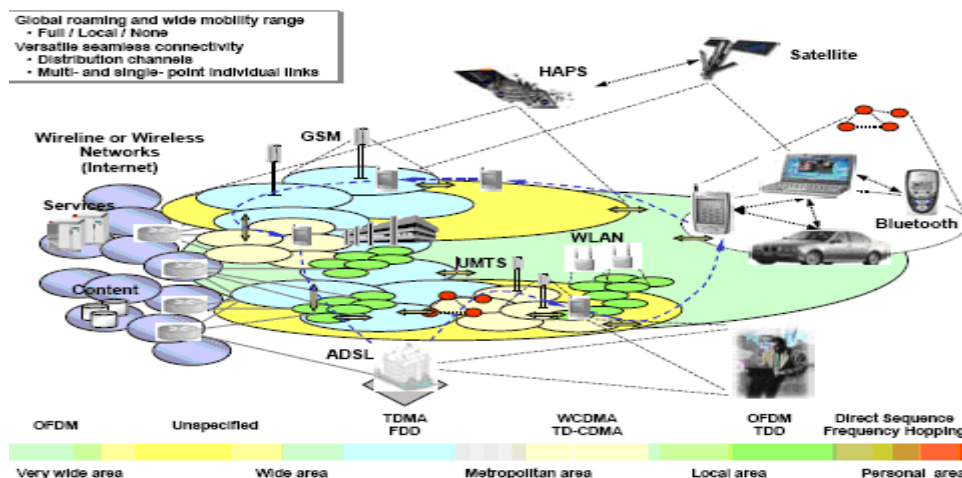


Figure 2: 4th Generation network

In the following Figure (Figure 3) Reconfigurability in 4G is illustrated in detail. As it was previously mentioned wireless communication systems require increased flexibility and implementation efficiency. Implementation platforms, new technologies and implementation hardware are used towards this scope. From Figure 3 as it can be easily observed there is the Value-Added Service Provider, the role of which is explained in Mobivas platform section, a Third-Party Application Provider and the Independent Application Developer. Furthermore there is a framework and various levels of middleware architecture. There is the Service Support Layer (SSL), the Messaging Transport and the Network Abstraction Layer. The successful cooperation of all the aforementioned entities and layers is vital in order reconfigurability to be achieved. The key is exactly this point: Reconfigurability can be achieved through middleware architecture, adapted to our needs in every case.

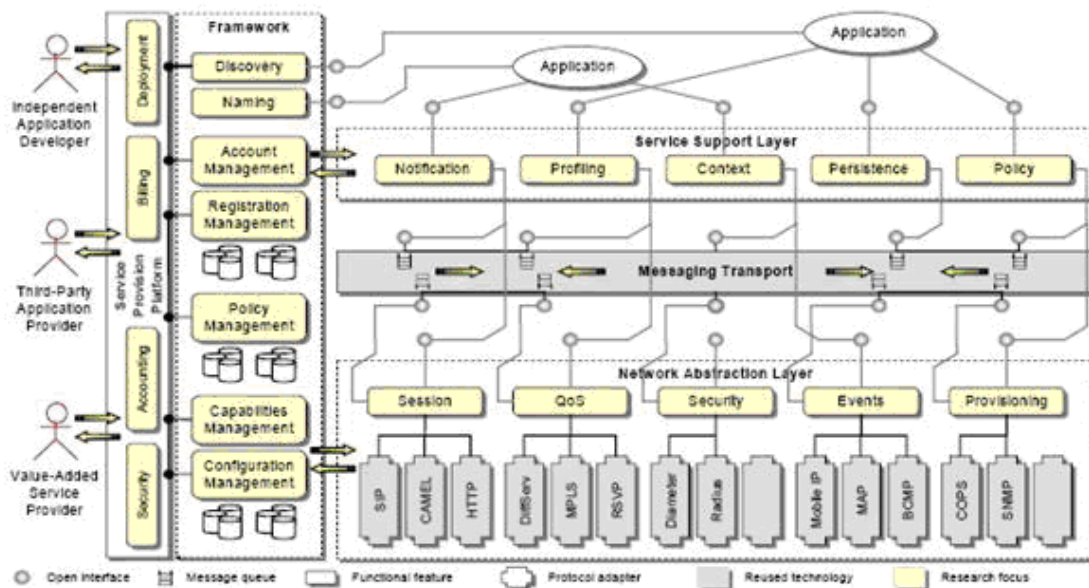


Figure 3: Reconfigurability in 4G

As far as the analysis of the demands of current wireless systems, **flexibility** is greatly required by wireless communication systems. The signal processing tasks add computational complexity to such systems. It has to be mentioned that instruction set processors are not sufficient and the processing requirements may only be handled by dedicated data path blocks. The use of dedicated data path structures could be restrictive only in the case where standards are not finalized at the time of the implementation or if a fast time-to-market is required. These are the cases where the reconfigurable hardware arises as the best and obvious choice. The main sources of flexibility requirements from a system point of view are presented in detail below [3]:

❖ **Upgradeability**

Equipment manufacturers move more and more towards solutions that can be upgraded in the field. Such an action gives them the ability to introduce not fully completed products versions for time-to-market reasons and as a consequence they have the ability to further extend products' lifetimes through substantial upgrades. The main reasons for such an action are mentioned below:

1. It is vital to be conformed to multiple international standards
2. The emergence of various improvements has to be implemented to standards
3. The addition of features and functionality to existing equipment is desirable
4. There is not so much certainty of the types of data services that will generate revenue in the wireless communications world
5. The introduction of bug fixing capability for hardware systems is a vital condition

This type of flexibility in most cases is covered by **Static reconfiguration**

❖ **Adaptivity**

The adaptation of wireless communication systems to dynamic factors in the operational environment is a desirable condition. These dynamic factors include the following:

1. Changing channel
2. Changing traffic
3. Changing applications
4. Power saving modes

This type of flexibility is mainly supported by **Dynamic reconfiguration**.

❖ **Expandability**

Many wireless systems such as outdoor wireless links may be based on chipsets that are basically built for mass market wireless equipment. A low price reconfigurable chipset after proper and suitable adaptation may be easily used as the core component for the architecture of a low volume product. This may vitally increase the market competitiveness of the corresponding product.

Finally, given a wireless communication system the task of identifying reconfigurability requirements from an implementation point of view can be easily translated and explained to the identification of the various systems' tasks that should be realized on reconfigurable hardware. Such an action consists part of an extended partitioning and assignment design procedure of the target system's tasks. These tasks are implemented in the different processing units of the architecture. Some basic guidelines that may have the ability to effectively drive the identification of tasks that should be realized on reconfigurable hardware are analyzed bellow.

Two basic features have to be combined for a given task in order the implementation of the task on reconfigurable hardware to be imposed: **Computational complexity** and **Flexibility need** [3].

- **Computational complexity:** Faster execution and reduced power consumption can be achieved by implementing a computationally complex task on reconfigurable hardware compared to that implementing on instruction set processors.

- **Flexibility need:** The implementation platforms that offer post-fabrication programmability, such as instruction set processors, reconfigurable hardware, can successfully satisfy the flexibility needs. Flexibility needs are created by two basic reasons: **Hardware time sharing** and **Upgradeability/expandability**.
 - ✓ **Hardware time sharing:** The action of sharing of the same hardware resources, within timing constraints, between different tasks may effectively reduce the cost of the target system.

Hardware sharing can be achieved in the following cases:

- Between tasks of a given system that are not simultaneously active (non-overlapping lifetimes).

- Between different instances of the same task, which correspond to different modes of operation of a specific, given, system.

- Between different algorithmic instances of the same task. For a variety of given tasks, it is very possible that more than one algorithm exist. The existence of this algorithm offers different algorithmic performance (for different modes of operation that take place) vs. the various implementation efficiency trade-offs.
- Between totally, in terms of functionality, different tasks of more than one systems that are realized on the same reconfigurable device with non-simplistic operation.
 - ✓ **Upgradeability/expandability:** Tasks of the target system can be upgraded in two different ways: either by adopting a more sophisticated and efficient algorithm or their functionality can be enhanced or expanded some time after the first version of the system has been successfully the developed. There is a need for the systematic design methodology support to be required. Such an action enables the efficient identification of the tasks of complex target systems that should be mapped on reconfigurable hardware.

Critical design methodology tasks should include the key characteristics that are mentioned bellow:

- a) Computational complexity estimation
- b) Task lifetime analysis
- c) Identification of mutually exclusive tasks
- d) Matching/comparison of pairs of tasks, which means the successful identification of differences between the various tasks of the system.

1.3 Mobivas platform

The Mobivas intelligent application platform [1] provides for the integration of basic entities, namely the contracted **Value-Added Service Provider**, the **Operator** and **network infrastructure**, the Access network and the end user. The Mobivas platform deals with major issues like reconfigurability, charging/billing, authentication, and security.

The general and basic objective of Mobivas (Mobile Value-Added Services) is the development of architectural approaches and prototypical implementations of integrated software platforms and systems. Such an action will enable the flexible provision of Value-Added Services (VAS) in mobile communication networks. These platforms can be effectively adapted to different network services and technologies, such as CDMA, OFDM, TDMA and CSMA, using different processing blocks, and constitute new opportunities for third party Value-added Service Providers (VASPs). The objective of Mobivas is not only the definition, design and development but also the validation of an integrated application architecture for downloadable Software Defined Radio (SDR) Value-Added-Services (VAS) and for innovative and modular network components for the efficient service provision in converging existing network [1].

More specifically, the activities that are assigned to Mobivas will be as they are mentioned below:

- ❖ First of all to access a framework for downloadable Value-Added Services, such as Mobile Data Services, Call Management Services and Push-to-Talk.
- ❖ Secondly, the platform has to contribute to the design and implementation of a framework which will enable efficient management of network resources to be provided.

- ❖ Additionally the platform must contribute to the design and development of mechanisms that will allow reliable switching between different wireless/mobile bearer technologies, taking into consideration the emerging software defined radio technology.

Overall, Mobivas's objective is the development of architectural approaches and prototypical implementations of integrated software platforms and systems, adaptable to different network services and technologies, which will open new opportunities for advanced Value Added Service Providers (VASPs). However, the key to the platform may be that: through this application of sophisticated mechanisms in order a required Quality of Service to be supported and by the introduction of novel methods for the development of innovative network elements, it manages to integrate existing technologies, such as GSM, DECT, Wireless LANs with upcoming ones such as GPRS and UMTS. As a consequence, new user-end technologies and applications can be easily implemented in this way like CDMA, TDMA, OFDM, and CSMA [1].

As far as the Operator and network infrastructure is concerned, the architectural component addresses technological issues. The proposed functional elements cater for open service provision and these services will be able to operate in dynamic usage conditions while the highest possible level of quality is maintained. An intelligent synthesis environment is proposed, which is based on reusable and extendable protocol. This will result not only in an integrated application architecture for downloadable software defined radio Value Added Services (VAS) but also in the introduction of an open, integrated Switching and Network Access architecture and application platform. This platform will enable the support of software defined radio features [1].

As far as the operational side, a charging and accounting model through an operator's network will be provided to the end-user. This will be achieved by contracted VASPs. Such an action will not only eliminate a lot of the administrative overhead for the VASP but will also simplify the total usage of multiple VAS by the end user. The key is that no separate contract agreements and billing with each VASP will be further required [1].

Power processing and memory are the basic limits that determine the choice of the technology that will support the development of MOBIVAS platform. These limits are imposed by the terminal [1]. Another significant aspect as far as the choice of the technology support is openness. Openness makes it easy for the various software engineers to develop new services, using standard languages and tools. Furthermore, it makes it easier for service providers not only the integration of new services in the VAS platform but also the prevention of interoperability problems. The best two candidates for the platform are CORBA and JAVA based mobile-agents. We have, moreover, to mention that the Intelligent Base Station will be designed to deal with the very demanding tasks of terminal classification negotiation, service negotiation and fast download of necessary service elements. On the other hand, the mobile user will be connected to available VASPs according to the type of terminal, access network and location.

In the following Figure (Figure 4) the deployment of the MOBIVAS service provision platform in a UMTS environment is illustrated.

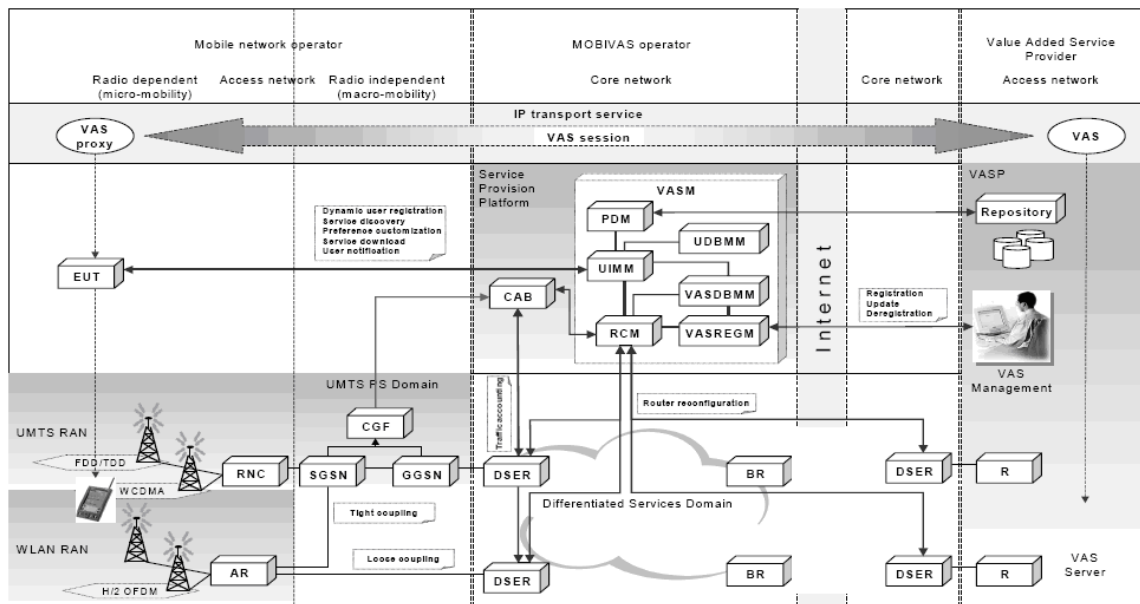


Figure 4: Deployment of the MOBIVAS service provision platform in a UMTS R5/R6 environment.

Chapter 2: Study and Analysis of the Reconfigurable Algorithms and Multiplexing Techniques

2.1 Media Access Control in Wireless Networks

Today wireless networks are ubiquitous. Most of the planet's population is covered by at least one wireless network, whether it is for use with a mobile phone, a Global Positioning System device, a laptop or even a TV remote, it is hard to find a household or an office without a single wireless network. The different types of wireless networks can be organized into 4 (four) groups in terms of their operating range. These groups of wireless networks are the following:

- ❖ Wireless Personal Area Networks (WPANs)
- ❖ Wireless Local Area Networks (WLANs)
- ❖ Wireless Metropolitan Area Networks (WMANs)
- ❖ Wireless Wide Area Networks (WWANs)

The operation range and the main characteristics of these groups will be explained bellow.

This variety in the usage of Wireless Networks has also led to the development of a large number of standards. Sometimes these standards apply to more than one category either by design or by innovation.

The Media Access Control is a sub-layer of the Data Link Layer (layer 2) in the specified seven-layer OSI model. It is responsible for providing addressing and channel access control mechanisms that allows several nodes to operate within a network. To address the issue two types of techniques can be employed, contention-free or contention-based [4].

2.1.1 Taxonomy of Wireless Networks

The different types of wireless networks can be organized into 4 (four) groups in terms of their operating range. These groups of wireless networks are listed below:

- ❖ **Wireless Personal Area Networks (WPANs)**
- ❖ **Wireless Local Area Networks (WLANs)**
- ❖ **Wireless Metropolitan Area Networks (WMANs)**
- ❖ **Wireless Wide Area Networks (WWANs)**

The operation range and the main characteristics of these groups are mentioned and explained below:

Wireless Personal Area Networks (WPANs)

These networks operate in a very small area, usually within the reach of a person, and hence the term Personal Area Networks is adopted. Common technologies and applications that correspond to Wireless Personal Area Networks are the following: Bluetooth headsets, wireless computer mice, remote controls.

Wireless Local Area Networks (WLANs)

This type of networks aims to replace the use of wired equivalent local area networks. They operate within the same distance constraints of a typical Ethernet network and are usually deployed either completely independent or in conjunction with a wired Local Area Network.

Wireless Metropolitan Area Networks (WMANs)

This type of networks has the ability to typically cover campuses, or even entire cities. One of the most advanced WMAN networks in Europe is the community owned Athens Metropolitan Wireless Network (AWMN).

Wireless Wide Area Networks (WWANs)

The Wireless Wide Area Networks, which are the last of the four wireless networks groups, are deployed in especially large scales. Typical examples in this case include the Global System for Mobile Communications (GSM) and the Universal Mobile Telecommunications System (UMTS) that are both used for mobile phones throughout the world.

2.1.2 Taxonomy of Wireless Networks based on range

This variety in the usage of Wireless Networks has also led to the development of a large number of standards. Sometimes these standards apply to more than one category either by design or by innovation – such as the AWMN using 802.11 technology for a WMAN network. Taxonomy of the most popular wireless standards is shown on the following Table (Table 1), organized using the aforementioned classification.

Protocol	WPAN	WLAN	WMAN	WWAN
Bluetooth [5]	X			
IrDA [6]	X			
Wireless USB [7]	X			
ZigBee [8]	X	X*		
DECT [9]		X		
HiperLAN [10]		X		
IEEE 802.11 [11]		X	X*	
HiperMAN [12]			X	
WiMAX [13]			X	X
GSM [14]				X
CDMA2000 [15]				X
UMTS [16]				X
HSPA [17]				X
LTEAdvanced[17]				X

* using extended multi-hop topologies

Table 1: Taxonomy of Wireless Networks based on deployment area

2.1.3 Channel Access Methods

The Media Access Control is a sub-layer of the Data Link Layer (layer 2) in the specified seven-layer OSI model [18]. It is responsible for providing addressing and channel access control mechanisms that allows several nodes to operate within a network. The addressing functions will not be examined as they are deemed out of scope and the focus will solely be on access control methods.

An analogy to the Multiple Access problem is a room where more than one person wishes to speak simultaneously. To address the issue two types of techniques can be employed, contention-free or contention-based.

Contention-free channel access methods involve the use of assignments to address who should take control of the medium. A few of the most popular contention-free techniques are:

- ❖ **Time Division Multiple Access (TDMA)** where the same frequency channel is divided into different time slots and each participating node in the network is assigned one or more of the slots to use [19].
- ❖ **Frequency Division Multiple Access (FDMA)** where the frequency spectrum is separated into several channels and each node is assigned one or more channels to use for communication.
- ❖ **Code Division Multiple Access (CDMA)** which employs a spread-spectrum technology to allow multiple nodes to be multiplexed over the same physical frequency using a special coding scheme where each node is assigned a code [20].
- ❖ **Token-based (Token Ring)** where token possession grants permission to transmit over the shared medium [21].

- ❖ **Polling techniques**, such as generalized polling, distributed polling, etc

The aforementioned access control methods are mostly used on static and networks with centralized control. All the algorithms summarized above have fixed assignments. Furthermore, most of them have been evolved to take into account demand on each node that creates combinations such as Demand Assigned TDMA (DA/TDMA), DA/FDMA etc. Flat and mesh ad-hoc wireless networks on the other hand tend to use contention-based access control methods. The most popular ones are mentioned and listed below:

- ❖ **Random Access Non-Carrier Sensing**, such as the ALOHA protocol that was developed by the Defense Advanced Research Projects Agency (DARPA) in the 1970s to facilitate the development of packet radio networks to use in battlefields, along with its ALOHA slotted variant [22].
- ❖ **Random Access Carrier Sensing**, such as the Carrier Sense Multiple Access (CSMA) with Collision Detection (CSMA/CD) as used in the original Ethernet protocol [23].
- ❖ **Control Packets Non-Carrier Sensing**, such as Multiple Access with Collision Avoidance (MACA) Protocol and its wireless variant MACA for Wireless (MACAW) [24].
- ❖ **Control Packets Carrier Sensing**, a set of protocols combining all the aforementioned technologies to produce Floor Acquisition Multiple Access (FAMA), part of the 802.11 MAC and Carrier Sense Multiple Access (CSMA) with Collision Avoidance (CSMA/CA) [25].

Evolution on the basic contention-based and contention-free protocols supplied new variations that take into account multi-channel communication, power awareness, QoS

metrics, directional antennas, receiver/sender initiated protocols, etc, which are deemed beyond the scope of this project.

Instead we will try to identify which of these access methods are used in each one of the main wireless technologies that were examined in Table 1. The following Table (Table 2) shows the Media Access Control method that is used in each wireless network protocol separately.

Protocol	Media Access Control
Bluetooth/1.0	TDMA
IrDA/1.0	Token-based
Wireless USB	TDMA
ZigBee/1.0	CSMA/CA along with Control Packets (ACK, Beacons)
DECT	FDMA and TDMA
HiperLAN/2	DA/TDMA
IEEE 802.11 AdHoc Mode	A Distributed Coordination Function (DCF) that includes CSMA/CA and Request-To-Send/Clear-To-Send control packets (RTS/CTS)
IEEE 802.11 Infrastructure Mode	In infrastructure mode it uses a Point Coordination Function (PCF) that complements DCF and adds an extra layer for polling functions
WiMAX	DA/TDMA with subscription tokens
GSM	TDMA with Frequency Hopping
CDMA2000	CDMA
UMTS	Wideband CDMA
HSPA	Wideband CDMA
LTE Advanced	Orthogonal Frequency-Division Multiple Access (OFDMA)

Table 2: Media Access Control per Wireless Network Protocol

As it is easily observed in Table 2 there is a large variation of Media Access Control methods that are used in current Wireless Network implementation. Unfortunately there is no obvious conclusion as to which contention-free or contention-based method is as best as it seems. The performance of the network is mainly based on a number of factors including the frequency, the type of the network (centralized or self-organized), the number of expected clients, the operating range and power demands.

2.2 Block Diagrams of the most common Reconfigurable Algorithms and Multiplexing Techniques

In this section we will analyze the most dominant and common reconfigurable algorithms and multiplexing techniques into block diagrams. This analysis will effectively help the observation of the basic characteristics and functions that correspond to each technique separately. Furthermore, a reference to the similar operations between these algorithms and multiplexing techniques will be carried out in order their main characteristics and reconfigurable operations to be possible to be found.

2.2.1 Time Division Multiple Access (TDMA) Block Diagram

In the following Figure (Figure 5) the Block Diagram of the **Time Division Multiple Access (TDMA)** is illustrated:

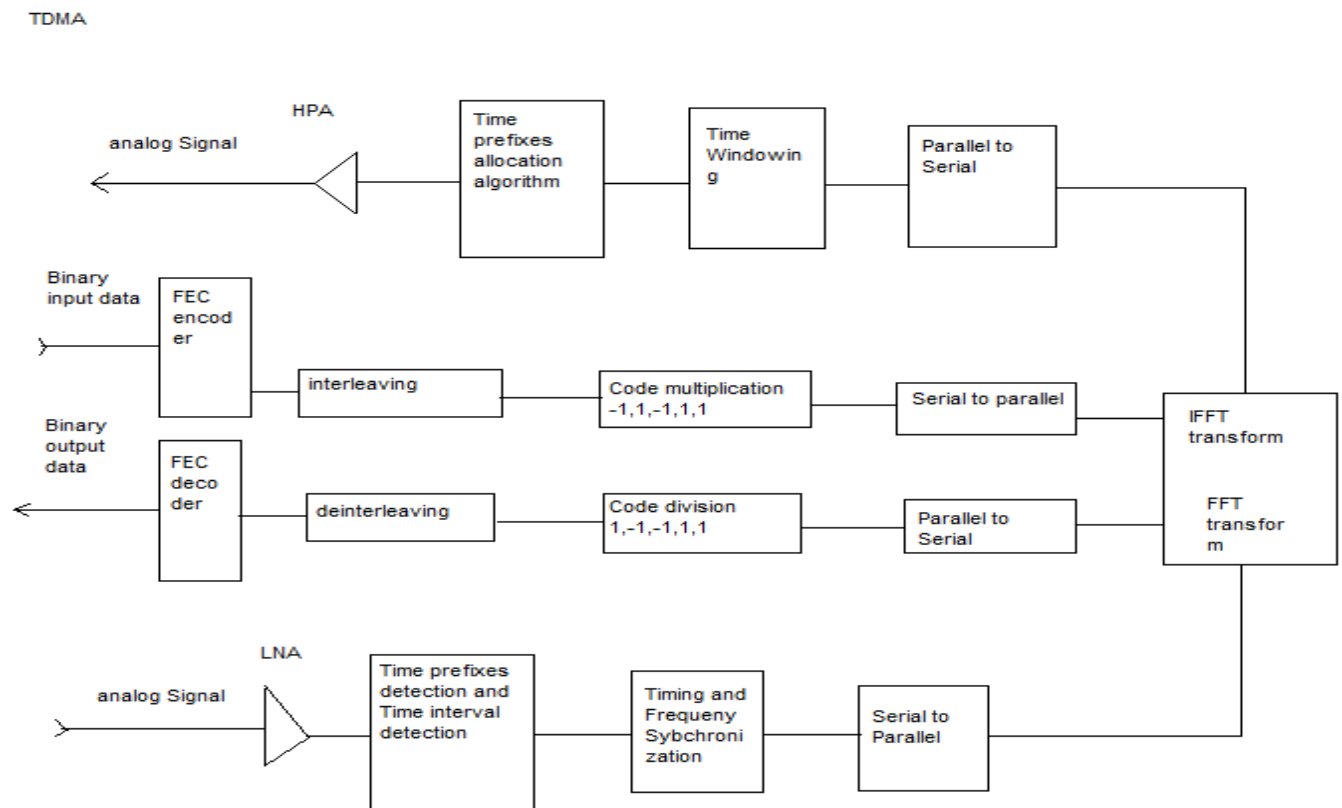


Figure 5: Time Division Multiple Access (TDMA) Block Diagram

As it can be easily observed by the above diagram (Figure 5), the input to the system corresponds to an analog signal. Immediately after, the signal is led to an amplifier. TDMA utilizes every time interval with different time interval prefixes in order to send and receive distinguishable messages. The receiver may encounter collisions unless a timing schedule is invented. The data is stored in a buffer and, with the help of a clock, depending on what time interval they arrived in, they are categorized and joined to the final messages. This scheme is mainly based in clocks and buffers that calculate the interarrival time and store the data with main aim the avoidance of possible collisions that may take place. The clock and the buffer may guarantee the stability of the system.

2.2.2 Orthogonal Frequency Division Multiplexing (OFDM) Block Diagram

In the following Figure (Figure 6) an **Orthogonal Frequency Division Multiplexer (OFDM)** is illustrated:

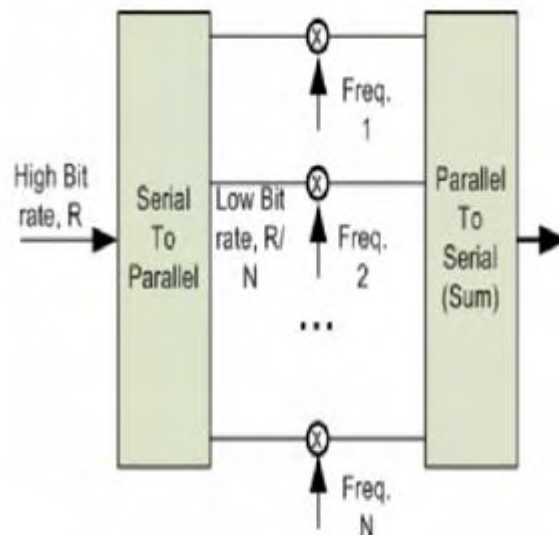


Figure 6: Orthogonal Frequency Division Multiplexer (OFDM)

The key point in Frequency Division Multiplexing is that the data can be transmitted at the same time. However, a different frequency carrier has to be assigned to each different message, usually orthogonal frequency to one another, so that the signals are not to be confused. This occurs because in the frequency domain case, the signals are not only separated but also occupy different frequency areas along with the multiplexing with each carrier.

In the following Figure (Figure 7) the amplitude of a signal along with the view of frequency and time domain can be easily observed:

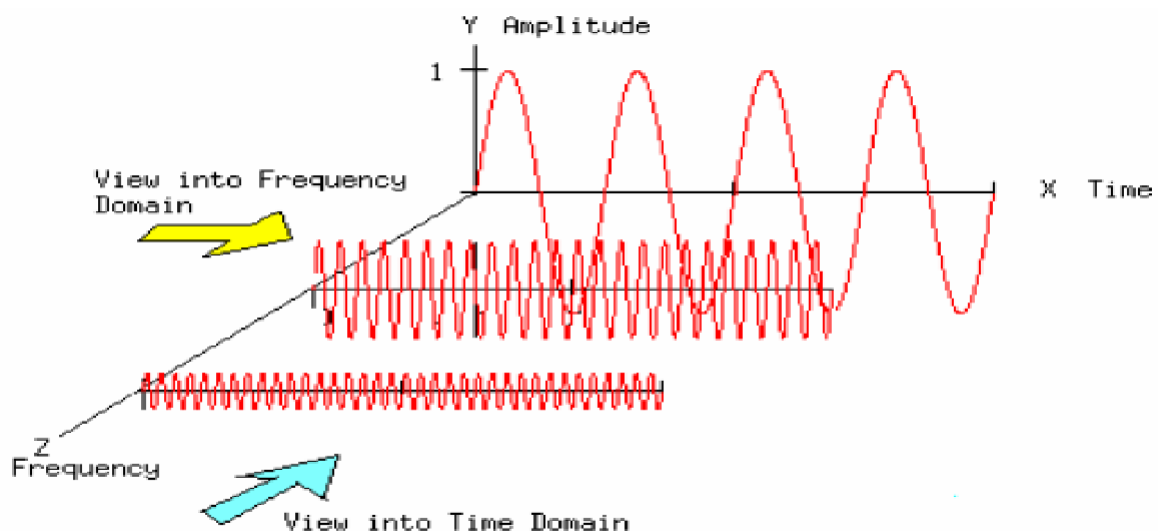


Figure 7: Frequency and Time domain view

In the case of Orthogonal Frequency Division Multiplexing we basically refer to QAM data multiplexing. In the case of QAM data multiplexing each signal is multiplied with a harmonic, a cosine, in a different frequency. As a consequence, even if the two signals travel together, they can never be combined in frequency domain. The key is exactly that it was previously mentioned. Each signal is multiplied with a harmonic cosine but in a different frequency.

In the following Figure (Figure 8) the Block Diagram of the **Orthogonal Frequency Division Multiplexing (OFDM)** is clearly illustrated:

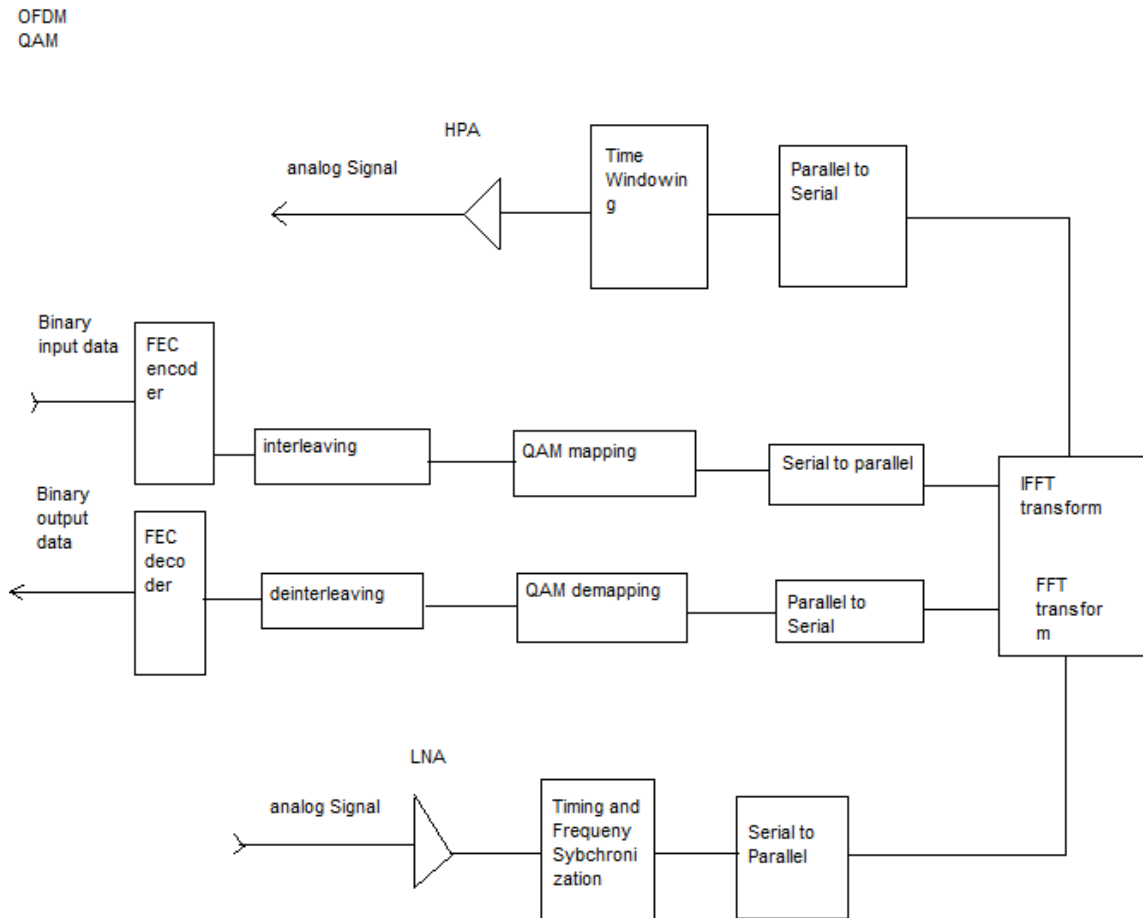


Figure 8: Orthogonal Frequency Division Multiplexing (**OFDM**) Block Diagram

2.2.3 Code Division Multiple Access (CDMA) Block Diagram

In the following Figure (Figure 9) a **Code Division Multiple Access Multiplexer (CSMA)** is clearly illustrated:

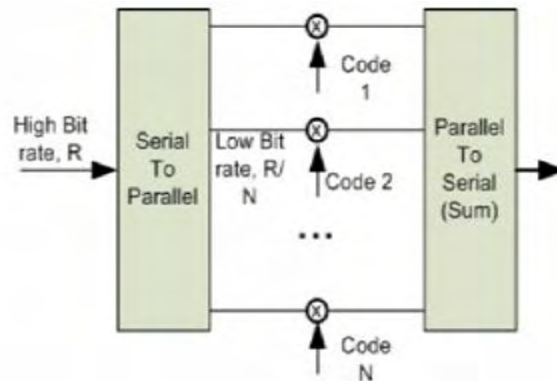


Figure 9: Code Division Multiple Access (CDMA) Multiplexer

The same process, as in the OFDM case that was previously explained in detail, is followed with Code Division Multiplexing. The only difference is that instead of multiplying each signal with a cosine of a unique frequency, which is orthogonal to others to the frequency domain, we multiply the signal with a unique code of +1s or -1s (for example +1,-1,+1,-1,-1,-1,-1,+1 or -1,-1,-1,-1,-1,-1,-1,-1). As a consequence, in this way two vital and important advantages can be achieved [26]:

- ❖ Firstly, each code is orthogonal to the other in time domain: This means that only with multiplying the right signal for a second time with the right code we can retrieve the signal. Multiplying it with another code of ones, will only distort it.
- ❖ Secondly, multiplying with a series of ones and minus ones makes our final signal wider and in frequency domain, below the noise level, it tends to be safer.

In the following Figure (Figure 10) the Block Diagram of the **Code Division Multiple Access (CDMA)** is easily observed:

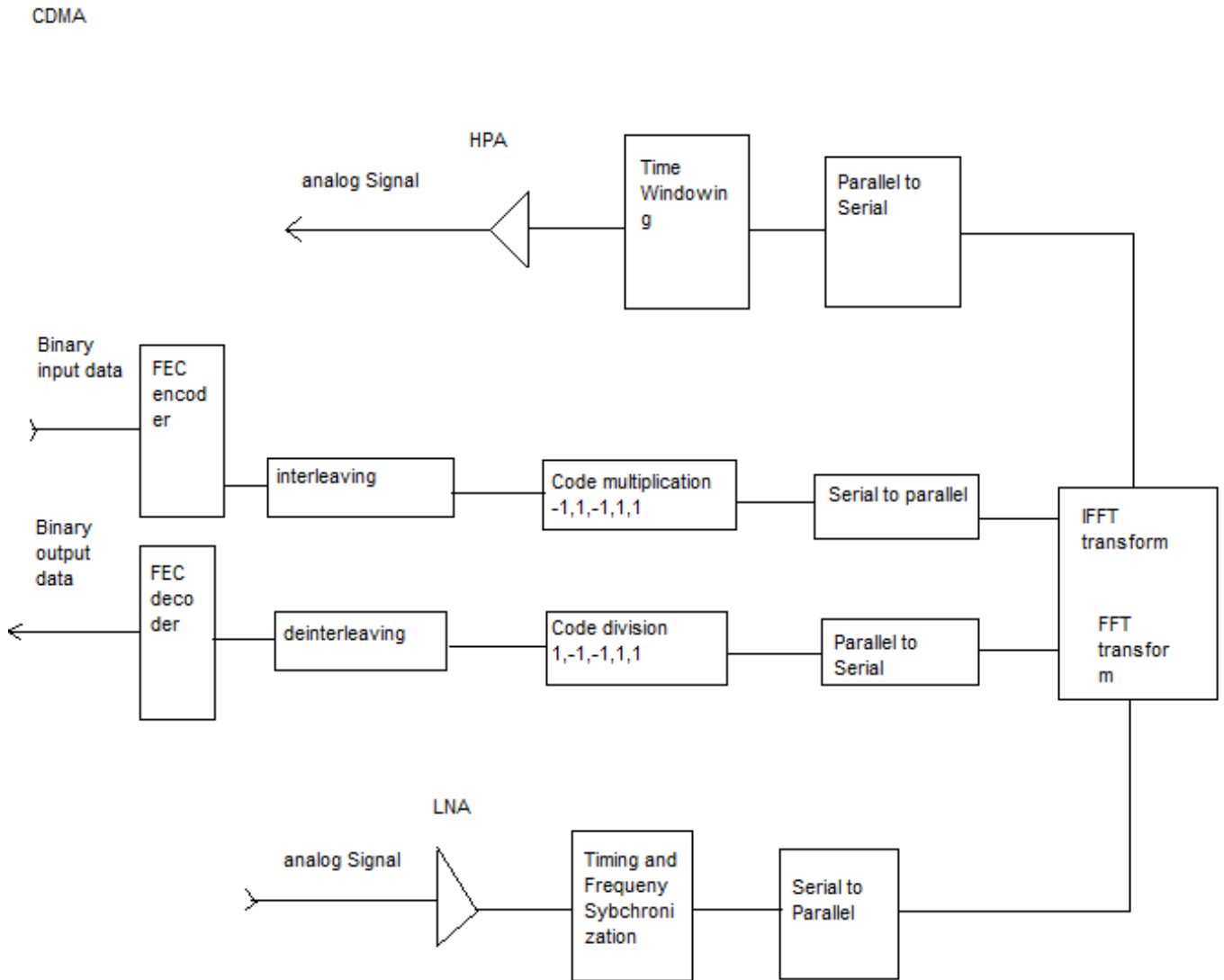


Figure 10: Code Division Multiple Access (CDMA) Block Diagram

2.2.4 Carrier Sense Multiple Access (CSMA) Block Diagram

In the following Figure (Figure 11) the channel access method in the **Carrier Sense Multiple Access (CSMA)** is depicted:

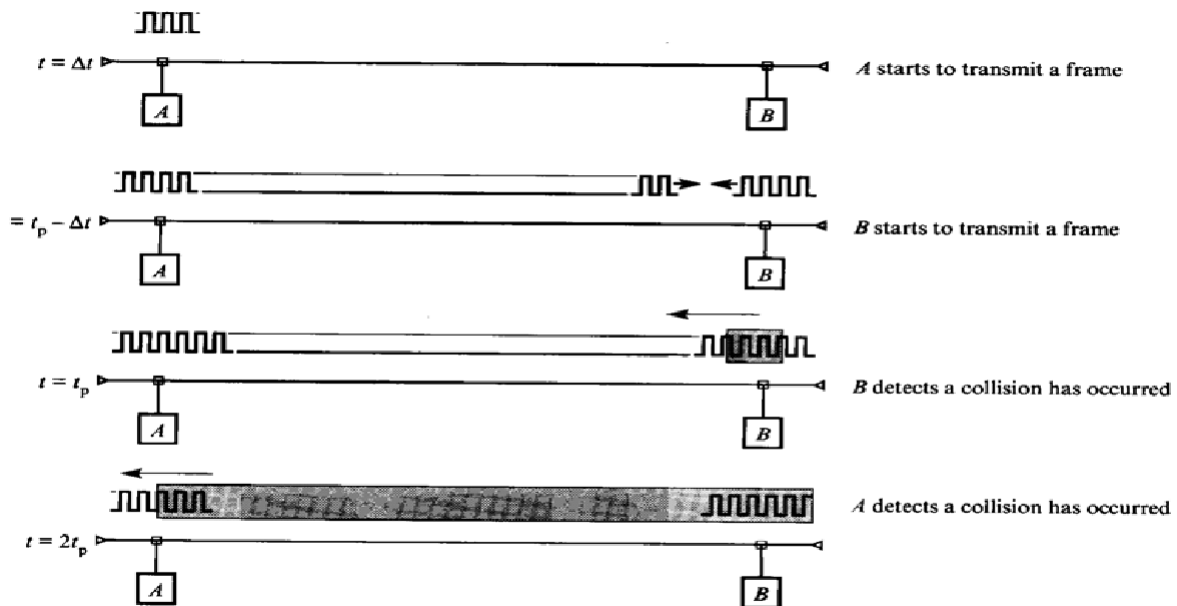


Figure 11: Channel Access method in CSMA

In this case, as it can be easily observed by the above Figure (Figure 11) when two or more stations wish to transmit concurrently in the medium this will merely lead to collisions and as a consequence to the congestion in the medium. In order the possible collisions to be avoided the Carrier Sense Multiple Access algorithm can be effectively implemented. The implementation of this algorithm can protect the medium from possible collisions that may take place when two or more stations try to send packets simultaneously.

In the following Figure (Figure 12) the Diagram of the **Carrier Sense Multiple Access (CSMA)** can be easily observed:

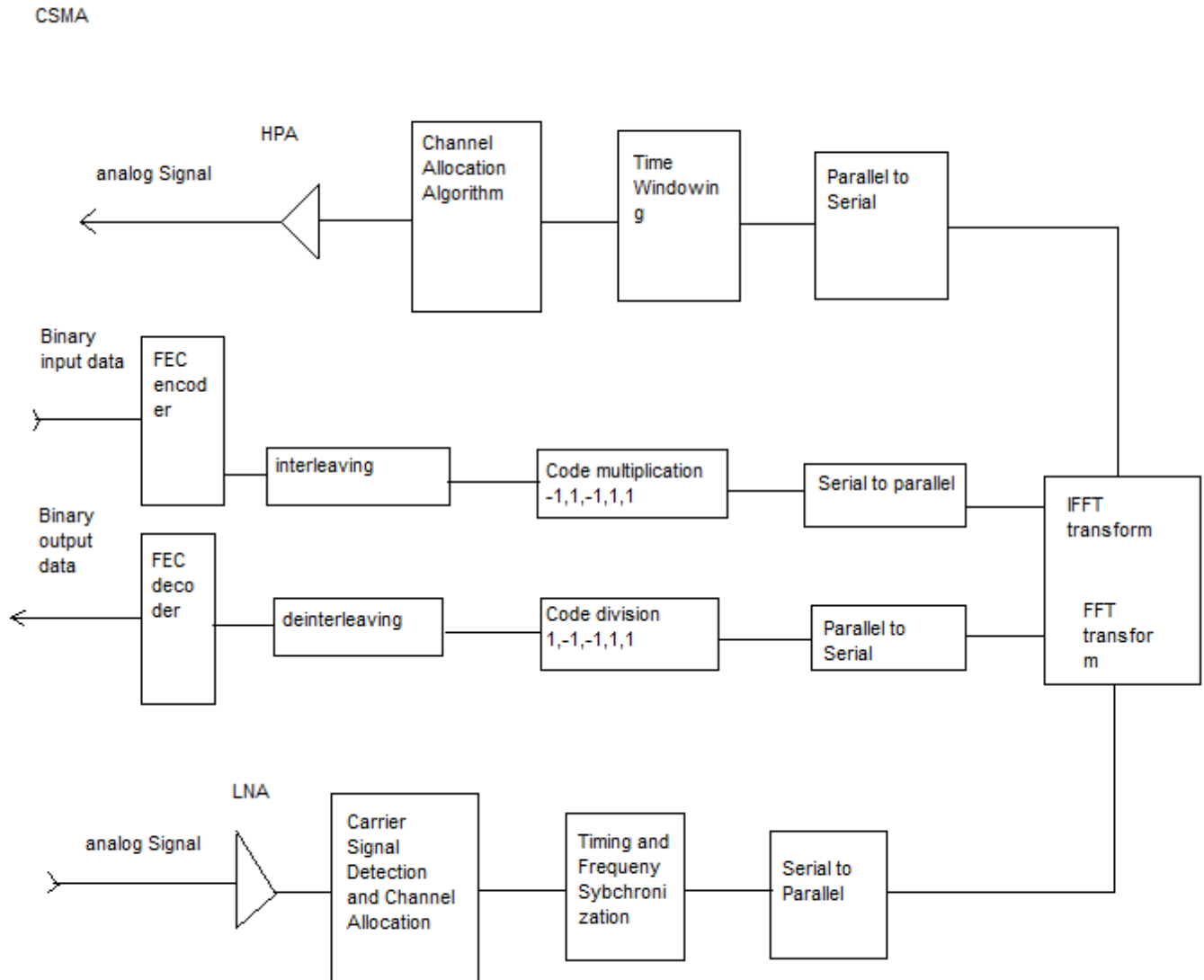


Figure 12: Carrier Sense Multiple Access (CSMA) Block Diagram

The important process of channel access is trying to avoid collisions, as soon as a collision in a channel is detected, whether it is in frequency domain or in time domain for example. As soon as the collision is recognized, usually through carriers or time intervals, the channel resource management takes place, either to send back notifications or trying to allocate the signal.

2.2.5 Combined Algorithm Block Diagram

As it can be noticed, by the analysis of the block diagrams of the most dominant and common reconfigurable algorithms and multiplexing techniques, some blocks that can be considered common to all multiplexers are **the amplifiers of the signal, the blocks of time and frequency synchronization as well as the interleaving and deinterleaving of the signal**. This interleaving of the signal corresponds to a data adding and transformation, which main aim is to make the signal closer to a uniform signal, which is hopefully clearer from noise and easier to be processed. As a consequence in these similar blocks is exactly where reconfigurability can be achieved.

In the following Figure (Figure 13) the **Combined Algorithm Block Diagram** can be easily observed:

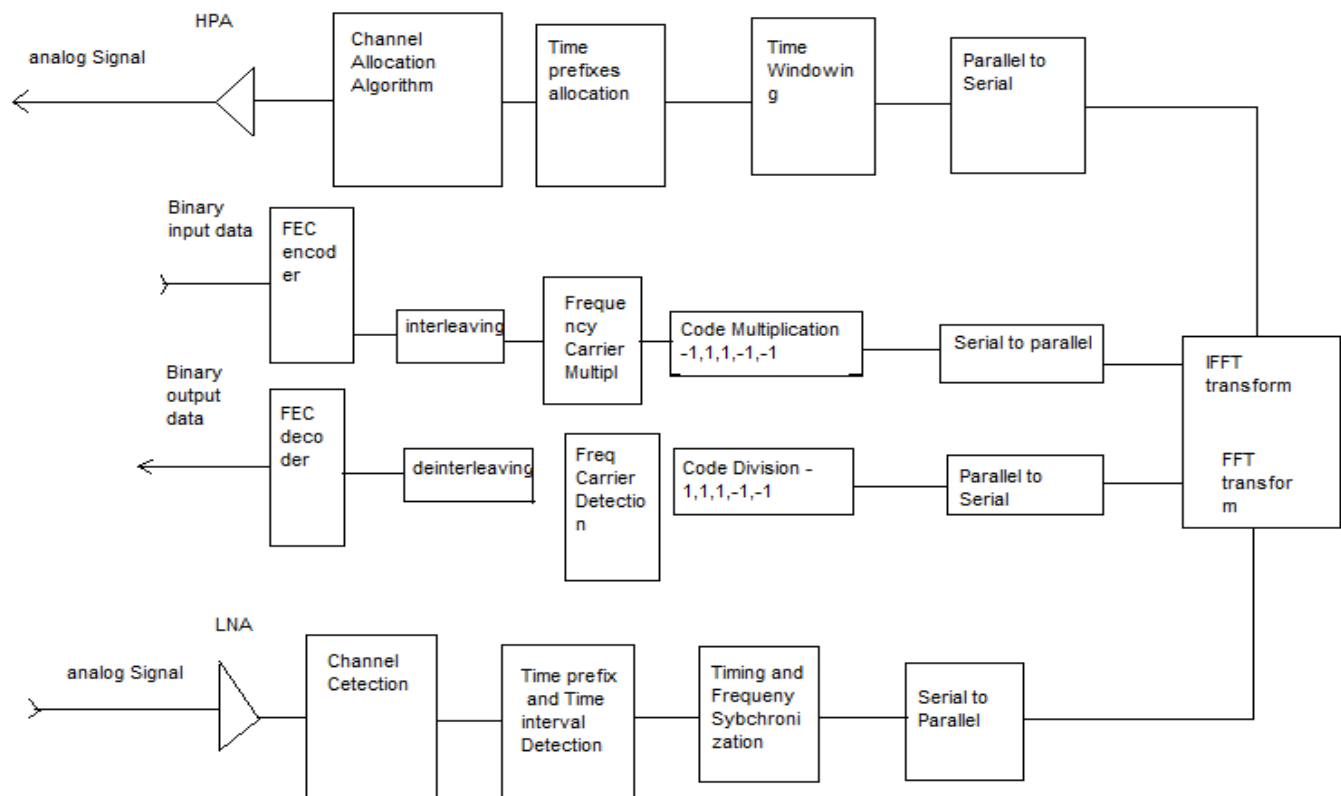


Figure 13: Combined Algorithm Block Diagram

In the above diagram (Figure 13), a combination of the main known blocks that comprise a current multiplexing system, can be easily observed. The four multiplexing algorithms in our work have been successfully combined in order to hopefully achieve further reconfigurability. Let us start from the lower left edge of the diagram (Figure 13). The analog signal, which has been transferred through a channel, for instance the atmosphere, or a closed room, enters and gets transferred to an amplifier, called **Low Noise Amplifier**, LNA. An RF amplifier corresponds to an active network that increases the amplitude of weak signals. As a consequence, it enables further processing by the receiver. Receiver amplification is distributed between RF and IF stages. An ideal amplifier increases the desired signal amplitude without adding distortion or noise [27]. Unfortunately, amplifiers are basically add noise and distortion to the signal. It has to be mentioned that the action of adding gain in front of a noisy network reduces the noise contribution from that network [26]. In the following Figure (Figure 14) a noise model that lets us calculate the amplifier noise figure is clearly illustrated:

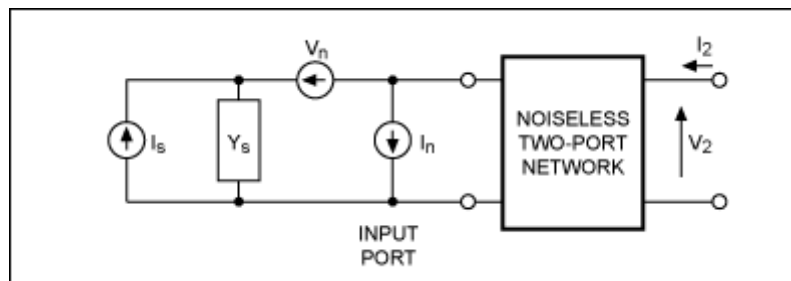


Figure 14: Example model figure enabling the calculation of the amplifier noise figure

A practical example to illustrate the theory of optimum noise matching for LNAs is the MAX2656, an LNA (Figure 15 below) with high third-order adjustable intercept point. Designed especially for PCS phone applications with gain selected by logic control, 14.5dB in high-gain mode and 0.8dB in low-gain mode, the amplifier exhibits an optimum noise figure of 1.9dB, depending on the value of bias resistor. The

MAX2655/MAX2656 IP3 is adjusted with a single external bias resistor, R_{BIAS} , which enables the optimization of the supply current for specific applications.

In the following Figure (Figure 15) a typical operating circuit for the MAX2656 LNA that shows design values for the input-matching network is clearly illustrated:

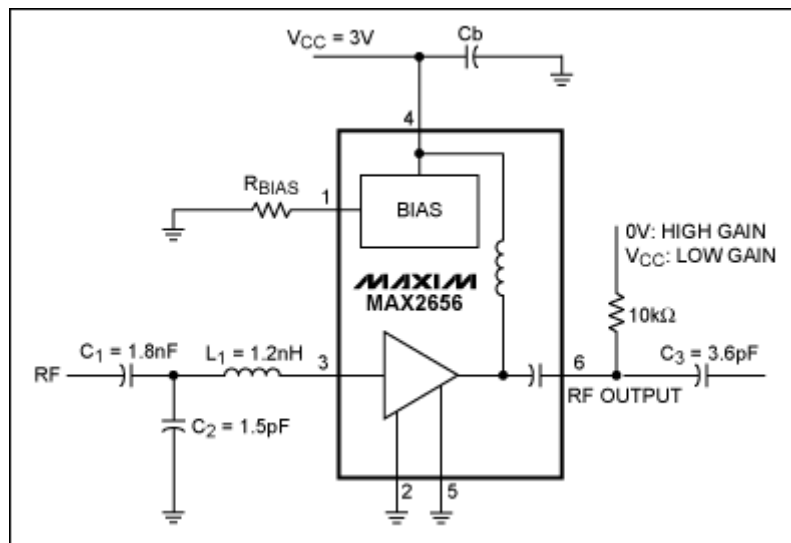


Figure 15: Typical operating circuit for the MAX2656 LNA

In the combined algorithm diagram, just before the **FEC Decoder**, and after the **FEC Encoder**, the **deinterleaving** and the **interleaving** process, during which input data units are distributed over a variety of output groups of data units, can be easily observed. Incoming data units are written to a contiguous RAM and output groups are read from said RAM [28]. After data has been read, these read locations are re-used for the storage of new input data. The duration over which a particular memory location stores a data unit depends upon the interleaving process delay for that particular data unit. The addressing circuitry includes modulo counters that are dedicated for the generation of addressing signals for a corresponding set of memory locations within the RAM. Look-up tables are used to select modulo counts in order conventional addresses to be provided to the RAM. In another block de-interleaving is performed during the writing of received bits to memory locations. However, said bits are written sequentially to said locations and

as a consequence enabling the remaining space to be used for other purposes. Said space may be effectively used for de-interleaving fast associated control channels. Bit position de-interleaving is thereby effected when the data is read either from the memory locations or from intermediate frame buffer.

In the case of data being transmitted over a radio telephony network from a computer, greater interleaving can be successfully used. A greater interleaving implies a longer delay time for certain parts that correspond to the original data signal before it is transmitted. The delayed data has to be stored either in a memory or buffer, and the amount of memory that has to be used for the data interleaving process may be great enough. This will significantly affect the price and power consumption of the equipment in which the interleaving process is considered to take place.

Interleaving and de-interleaving are achieved by providing a sufficient data memory for the storage of any data bit over the maximum length of time that is imposed by the interleaving process. However this is wasteful, due to the fact that the data supplied to the interleaving process is delayed by different times according to its eventual position in the interleaved bit stream output. This interleaved bit stream output is eventually transmitted. As a consequence, in known systems for interleaving and de-interleaving the full degree of efficiency which is implied in a theoretic way by the variable timing characteristic of the interleaving or de-interleaving process is not totally provided [29].

A **FEC Encoder** is a Forward Error Correction Encoder. Forward error-correction coding corresponds to a type of digital signal processing, which enables the improvement of data reliability with the introduction of a known structure into a data sequence prior to transmission or storage. This structure allows the detection and possibly the correction of errors by a receiving system caused by corruption from the channel and the receiver. As the name indicates, this coding technique gives the ability to the **decoder** to correct errors without requesting retransmission of the original information [30].

In a communication system that employs forward error-correction coding, a digital information source sends a data sequence to an encoder [31]. The encoder inserts redundant (or parity) bits, leading to an output of a longer sequence of code bits, called a codeword. Such codewords can then be transmitted to a receiver, which making use of a suitable decoder enables the original data sequence to be successfully extracted. The introduction of a large measure of redundancy by specific codes conveys relatively little information per code bit. This is an advantage due to the fact that the likelihood that all of the original data will be wiped out during a single transmission is reduced. On the other side, adding parity bits will generally increase transmission bandwidth requirements or message delay or maybe both of them [32].

Finally, an important block- procedure is the FFT block, the **Fast Fourier Transform block**. This block is used both for the discretion of the data and for the conversion to digital. The representation of discrete time signals by a sequence of numbers or symbols as well as the action of processing of these signals corresponds to Digital Signal Processing (DSP) [33, 34]. Both Digital signal processing and Analog signal processing are considered to be subfields of the general signal processing. Some subfields that are included in DSP are the following: audio and speech signal processing, sensor array processing, spectral estimation, statistical signal processing, digital image processing and signal processing for communications.

The main aim of DSP corresponds to the measurement, filtering and to the compressing of continuous analog signals. Firstly, a conversion from an analog to a digital form of the signal takes place. This conversion is achieved by originally sampling and in turn digitizing the signal with the use of an analog-to-digital converter (ADC). This converter turns the analog signal into a stream of numbers. At this point, it has to be mentioned that in most cases the required output signal is again another analog output signal. This means that this signal requires in this case a digital-to-analog converter (DAC). This process has to exhibit advantages and disadvantages. The main disadvantage is its complexity compared to the analog signal processing and additionally it exhibits a discrete value range. On the other hand the computational power that can be applied to digital signal

processing enables a variety of advantages over analog processing in many applications. Some characteristic instances are the error detection and correction in the transmission period as well as the compression of data.

Having combined our four methods of multiplexing algorithms, the signal follows the procedure described in the diagram (Figure 13) as the multiplexing has taken place in the following order: Firstly, different channels and antennas are used in order to exploit the different beams of the signal transmitted. After that signals received in different time intervals have different time arrivals as well as interval prefixes in order to be recognised. Following on, a different frequency carrier, which is a different cosine practically, is multiplied with the signal making it orthogonal to others in terms of frequency. And finally, whatever has come out as a result of the previous procedures, is multiplied with a different code of ones and minus ones (e.g. 1,1,1,-1,1,-1) in order to widen the signal frequency span and avoid the noise but also in order to make it orthogonal to others in terms of time. To be more specific, **TDMA utilizes every time interval with different time interval prefixes** in order to send and receive distinguishable messages. **OFDM multiplies every signal with a carrier orthogonal to other carriers in terms of frequency**. As a result, every signal lies in a different frequency interval. **Multiplication also takes place in CDMA**, but with a unique code for each signal which comprises of ones and minus ones, for example -1, -1, 1, 1, -1. **This widens the signal in terms of frequency and makes each signal again almost orthogonal to the other**. Finally, it has been proven that an **efficient allocation of channels and antennas** and a careful use of them, as in **CSMA**, can give us competitive results.

2.3 Reconfigurable Algorithms and Multiplexing Techniques

In this section an in detail reference to the most widely reconfigurable algorithms and multiplexing techniques that are widely used in wireless communications will be carried out. An in depth reference to their main characteristics and operations will take place and through diagrams and figures the way in which these multiplexing techniques operate in wireless communications will be shown.

2.3.1 Time Division Multiple Access TDMA

Time Division Multiple Access (TDMA) is a channel access method which enables the same frequency channel to be shared into several users with the division of the signal into different time slots. The users successively transmit in a rapid way, one after the other, each using a different time slot. An outlink control to each remote site separately is required for the operation of TDMA. This outlink control includes some control information. Furthermore, this outlink carrier has also a frame structure that offers accurate timing information for all the remote sites. The teleport hub equipment computer informs each VSAT site what particular time slot is to be used in the TDMA frame. It has to be mentioned that this time plan information is broadcast to all sites periodically. The burst time plan may either be fixed or dynamic. In the first case, a particular proportion of the total TDMA frame time is allocated in each site. In the second case the time slot that is allocated is adjusted in response to the traffic needs of each site. In the following Figure (Figure 16) the sequence between two successive TDMA frames is illustrated [35].

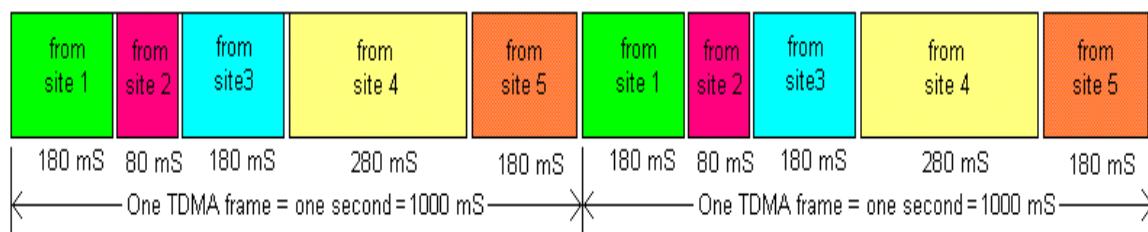


Figure 16: Sequence of two successive TDMA frames. Carrier bit rate 250 Kbit/s.

In this example that is shown in the above Figure (Figure 16), site 1 transmits a burst, starting at the beginning of each TDMA frame. The burst lasts 180 mS, so at a rate of 250kbit/s site 1 sends 45,000 bits per burst, or 45,000 bits per second. Site 2 transmits a burst, timed to arrive at the satellite just after the end of burst 1. The red, second, burst lasts 80 mS, so at a rate of 250kbit/s, site 2 sends 20,000 bits per burst, or 20,000 bits per second. In the above diagram (Figure 16) a fixed time plan is illustrated, where at each VSAT has been allocated a predetermined and particular portion of the total time. An amount of 20mS guard period between each burst has been designed. This allows for a little mistiming in the transmission of the bursts. Severe mistiming would cause bursts to arrive overlapping or on top of each other, causing loss of service to both sites involved in the mutual interference. The long 20mS guard period can be easily observed by the white space in the Figure (Figure 16) above. In TDMA systems the guard period may be very much less and there may be many more bursts per frame [35].

At this point it has to be mentioned that the circuit combines signals at the source, where the signals are transmitted, end of a communication link. This is known as a **multiplexer**. It accepts the input from each individual end user separately, then breaks each signal into different segments, and finally assigns the segments to the composite signal in a rotating and repeating sequence. As a consequence, the composite signal contains data from multiple transmitters. At the other end of the cable, the individual signals are separated out, in the reverse process, by means of a circuit which is known as **demultiplexer**. After the separation they routed to the proper end users [35].

A two-way communications circuit requires a multiplexer/demultiplexer at each end of the long-distance cable. If many signals have to be sent along a single long-distance line, a very careful engineering is demanded in order the proper performance of the system to be ensured. A very significant advantage of TDM is its **flexibility**. The variation in the number of signals that are sent along the line is allowed by the scheme and the time intervals are successively adjusted in order to enable the optimum use of the bandwidth that remains available. The Internet is a representative instance of a communications network where the volume of traffic can be drastically fluctuated from hour to hour.

2.3.2 Code Division Multiple Access CDMA

Code Division Multiple Access (CDMA) represents a technique that was suggested as channel access technology for 3G, and beyond this generation, wireless mobile services. In CDMA technique users are allotted with same frequency at the same time with different orthogonal spreading codes. The other users communicating at the same time create interference to each other. This interference is the mostly limiting factor of the system capacity. As a consequence, scheduling mechanisms are required for CDMA systems in order the limited system resources to be used in an efficient way [36].

One of the basic ideas in data communication is the concept of allowing a variety of senders to transmit information in a simultaneous way over a single communication channel. Such an action enables several users to share a band of frequencies (bandwidth). This concept is known as multiple access. CDMA offers spread-spectrum technology and a special coding scheme (where a code is assigned to each transmitter) to give the ability multiple and various users to be multiplexed over the same physical channel. Time Division Multiple Access (TDMA) divides access by time, while Frequency Division Multiple Access (FDMA) divides it by frequency. CDMA is a form of spread-spectrum signalling. This can be safely stated since the modulated coded signal has a much higher data bandwidth compared to that of the data being communicated [36].

CDMA is a form of Direct Sequence Spread Spectrum communications. In general, Spread Spectrum communications can be detected by three key elements [36]:

- ❖ The signal occupies a bandwidth much greater than that which is necessary to send the information. This leads to many benefits. Some characteristic instances are the immunity to interference and jamming and multi-user access.
- ❖ The bandwidth is spread by means of a code which is independent of the data. This independence of the code detects this one from standard modulation schemes where the data modulation will always spread the spectrum in a different way.

- ❖ The data recovery necessitates the receiver synchronization to the code. The use of both independent code and synchronous reception permits multiple users to gain access to the same frequency band at the same time.

In order the signal to be effectively protected, the code that is used is pseudo-random. It seems random, but is actually deterministic, so that the code can be reconstructed by the receiver for synchronous detection. This is also known as pseudo-noise (PN) [36].

In the following Figure (Figure 17) is shown the spreading way in terms of spectrum in CDMA. As it is easily observed, **as the pulses in time get shorter, the frequency bandwidth gets larger.**

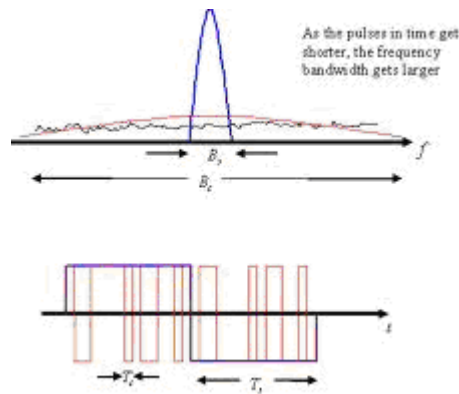


Figure 17: Spreading in terms of Spectrum in CDMA

In practice, what really happens is that CDMA spread spectrum technique multiplies the signal with orthogonal codes. This action is shown in the Figure 18 below. These orthogonal codes are known to the demultiplexer and this enables the spreading of the signal, in frequency terms, along a wide spectrum. This frequency band reaches so wide levels that normally falls below the levels of noise that correspond to an ordinary channel. However, it cannot be too wide because we have not only to avoid interfering with other existing signals but also transmitting on high frequency. The transmission in higher frequencies means higher power levels [36].

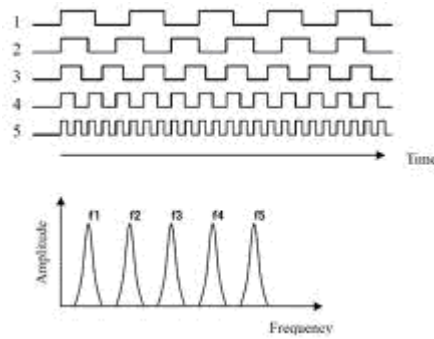


Figure 18: Examples of Codes of +-1 in CDMA

2.3.3 Orthogonal Frequency-Division Multiplexing OFDM

Frequency-Division Multiplexing (FDM) corresponds to a scheme in which a variety of signals are combined to be transmitted on a single communications line or channel. Its main characteristic is that at each signal is assigned a different frequency (sub channel) within the main channel. Furthermore, when FDM takes place in a communications network, each input signal is sent and received at maximum speed at all times. This is the chief, main advantage of this technique. On the other hand, if a lot of signals have to be sent along a single long-distance, the bandwidth that is demanded is large, and as a consequence careful engineering is required in order for the system proper performance to be guaranteed. In some systems Time-Division Multiplexing (TDM) is used instead.

Orthogonal Frequency-Division Multiplexing (OFDM) is a method of digital modulation which main characteristic is the split of a signal into several narrowband channels at different frequencies [3]. OFDM exhibits some similarities with the conventional frequency-division multiplexing (FDM). The difference basically lies in the modulation and demodulation of the signals. The biggest priority is assigned to the minimization of the interference, or crosstalk, among the channels and symbols comprising the data stream.

OFDM uses many carrier frequencies where each transmitting a section of an input data sequence. This operation allows OFDM to achieve its high data-rate capabilities. The carrier frequencies may have the ability to be spaced as close together as it is possible, yet they do not interfere with one another. These carrier frequencies are known as orthogonal carriers. Due to its frequency diversity and transmission structure characteristics OFDM overcomes better than single carrier transmission systems in a channel environment where multipath fading often takes place. In the following Figure (Figure 19) the OFDM modulation is illustrated.

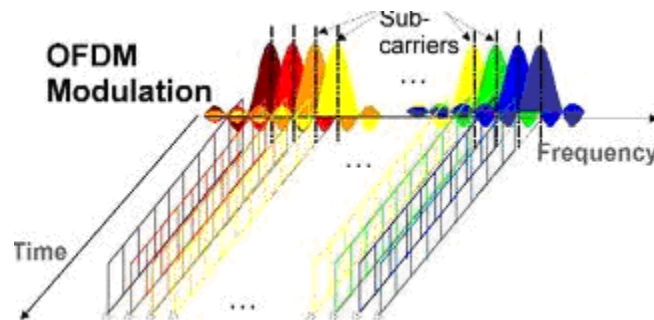


Figure 19: OFDM Modulation

As it was previously mentioned and it is easily observed by the above Figure (Figure 19) Orthogonal Frequency-Division Multiplexing (OFDM) is a method of digital modulation. This method corresponds to a modulation in which a signal is split into several narrowband channels at different frequencies. Additionally, from the above Figure (Figure 19) it is observed that OFDM uses many carrier frequencies in unison, each transmitting a section of an input data sequence. This is a very important operation which enables OFDM to achieve its high data-rate capabilities [3].

OFDM is used in European digital audio broadcast services. The technology lends itself to digital television. It has to be mentioned that OFDM is considered as an ideal method of obtaining high-speed digital data transmission over conventional telephone lines. Furthermore, it is also used in wireless local area networks.

2.3.4 Carrier Sense Multiple Access CSMA

Carrier Sense Multiple Access (CSMA) is a probabilistic Media Access Control (MAC) protocol in which a node confirms the absence of other traffic before transmitting on a shared, common, transmission medium, such as a band of the electromagnetic spectrum [37].

Carrier Sense describes the fact that a transmitter uses feedback from another receiver that detects a carrier wave before trying to send. This is the key point: it tries to detect the presence of an encoded signal from another station before starting the transmission attempts. If a carrier is sensed, then the station waits for the transmission that is already in progress to finish. After it has finished, then its own transmission is initiated [38, 39].

Multiple Access refers to the fact that multiple stations not only send but also receive on the medium. Transmissions that correspond to one node are generally received by all other stations that make use of the medium [38, 39].

Carrier Sense Multiple Access/Collision Detection (CSMA/CD) is the protocol for carrier transmission access in Ethernet networks. On Ethernet, any device has the ability to try to send a frame at any time. Each device senses whether the line is idle. After this sensing it recognizes if the medium is available to be used. If it is, the station begins to transmit its first frame. On the other hand, if another station has tried to send at the same time with the first one, a collision takes place and the frames are discarded. Each station then waits a random amount of time. After this time, it retries until successful in getting its transmission sent [23, 40].

Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) corresponds to a widely used protocol in systems where the medium (channel) is sensed by the user and through this sensing, tries to determine if the access to the medium is allowed or not. The outcome result of this sensing enables the user to understand if can gain access to the medium. Vital wireless standards, such as ZigBee, have already adopted CSMA/CA and as a result, it has reached high levels of importance due to this adoption [41, 42].

At this point, in order to understand the operation of CSMA/CA in a more successful and effective way, we will give the operation of the Media Access Control of the IEEE 802.11 Ad hoc Mode. This is based on a Distributed Coordination Function (DCF) that includes CSMA/CA and Request-to-Send/Clear-to-Send control packets (RTS/CTS). We are mentioning to this mechanism because it is convenient for ad hoc or other configurations that assume exchange of bursty traffic. The operation of CSMA/CA can be described as follows [43]:

- 1)** The station with the frame to send senses the medium,
 - a)** if it is idle, the station waits for IFS and then do **2)**
 - b)** if it is busy, it does the **2b)**

- 2a)** if the medium is idle again, then the station transmits immediately.

- 2b)** if the medium is busy, the transmission is deferred and the station has to monitor the medium until the current transmission is terminated. Then go to **3)**

- 3)** The station waits for another IFS
 - a)** if the medium is idle during IFS, a back off random amount of time takes place and then senses the medium again.
 - b)** if the medium becomes busy, then go back to **3)**

In the following Figure (Figure 20) a diagram of the operation of CSMA/CA is illustrated in detail:

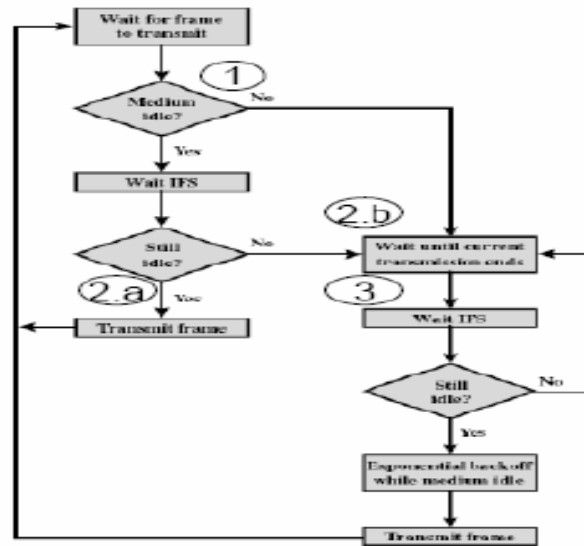


Figure 20: The diagram of the operation of CSMA/CA

In the following Figure (Figure 21) the operation of slotted CSMA/CA for a two-user example system is clearly illustrated [44, 45]. The steps of this operation are these that were previously mentioned in the explanation of the CSMA/CA [46, 47, 48].

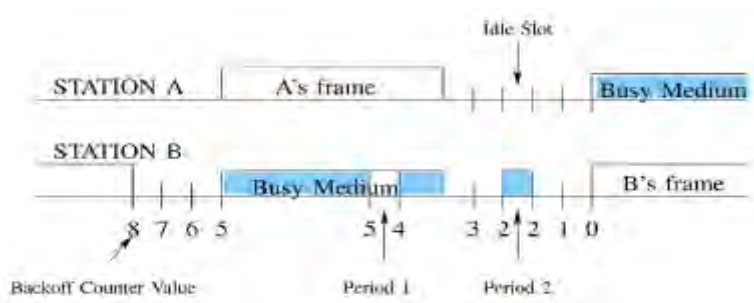


Figure 21: Operation of the CSMA/CA basic-access protocol

As far as the IFS 3 (three) values are possible:

- ❖ Short IFS (SIFS)
- ❖ PCF IFS (PIFS)
- ❖ DCF IFS (DIFS)

The SIFS is used for highest priority packets such as the following:

- ACK
- CTS
- Response to CTS

PIFS is used for PCF operation. On the other hand, DIFS is used for longest IFS that has to do with contention data spacing [49, 50].

In terms of priorities, the PIFS duration is defined as the SIFS duration plus 1(one) time slot. On the other hand DIFS is defined as SIFS plus 2 time slots [51]. In the following Figure (Figure 22) the transmission priorities in MAC obeying to that scheme are clearly depicted.

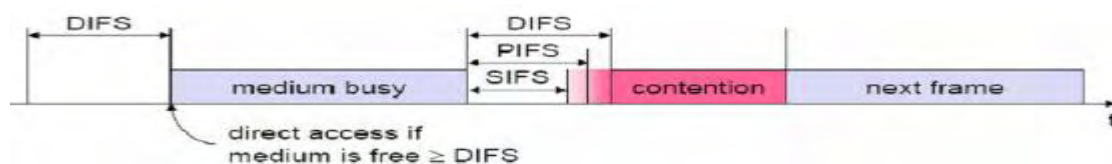


Figure 22: Transmission priorities in MAC

Under conditions of high offered traffic loads, collisions occur in most cases in the most basic random-access system which is the ALOHA system. Such an action not only reduces the throughput but may also lead to instability if collided packets collide more than one time during their retransmission. In order this problem to be solved a number of other protocols have been proposed towards this direction.

Reconfiguration between the different congestion avoidance and control algorithms for 4G handsets

Busy Channel Multiple Access (BCMA) corresponds to the class of multiple access schemes in which new packet transmissions are not (permitted) allowed when the inbound channel is busy. A big variety of strategies have been proposed in order information on the channel state to be successfully acquired. Carrier Sense Multiple Access (CSMA) does not employ feedback and positive acknowledgements of correct reception of a data packet. The key characteristic is that all terminals have the ability listen to the inbound channel. In the case where the inbound channel is notified (sensed) busy by the mobile terminal, it means that new packet transmission is impossible to be initiated. In order for this to be achieved the vital condition is that all mobile terminals can receive each others signals on this inbound frequency [52].

There are cases, especially in mobile radio nets with fading channels, where a mobile terminal might not be able to sense a transmission by another terminal. This effect is known as the "hidden-terminal" problem. The **Inhibit Sense Multiple Access (ISMA)** gives solution to this problem. In this case the base station transmits a busy signal on an outbound channel. The goal of this action to avoid all other mobile terminals from transmitting during an inbound packet is being received. The main drawback of ISMA is that a real-time (continuous) feedback channel is merely needed. It has to be mentioned that collisions can occur in ISMA despite of the fact that signalling messages on the feedback channel may always received in a direct and correct way by all mobile terminals. This occurs due to the fact that new packet transmissions can take place during the reception delay of the inhibit signal or packets from various persistent terminals can collide immediately after a previous packet transmission has been terminated [53].

Busy Tone Multiple Access (BTMA) has also been proposed. In the case where the feedback channel contains a narrowband "busy tone", then the mobile terminal may do not notify the presence of this specific tone. The solution to this problem may be given by casting an idle tone, rather than a busy tone. Furthermore the retransmission of busy reports accompanied with error control coding consist a vital solution to the aforementioned problem [54].

Chapter 3: Simulation

In this part the simulations of the aforementioned basic algorithms will take place. The Simulations that take place are developed with the help of OMNEST++ Network Tool by Andras Varga through the basis model of Aloha network. In the following sections is given an in detail presentation of these simulations for each one of the aforementioned algorithms separately. These simulations cover the implementation of different scenarios that correspond to the operation of each algorithm separately.

3.1 OMNEST Network Simulation Tool

OMNEST is used by RD staff, researchers and engineers worldwide to investigate various scenarios and design alternatives such as architectural designs, wireless or wired protocols and networks, queuing-based and other systems. OMNEST provides building and evaluating simulations in an integrated Simulation IDE. We can also embed simulations in our own software products. OMNEST is the commercial version of OMNeT++, one of the most popular network simulation frameworks in academic and research communities.

OMNEST is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains:

- modelling of wired and wireless communication networks
- protocol modelling
- modelling of queuing networks
- modelling of multiprocessors and other distributed hardware systems
- validating of hardware architectures
- evaluating performance aspects of complex software systems
- in general, modelling and simulation of any system where the discrete event approach

- is suitable, and can be conveniently mapped into entities communicating by exchanging messages

OMNEST itself is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is component architecture for simulation models. Models are assembled from reusable components termed modules. Well-written modules are truly reusable, and can be combined in various ways like LEGO blocks.

OMNEST simulations can be run under various user interfaces. Graphical, animating user interfaces are highly useful for demonstration and debugging purposes, and command-line user interfaces are best for batch execution.

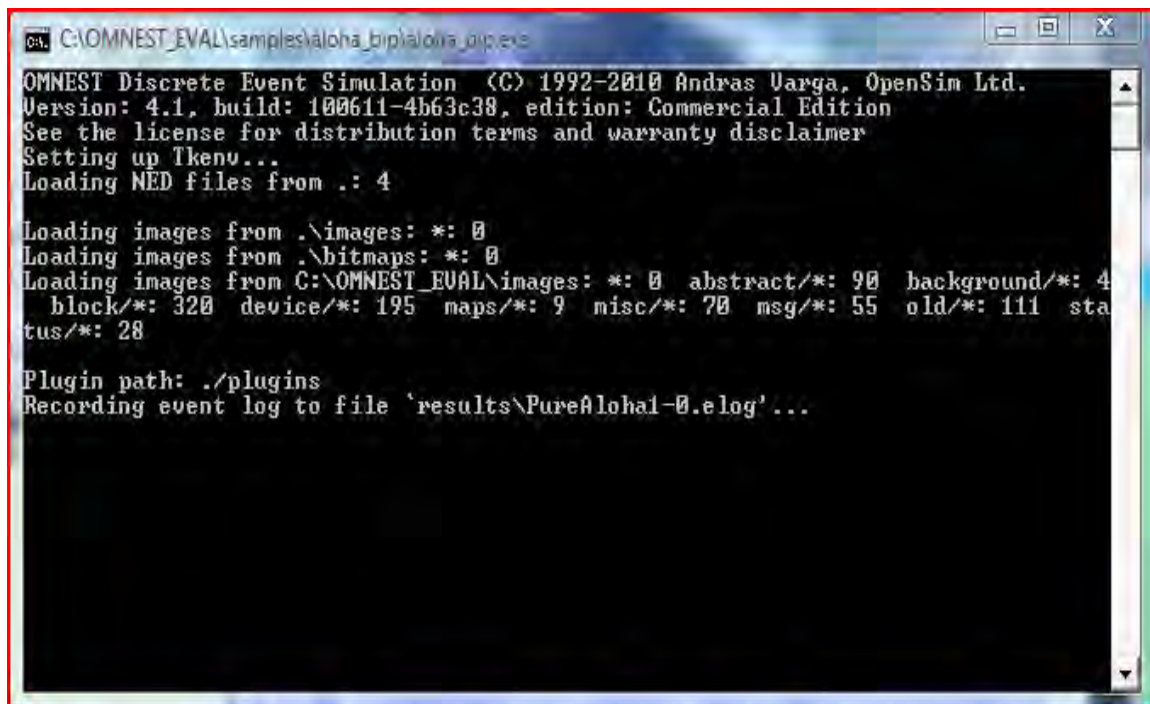
The simulator as well as user interfaces and tools are highly portable. They are tested on the most common operating systems (Linux, Mac OS/X, Windows), and they can be compiled out of the box or after trivial modifications on most Unix-like operating systems. Below, in Figure 23, the Aloha network which is a sample of mobile communications of the program is illustrated.



Figure 23: Aloha example network with OMNEST

3.2 Simulation of Time Division Multiple Access TDMA

The Simulation of a simple network using Time Division Multiplexing is achieved through the basis model of Aloha network of OMNEST EVAL or OMNET++. In this case the network is supplemented with the addition of TDMA multiplexing in a simple way. In the following Figure (Figure 24) the running of TDMA Aloha simulation through executable in mgnv is illustrated.



```
C:\OMNEST_EVAL\samples\aloha_bip\aloha_bip.exe
OMNEST Discrete Event Simulation (C) 1992-2010 Andras Varga, OpenSim Ltd.
Version: 4.1, build: 100611-4b63c38, edition: Commercial Edition
See the license for distribution terms and warranty disclaimer
Setting up Tkenv...
Loading NED files from .: 4
Loading images from .\images: *: 0
Loading images from .\bitmaps: *: 0
Loading images from C:\OMNEST_EVAL\images: *: 0 abstract/*: 90 background/*: 4
block/*: 320 device/*: 195 maps/*: 9 misc/*: 70 msg/*: 55 old/*: 111 sta
tus/*: 28
Plugin path: ./plugins
Recording event log to file 'results\PureAloha-0.ealog'...
```

Figure 24: Running TDMA Aloha Simulation through executable in mgnv

In the following Figure (Figure 25) the files that exist in the File of TDMA Aloha example can be easily observed. As we shall see, in every different case of multiplexing the basic existing files are the same and their code is the only one that is modified in each case.

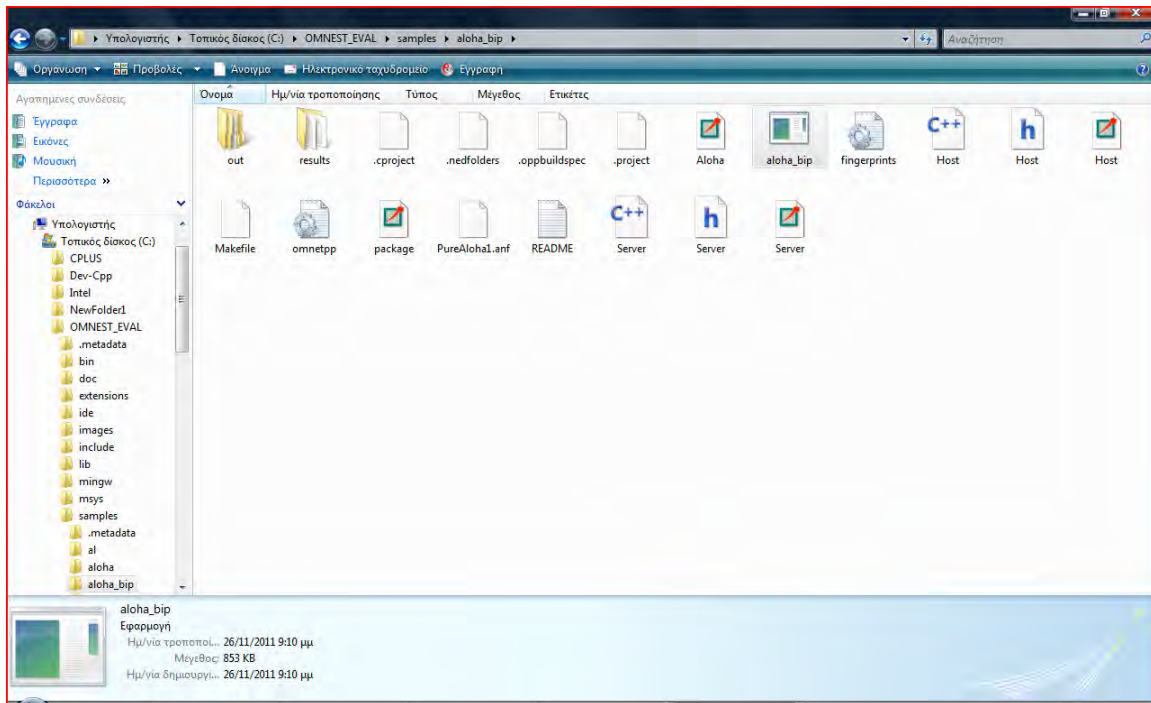


Figure 25: Files in the file of TDMA Aloha example

The code that is used in each one of the files that exist in the TDMA Aloha example separately, which can be easily observed in the Figure 25 above, is presented in the appendix of this project.

In this way if another packet arrives to the server while the receiver is already occupied with another one, that has been arrived earlier, then the second message is dismissed and the throughput at that moment is only one message out of two. The collision case can be easily notified by the addition of an integer. This means that an integer value is used in order to notify us every time a collision of packets takes place. This pointer increases only if a packet makes it through without colliding with another one during the transmission time, therefore **the throughput is calculated as the value of this pointer**. We have named our pointer **bu** as it can be easily observed in the following Figure (Figure 26) of our application.

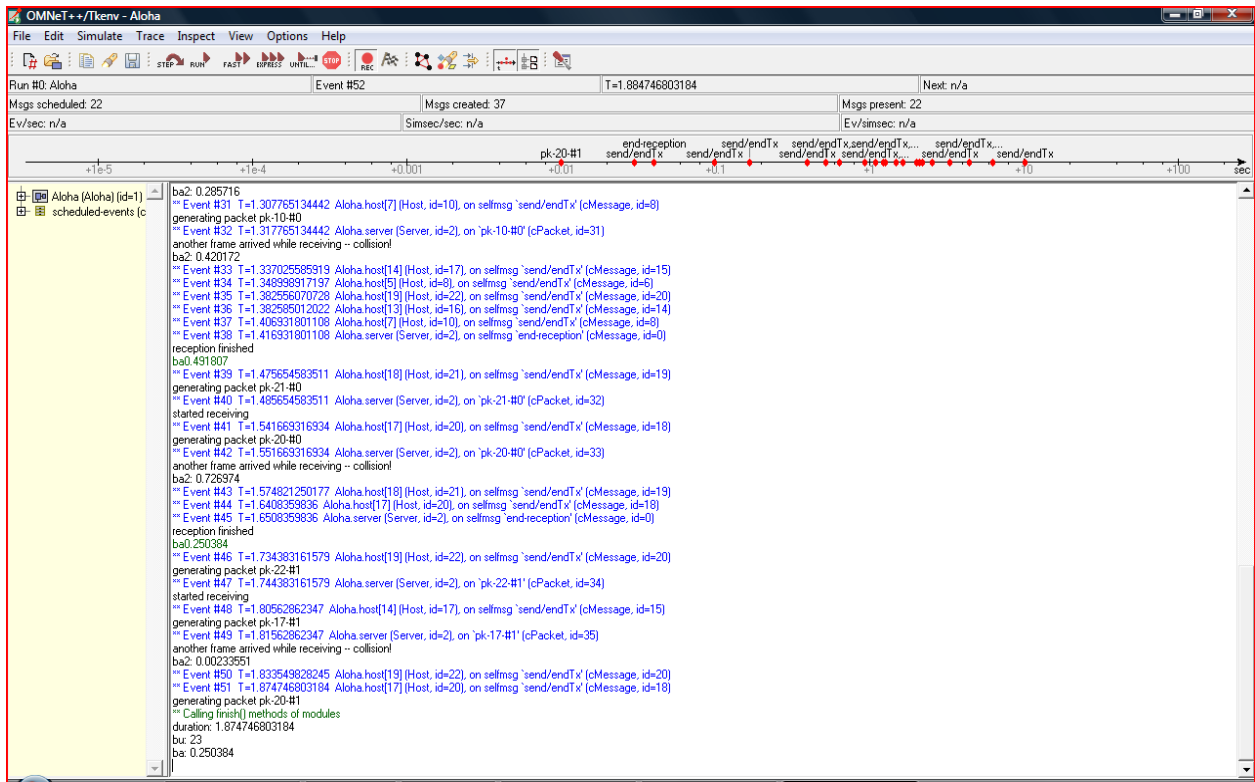


Figure 26: Total number of collisions and consequently of lost packages indicated by the pointer bu

As it can be easily observed by the above Figure (Figure 26), immediately after we call finish on the simulation, the total number of collisions and consequently of lost packages is (bu) 23 out of the total of 51 transmitted packets. In this run, the throughput is merely 23. However, as the transmission value is considered to be stochastic, it has to be mentioned that we conducted the simulation multiple times and we gathered some vital results as shown in the diagrams that are illustrated in the Figures bellow.

In the following Figure (Figure 27) the **Throughput** that is derived over the offered load is clearly illustrated:

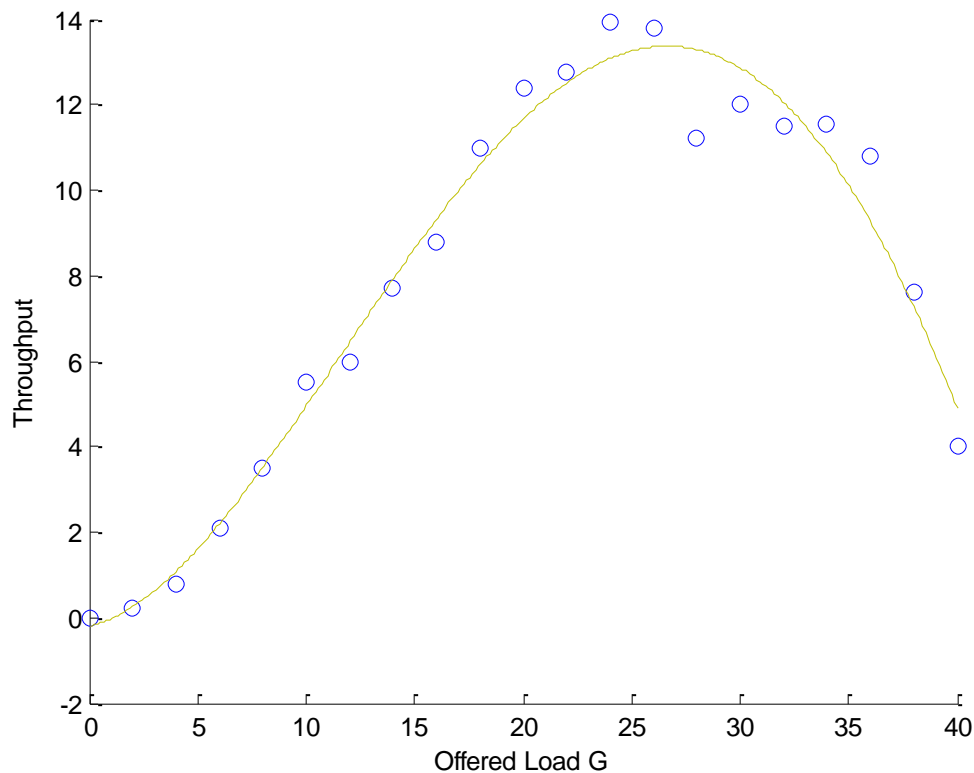


Figure 27: Throughput results obtained after the conduction of the TDMA simulation

The diagram is a plot of the observed values after a basic polynomial fitting has been conducted. As it can be easily observed by the above diagram, the graph is centered at approximately 20 packets, which means that there is better service when the load is around 20. The curve is decreased on the left and the right as expected. As we move beyond 20 packets, the throughput is decreased because the load obviously, in our system with our specific parameters, is getting bigger than the system can handle. As a consequence beyond this specific number of packets the system cannot effectively and efficiently handle the offered load.

In the following Figure (Figure 28) the **Performance over Latency** that is achieved can be easily observed:

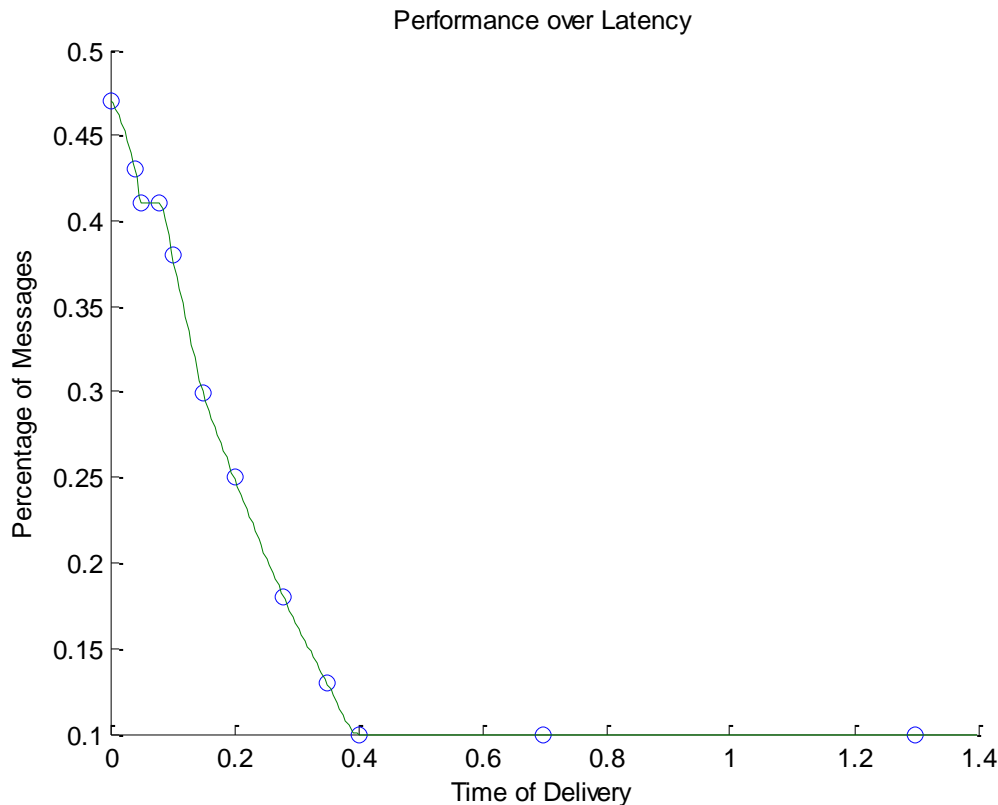


Figure 28: Performance over Latency results obtained after the conduction of the TDMA simulation

As far as performance over latency is concerned and the delivery of the messages, in the TDMA case, it can be easily observed by the above diagram (Figure 28), that the performance that is achieved in TDMA has to exhibit vital and important results. To be more specific, as it is shown in the above diagram (Figure 28) for smaller periods of delivery time the percentage of messages that are delivered successfully reaches high prices with its best, peak, to be observed at a value of 0.47 for a delivery time value around of 0.0 and 0.1. On the other hand, as the delivery time reaches higher levels the percentage of messages that have to be delivered is reduced, which means that the system

shows more stable in comparison with the others that arise from the other algorithms (CDMA, OFDM and CSMA) that are examined below. More specifically, after the delivery time value of 0.4 the percentage of messages remains stable at the value of 0.1 with no further fluctuations beyond this value.

In the following Figure (Figure 29) the **Error Probability** that arises can be easily observed:

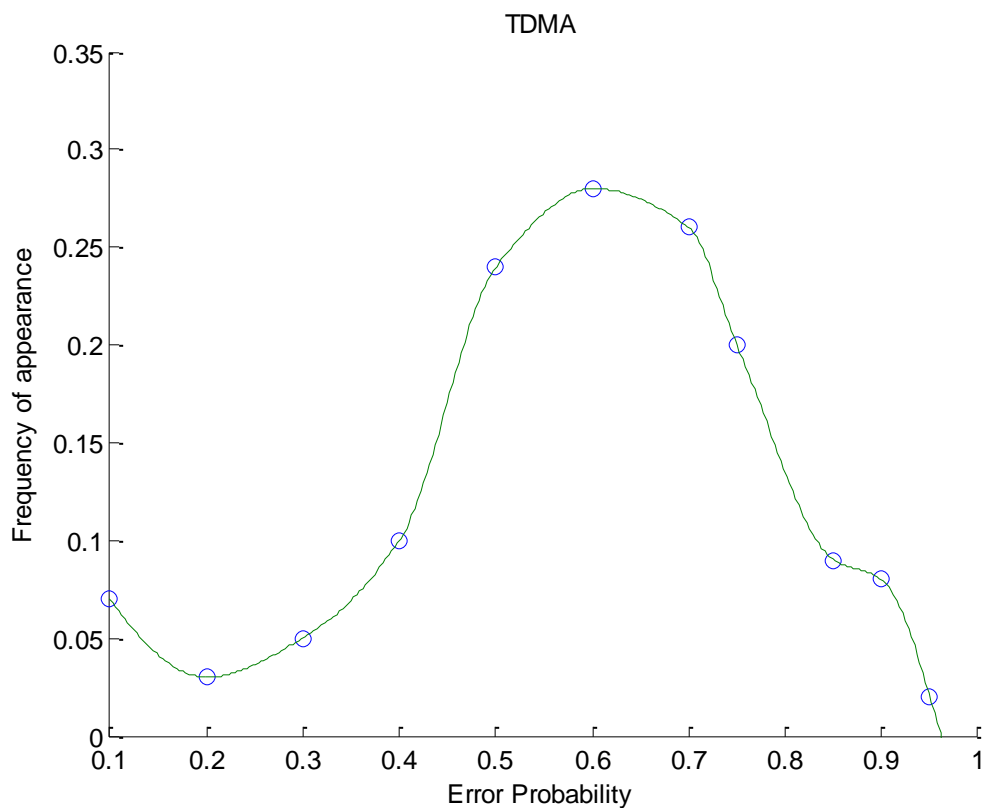


Figure 29: Error Probability results obtained after the conduction of the TDMA simulation

As far as Error Probability is concerned and the frequency of appearance, in the TDMA case, as it can be easily observed by the above diagram (Figure 29), medium performance is achieved. This can be supported by the results that obtained. As it is shown in the

diagram of Figure 29 the TDMA case exhibits various fluctuations. For low error probability values variations and fluctuations are observed in the system. After the value of approximately 0.42 of error probability, the frequencies are increased and the system reaches its highest value at high error probability values. After this peak, around 0.62 of error probability, lower frequencies on the right of the peak are observed which correspond to higher error probabilities. From all these results that are obtained by the conduction of the simulation of error probability for the TDMA case it can be stated that medium performance is achieved as far as error probability is concerned. By the various fluctuations of the curve of the above diagram (Figure 29), it can be safely supported that the system does not reach high stability levels in the TDMA case due to these fluctuations that are observed. The system is not stable due to the fact that the curve is fluctuated at various values of error probability and at various frequencies all the time. As it was previously mentioned, the system reaches its highest value at high error probability values. Furthermore, it remains in the high error probability area for relatively high frequency values and this is a very significant factor for the stability of the system because the errors occur more often and this is an undesirable condition.

3.3 Simulation of Code Division Multiple Access CDMA

For the needs of this exercise, we have added several variables. First of all, we introduced a new parameter in the parameter file named omnetpp.ini. The parameter SNR, produces a value that comes from a normal distribution every time this parameter is read in the c++ Server.cc source file. In this way, even if we have selected 20 hosts, we speculate that their distances from the Server and therefore their corresponding snrs are random numbers. Such an action contributes to the randomness of the whole experiment. These parameters can be observed in the following Figure (Figure 30):

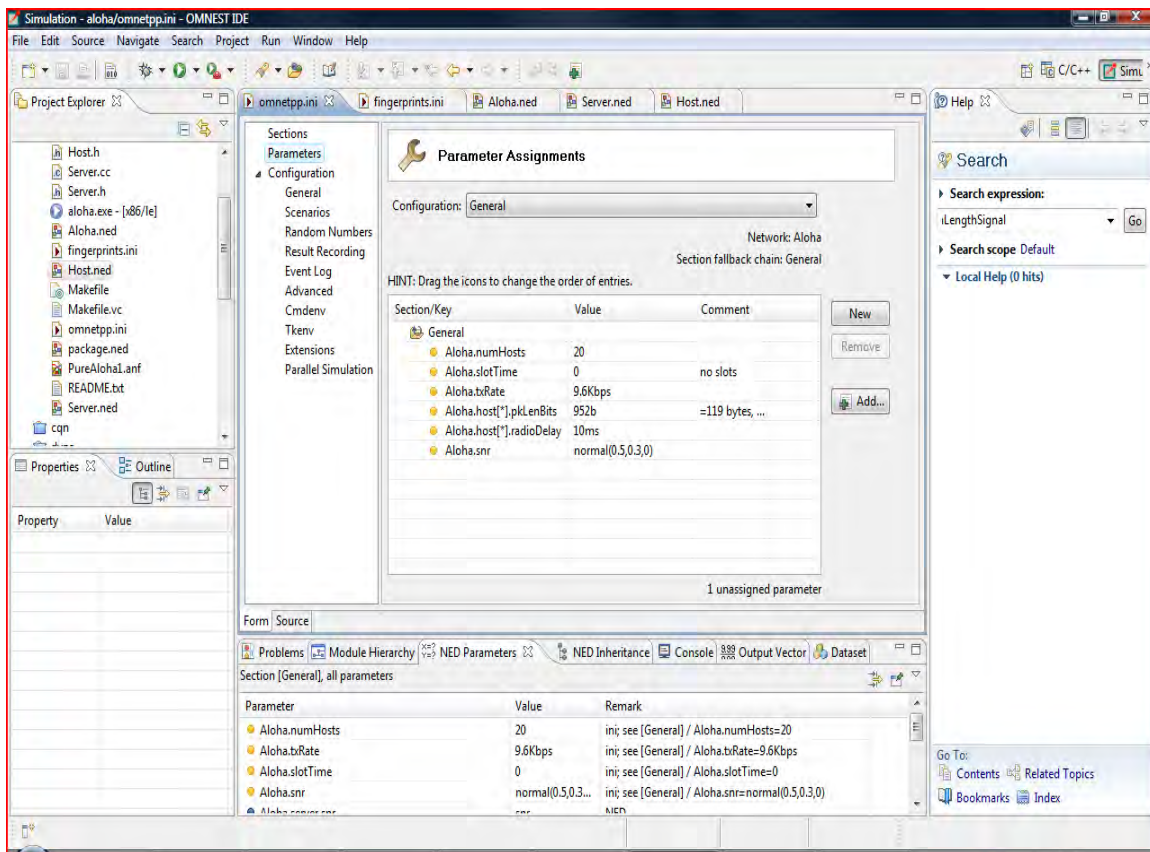


Figure 30: Omnetpp.ini Parameter file

A big variety of research and experiment efforts has already taken place in the area of the exact calculation of bit error rates that occur in Code Division Multiple Access CDMA systems. However the absolute and accurate calculation of bit error rates in case of Reconfiguration between the different congestion avoidance and control algorithms for 4G handsets

CDMA remains still difficult. As a consequence the main goal is to try to calculate these errors approximately [55, 56, 57].

A very significant approximation towards this field is considering the multiple access interference as Gaussian noise and as a consequence this approximation is known as Standard Gaussian Approximation (SGA). The most significant aspect of this approximation is that its accuracy relies on the number of users that access the channel. The larger is the number of users that access the channel simultaneously, the more accurate is the approximation. If the number of users that access the channel is not very large, then the approximation tends to be optimistic [58].

The key point is that Gaussian Approximation may be further improved. This is known as Improved Gaussian Approximation (IGA). Such an action leads not only to vital computational time needed but also to more difficult and complicated ways in order to achieve Improved Gaussian Approximation (IGA) [59].

Following the introduction of the SNR levels of each host, we use the Packet Error probability P_e as well as the Packet Success Probability P_s and the **Throughput** of the messages. Let us note that if the number of collided packets is k , the spreading gain factor in CDMA is N , the SNR lever of the sender of the message is E_b/N_0 and Q is the error function, then the respective probabilities are defined as [46, 60]:

$$P_e \approx \frac{2}{3} Q \left(\sqrt{\frac{N^2}{2 * \mu_\psi + \frac{N_0}{2Eb} * N^2}} \right) + \frac{1}{6} Q \left(\sqrt{\frac{N^2}{2 * (\mu_\psi + \sqrt{3} \sigma_\psi) + \frac{N_0}{2Eb} * N^2}} \right) + \frac{1}{6} Q \left(\sqrt{\frac{N^2}{2 * (\mu_\psi - \sqrt{3} \sigma_\psi) + \frac{N_0}{2Eb} * N^2}} \right) \quad (1)$$

And

$$P_S(L) = \sum_{k=0}^{\infty} \sum_{k_1=0}^{\infty} P_S(k, L, k_1)(1 - P_e(k)) \quad (2)$$

Where, the possibility of **k colliding packets** arriving to the originally received is [60]

$$P(k) = \frac{\lambda^k}{k!} \exp(-\lambda) \quad (3)$$

And λ is the mean value of the exponential distribution according to which the messages in the network are usually produced [60].

Finally the Q error function is as it is presented bellow [60]:

$$\mu_\psi = \frac{N}{6}k, \quad \sigma_\psi^2 = \frac{k}{4} \left[\frac{23N^2}{360} + N \left(\frac{1}{20} + \frac{k-1}{36} \right) - \frac{1}{20} - \frac{k-1}{36} \right]$$

$$P_i = 1, \quad i = 0, 1, \dots, k \quad (4)$$

$$Q(v) = \frac{1}{\sqrt{2\pi}} \int_v^{\infty} \exp\left(-\frac{x^2}{2}\right) \quad (5)$$

The code that is used in each one of the files that exist in the CDMA Aloha example separately is presented in the appendix of this project.

In the following Figure (Figure 31) is illustrated the running of CDMA simulation.

```

C:\OMNEST_EVAL\samples\aloha_dip\aloha_dip.exe
OMNEST Discrete Event Simulation (C) 1992-2010 Andras Varga, OpenSim Ltd.
Version: 4.1, build: 100611-4b63c38, edition: Commercial Edition
See the license for distribution terms and warranty disclaimer
Setting up Tkenv...
Loading NED files from . : 4
Loading images from .\images: *: 0
Loading images from .\bitmaps: *: 0
Loading images from C:\OMNEST_EVAL\images: *: 0 abstract/*: 90 background/*: 4
block/*: 320 device/*: 195 maps/*: 9 misc/*: 70 msg/*: 55 old/*: 111 sta
tus/*: 28
Plugin path: ./plugins

```

Figure 31: Running the CDMA Simulation

In the following Figure (Figure 32) an example of CDMA executable, with the pointer bu easily observed, is illustrated. This pointer is used for collision detection and, as it has been already mentioned, increases only if a packet makes it through without colliding with another one during the transmission time, therefore **the throughput is calculated as the value of this pointer.**

```

OMNeT++/Tkenv - Aloha
File Edit Simulate Trace Inspect View Options Help
Run #0: Aloha      Event #10      Msgs created: 24      T=0.833155932485      Next: n/a
Msgs scheduled: 21      Simsec/sec: n/a      Msgs present: 22      Ev/sec: n/a
Ev/sec: n/a

+1e-5      +1e-4      +0.001      +0.01      +0.1      +1.0      sec
pk-12-#0
send/endTx... send/endTx... send/endTx... send/endTx... send/endTx...
send/endTx... send/endTx... send/endTx... send/endTx... send/endTx...

Aloha (Aloha) [id=1]
  Initializing module Aloha.server, stage 0
  Initializing module Aloha.host[0], stage 0
  Initializing module Aloha.host[1], stage 0
  Initializing module Aloha.host[2], stage 0
  Initializing module Aloha.host[3], stage 0
  Initializing module Aloha.host[4], stage 0
  Initializing module Aloha.host[5], stage 0
  Initializing module Aloha.host[6], stage 0
  Initializing module Aloha.host[7], stage 0
  Initializing module Aloha.host[8], stage 0
  Initializing module Aloha.host[9], stage 0
  Initializing module Aloha.host[10], stage 0
  Initializing module Aloha.host[11], stage 0
  Initializing module Aloha.host[12], stage 0
  Initializing module Aloha.host[13], stage 0
  Initializing module Aloha.host[14], stage 0
  Initializing module Aloha.host[15], stage 0
  Initializing module Aloha.host[16], stage 0
  Initializing module Aloha.host[17], stage 0
  Initializing module Aloha.host[18], stage 0
  Initializing module Aloha.host[19], stage 0
  Event #1 T=0.040951176798 Aloha.host[10] [Host, id=13], on selfmsg `send/endTx' (cMessage, id=66)
    generating packet pk-13-#0
  Event #2 T=0.050951176798 Aloha.server (Server, id=2), on `pk-13-#0' (cPacket, id=76)
    started receiving
  Event #3 T=0.140017043464 Aloha.host[10] [Host, id=13], on selfmsg `send/endTx' (cMessage, id=66)
  Event #4 T=0.150017043464 Aloha.server (Server, id=2), on selfmsg `end/reception' (cMessage, id=55)
    reception finished
    ba0.0282327
  Event #5 T=0.636712281683 Aloha.host[5] [Host, id=8], on selfmsg `send/endTx' (cMessage, id=61)
    generating packet pk-8-#0
  Event #6 T=0.646712281683 Aloha.server (Server, id=2), on `pk-8-#0' (cPacket, id=77)
    started receiving
  Event #7 T=0.735978948349 Aloha.host[5] [Host, id=8], on selfmsg `send/endTx' (cMessage, id=61)
  Event #8 T=0.745978948349 Aloha.server (Server, id=2), on selfmsg `end/reception' (cMessage, id=55)
    reception finished
    ba0.0473754
  Event #9 T=0.823155932485 Aloha.host[9] [Host, id=12], on selfmsg `send/endTx' (cMessage, id=65)
    generating packet pk-12-#0
    Calling finish() methods of modules
    duration: 0.823155932485
    bu: 7
    ba: 0.473754
    Throughput: 0.541341

```

Figure 32: Example of a Run of CDMA executable

Consequently, according to the channel Throughput which expresses the possibility of the successful transfer of the whole message, we calculated the success with respect to the sum of the messages being transmitted in the CDMA case. The experiment was conducted a number of times and we gathered some vital results as shown in the diagrams that are illustrated in the Figures bellow.

In the following Figure (Figure 33) the **Throughput** that is derived over the offered load is clearly illustrated:

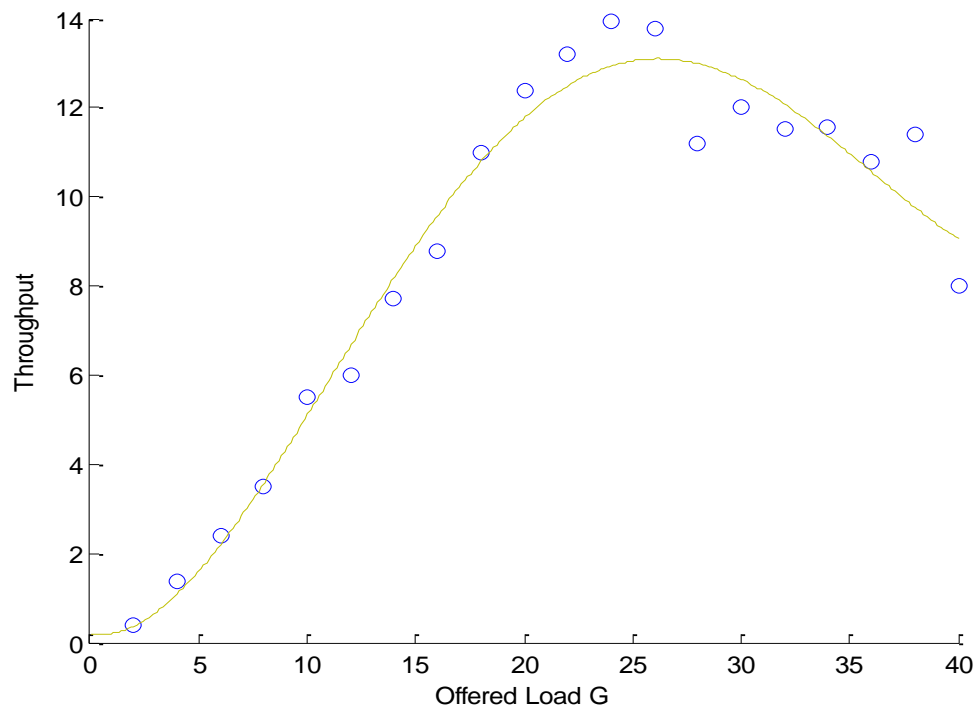


Figure 33: Throughput results obtained after the conduction of the CDMA simulation

As it can be easily observed by the Figure 33 above, the channel traffic is fairly controlled and that is because our results are restricted to a small range of high possibility throughputs, which is around 50%. This means that we achieved better results around the value of 51, where we had our maximum, for a wider range. Additionally, either for very

little traffic in the channel or for quite a lot the results tend to be worse and the multiplexing less effective. However, we note that the curve is fairly stable after the peak of almost 25 packets, because the throughput has been improved compared to TDMA.

In the following Figure (Figure 34) the **Performance over Latency** that is achieved can be easily observed:

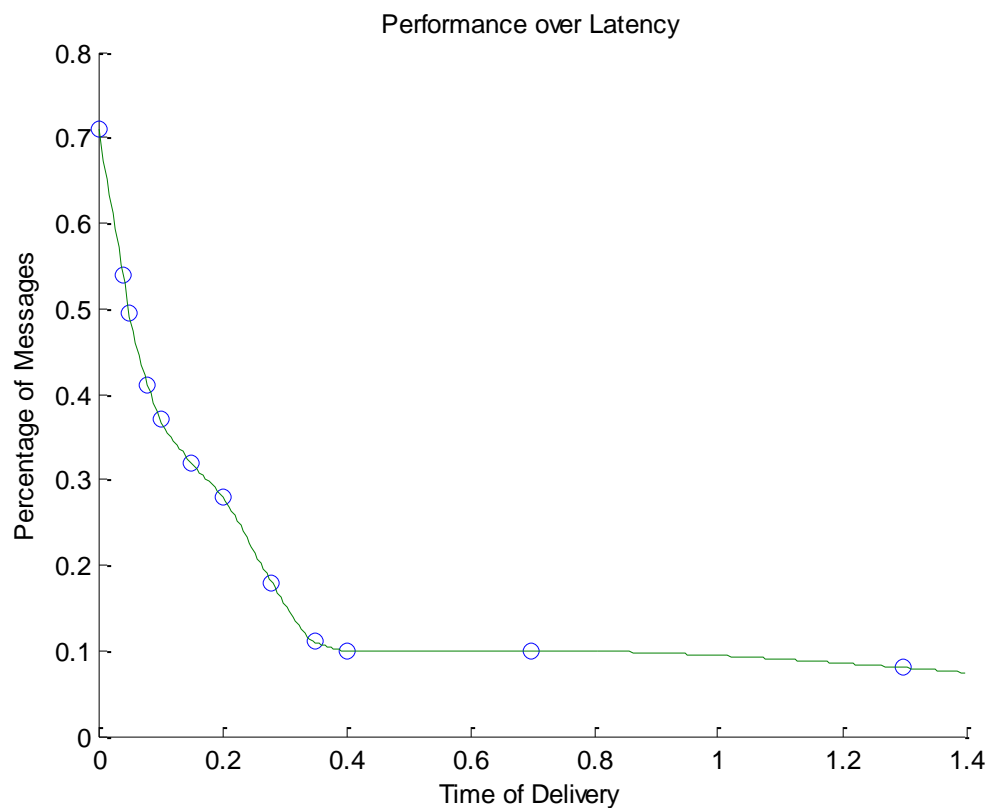


Figure 34: Performance over Latency results obtained after the conduction of the CDMA simulation

As far as performance over latency is concerned and the delivery of the messages, in the CDMA case, it can be easily observed by the above diagram (Figure 34), that the performance that is achieved in CDMA has to exhibit significant and important results.

To be more specific, as it is shown in the above diagram (Figure 34) for smaller periods of delivery time the percentage of messages that are delivered successfully reaches high prices with its best, peak, to be observed at a value of 0.7 for a delivery time value around of 0.0 and 0.015. There are some fluctuations until the delivery time reaches the value around 0.37 and 0.4. After the value of 0.4 that corresponds to the delivery time the percentage of messages is reduced and fluctuates around the value of 0.1. The systems exhibits some stability characteristics but in the Performance over Latency case the TDMA outperforms better than CDMA due to the fact that the messages that have to be delivered tend to be zero whereas in the CDMA case the percentage may be almost stable but always remain messages for being delivered.

In the following Figure (Figure 35) the **Error Probability** that arises can be easily observed:

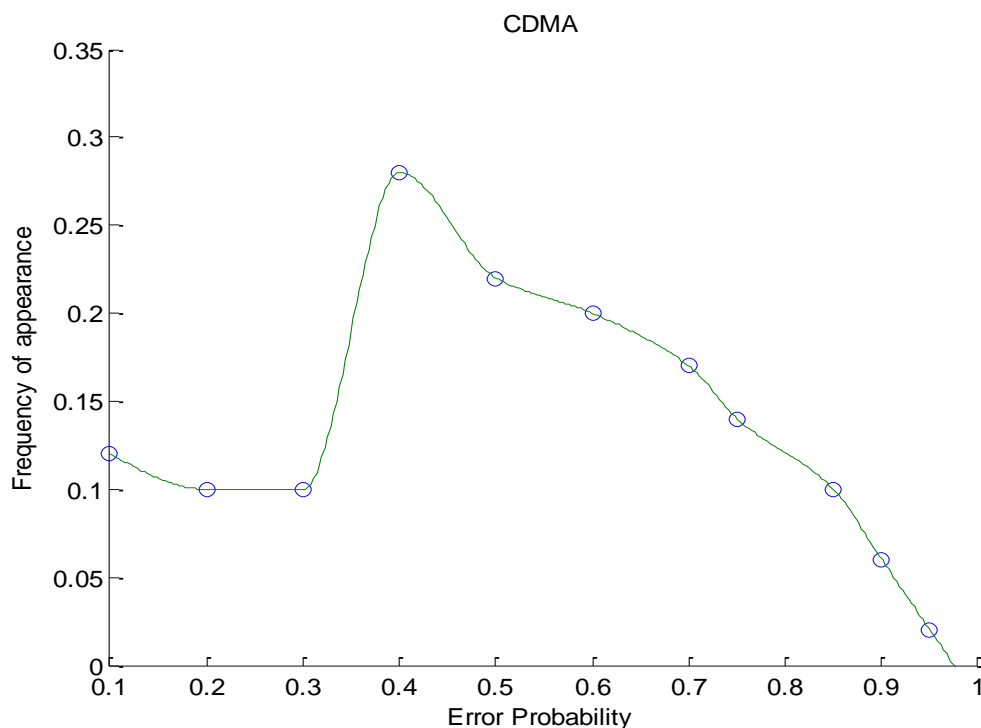


Figure 35: Error Probability results obtained after the conduction of the CDMA simulation

As far as Error Probability is concerned and the frequency of appearance, in the CDMA case, as it can be easily observed by the above diagram (Figure 35), a very good performance is achieved. This can be supported by the results that obtained. As it is shown in the diagram of Figure 35 the CDMA case exhibits the least fluctuations compared to those that are observed in the other techniques. As it can be easily observed, in CDMA we have the best performance, with relatively stable values in the first section. After the value of around 0.3 the frequency of appearance is increased and reaches its maximum value approximately at 0.28 which corresponds to an approximate value of 0.4 in error probability. The significant that has to be mentioned in this case is that the maximum frequency of appearance corresponds to a relatively low error probability. This means that errors do not take place so often and this is the best guarantee for the stability of the system. Furthermore, another important attribute that conducted by this simulation is that high error probabilities take place in low enough frequencies, as it is clearly observed by the above diagram. On the right of the maximum value of the peak it is clearly observed that the higher error probabilities correspond to lower frequencies and this is the most important advantage of CDMA technique. The key for the stability of a system is the errors to occur as seldom as possible. The results that were obtained by this simulation confirm this aspect. As a consequence, it can be safely supported that the system in the CDMA case outperforms better than this in the TDMA case values. In TDMA case it remains in the high error probability area for relatively high frequency values whereas in the CDMA case higher error probabilities correspond to lower frequencies. As it was previously mentioned this is a very significant factor for the stability of the system because errors that occur more often lead to undesirable situations and conditions.

3.4 Simulation of Orthogonal Frequency Division Multiplexing OFDM

Among all the simulations that are carried out using orthogonal in frequency carriers, we chose the constellation of 16 QAM (**Quadrature Amplitude Modulation**) as the ideal for the needs of our simulation. In the following Figure (Figure 36) the 16 QAM is illustrated.

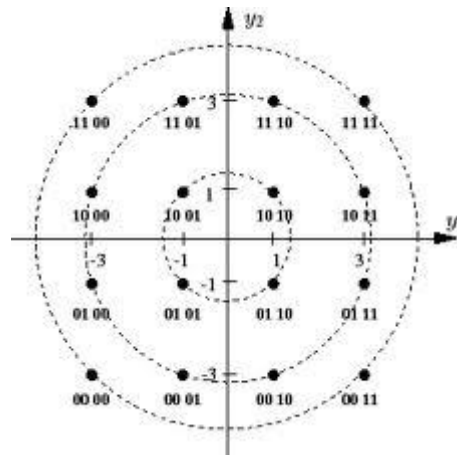


Figure 36: 16 QAM

Quadrature amplitude modulation (QAM) is both an analog and a digital modulation scheme. It is mainly based on the conveyance of two analog message signals, or two digital bit streams. This is achieved by modulating the amplitudes of two carrier waves, using two different schemes: the amplitude-shift keying (ASK) digital modulation scheme or amplitude modulation (AM) analog modulation scheme [61].

The two carrier waves, usually sinusoids, are out of phase with each other by 90° and are thus called quadrature carriers or quadrature components. As it can be easily noticed the name of the scheme derives exactly from that. The waveform, which results by summing the two waves, is a combination of both phase-shift keying (PSK) and amplitude-shift keying (ASK). Furthermore it can be expressed as the combination of phase modulation

(PM) and amplitude modulation (AM). In the digital QAM case, a number of at least two phases and at least two amplitudes are used. QAM is extensively used as a modulation scheme that is dedicated for digital telecommunication systems [61]. The main characteristics of Digital QAM and Analog QAM are mentioned and explained below.

- ❖ **Digital QAM:** Like in the majority of modulation schemes, QAM conveys data by changing some aspect of a carrier signal or wave, which corresponds usually to a sinusoid, in response to a data signal. In the case of QAM, the representation of the data signal is achieved by the modulation or keying of the amplitude of two waves, which are 90° degrees out-of-phase with each other (in quadrature). The Amplitude modulation of two carriers in quadrature can be equivalently viewed as modulating a single carrier in both amplitude and phase modulating way [61].

Phase modulation (analog PM) and phase-shift keying (digital PSK) can be considered as a special case of QAM. In this case the magnitude of the modulating signal is a constant and the phase is the only one that changes (varies). This aspect can also be extended to Frequency Modulation (FM) and Frequency-Shift Keying (FSK), where in these cases it can be considered as a special case of phase modulation.

- ❖ **Analog QAM:** When transmitting two signals by modulating them with QAM, the transmitted signal will be of the form:

$$s(t) = I(t) \cos(2\pi f_0 t) + Q(t) \sin(2\pi f_0 t), \quad (6)$$

where $I(t)$ and $Q(t)$ are the modulating signals and f_0 is the carrier frequency.

At the receiver, these two modulating signals can be demodulated using a coherent demodulator. In this case the receiver multiplies the received signal separately with both a cosine and sine signal. As a consequence the received estimates of $I(t)$ and $Q(t)$ are respectively produced. The detection of each one of the modulating signals separately in an independent way can be achieved by the orthogonal property of the carrier signals [61].

In the ideal case $I(t)$ is demodulated by multiplying the transmitted signal with a cosine signal:

$$\begin{aligned} r_i(t) &= s(t) \cos(2\pi f_0 t) \\ &= I(t) \cos(2\pi f_0 t) \cos(2\pi f_0 t) + Q(t) \sin(2\pi f_0 t) \cos(2\pi f_0 t) \end{aligned} \quad (7)$$

Using standard trigonometric identities, we can write it as:

$$\begin{aligned} r_i(t) &= \frac{1}{2} I(t) [1 + \cos(4\pi f_0 t)] + \frac{1}{2} Q(t) \sin(4\pi f_0 t) \\ &= \frac{1}{2} I(t) + \frac{1}{2} [I(t) \cos(4\pi f_0 t) + Q(t) \sin(4\pi f_0 t)] \end{aligned} \quad (8)$$

In the following Figure (Figure 37) a measured PAL color bar signal on a vector analyzer screen is illustrated:

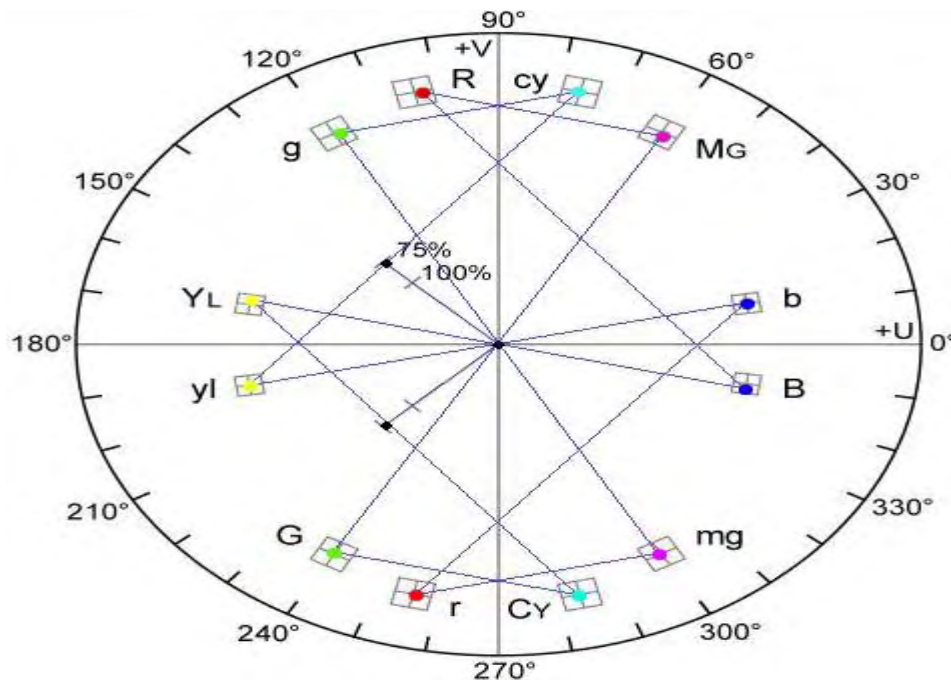


Figure 34: Analog QAM: measured PAL colour bar signal on a vector analyser screen

As it was previously mentioned, 16 QAM constellation was used in our case. The probability of the Throughput was calculated in the same way. However, the Probability of packet error **Pe** has now changed. More specifically the packet error probability is given by the following form [59, 60]:

$$Pe = 2 \cdot \left(1 - \frac{1}{\sqrt{M}}\right) \cdot \text{erfc} \sqrt{\frac{3}{2 \cdot (M - 1)} \cdot SNR} \quad (9)$$

Where, **erfc** corresponds to the complementary error function and is given by the following form [59]:

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-u^2} du \quad (10)$$

The code that is used in each one of the files that exist in the OFDM Aloha example separately is presented in the appendix of this project.

In the following Figure (Figure 38) an example of a run of the OFDM Simulation is illustrated. After a packet is received or after a collision is conducted, the Throughput is shown on screen, so as to make it easier to note down the statistics.

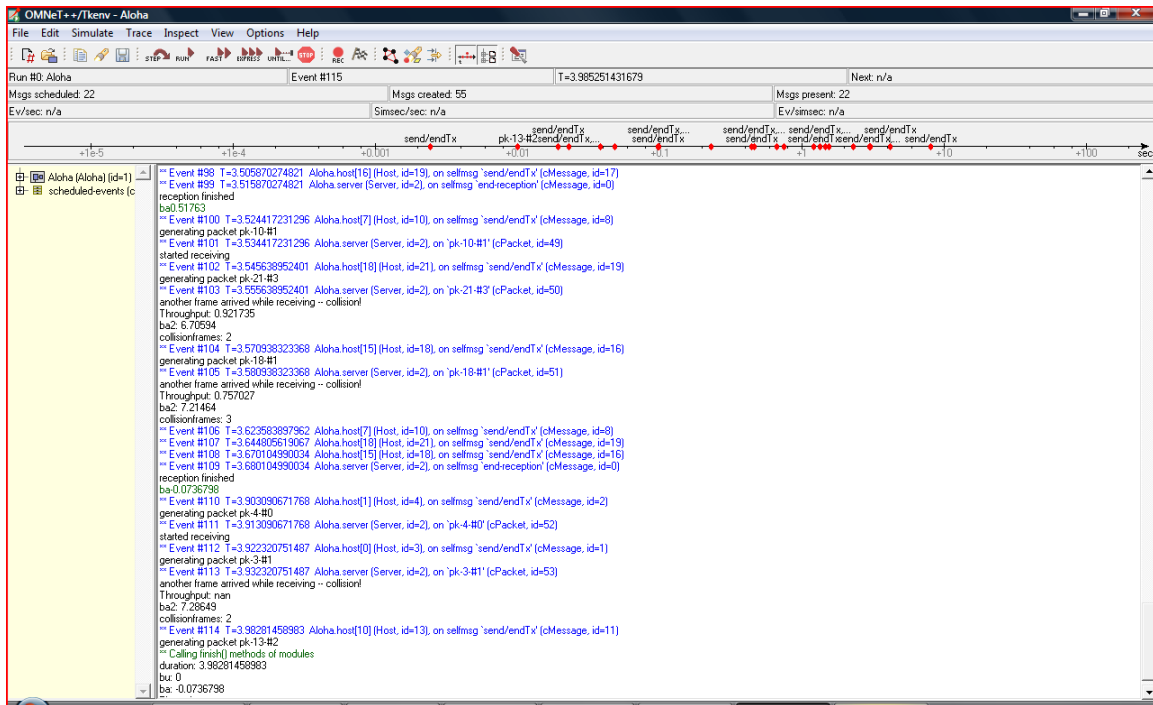


Figure 38: Example of a run of an OFDM executable

The experiment was conducted a number of times and we gathered some vital results as shown in the diagrams that are illustrated in the Figures below.

In the following Figure (Figure 39) the **Throughput** that is derived over the offered load is clearly illustrated:

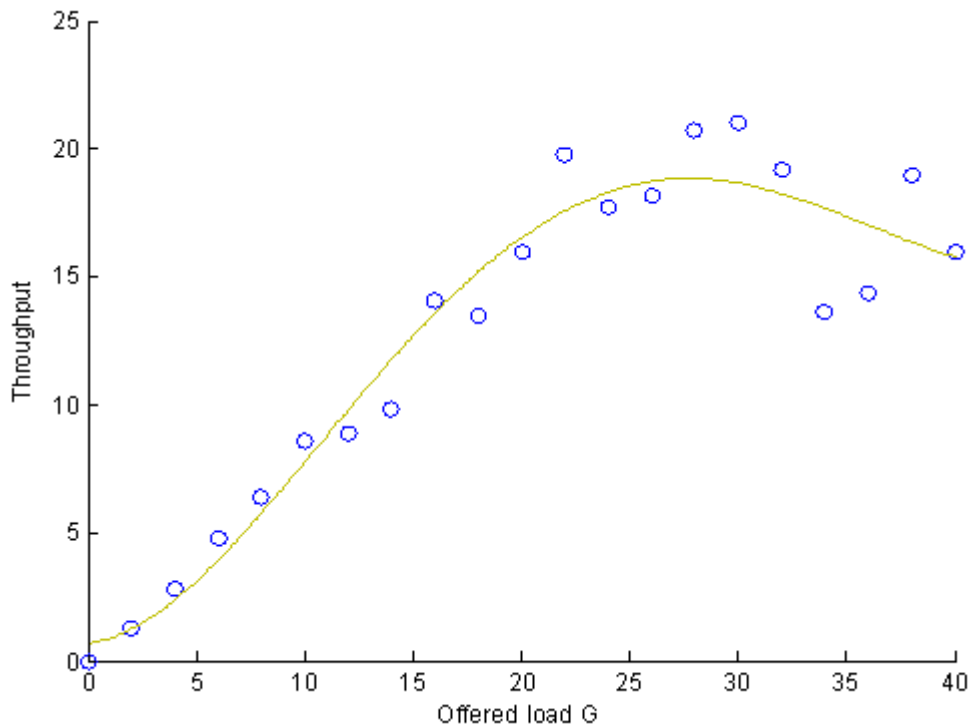


Figure 39: Throughput results obtained after the conduction of the OFDM simulation

In this case, as it can be easily observed by the above diagram, we have to notice that the curve seems more stable as the load increases, which means that there is a slight improvement compared to that of CDMA. It can be easily observed that similarly to TDMA and CDMA case for very little traffic in the channel the throughput fluctuates in low levels. The point that has to be mentioned is that the curve in the case of OFDM reaches a bit higher levels compared to those of TDMA and CDMA. Additionally, when the traffic in the channel is increased then the curve reaches higher levels and as we move beyond 25 packets, the throughput is decreased because the load obviously, in our system with our specific parameters, is getting bigger than the system can handle. But the main point that has to be mentioned is that the curve seems more stable as the load increases and does not exhibit the rapid decrease on the right side like in the previous (TDMA and CDMA) multiplexing techniques. This means that there is a slight improvement compared to that of CDMA.

In the following Figure (Figure 40) the **Performance over Latency** that is achieved can be easily observed:

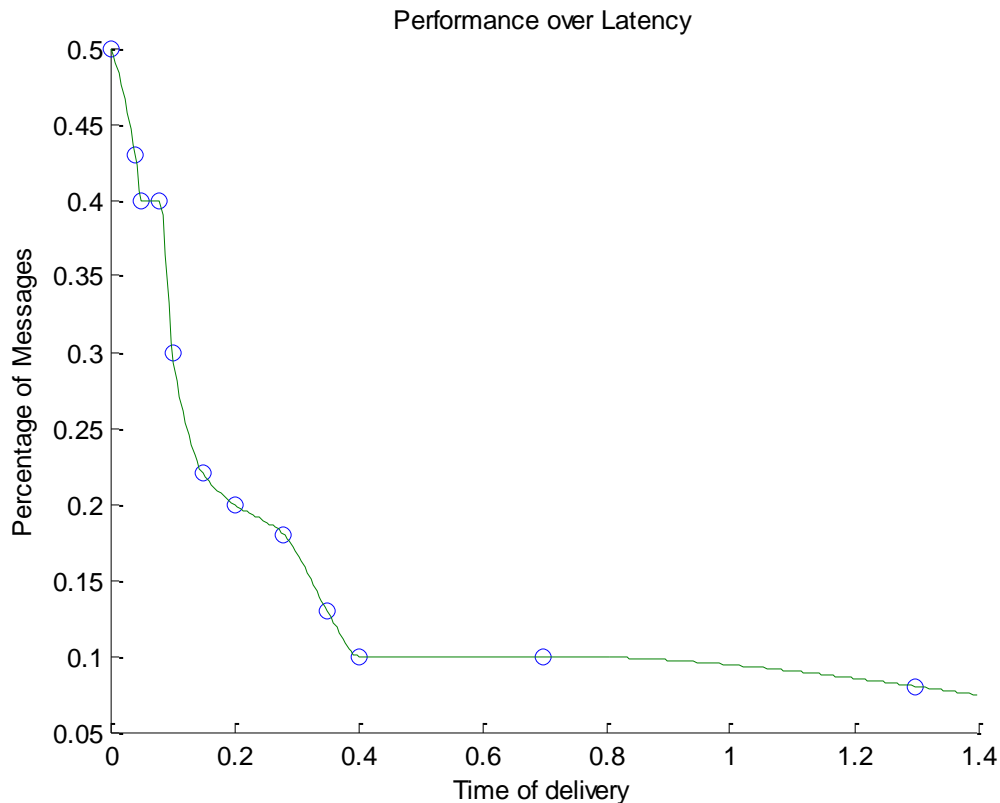


Figure 40: Performance over Latency results obtained after the conduction of the OFDM simulation

As far as performance over latency is concerned and the delivery of the messages, in the OFDM case, it can be easily observed by the above diagram (Figure 40), that the performance that is achieved in OFDM has to exhibit vital and interesting results. To be more specific, as it is shown in the above diagram (Figure 40) for smaller periods of delivery time the percentage of messages that are delivered successfully reaches high prices with its best, peak, to be observed at a value of 0.5 for a delivery time value around of 0.1. On the other hand, as the delivery time reaches higher levels, the percentage of messages that are delivered successfully is reduced and at a value of 0.4 for delivery time

the percentage corresponds to a value of about 0.1. After this value the system tends to be more stable but, as in the CDMA case, the percentage may be almost stable but always remain messages for being delivered. There are similarities in the Performance over Latency scenario in both CDMA and OFDM techniques which indicate the fact that they operate in a similar way in this case, but compared to TDMA it can be safely stated that it outperforms better not only than CDMA but also than OFDM.

In the following Figure (Figure 41) the **Error Probability** that arises can be easily observed:

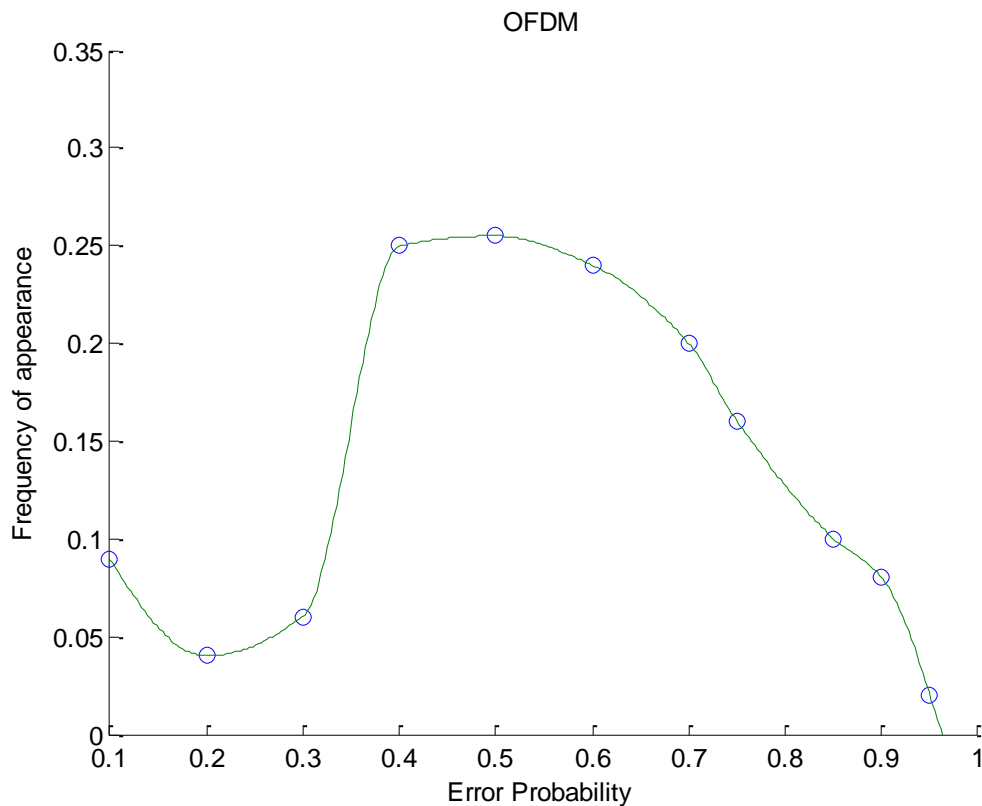
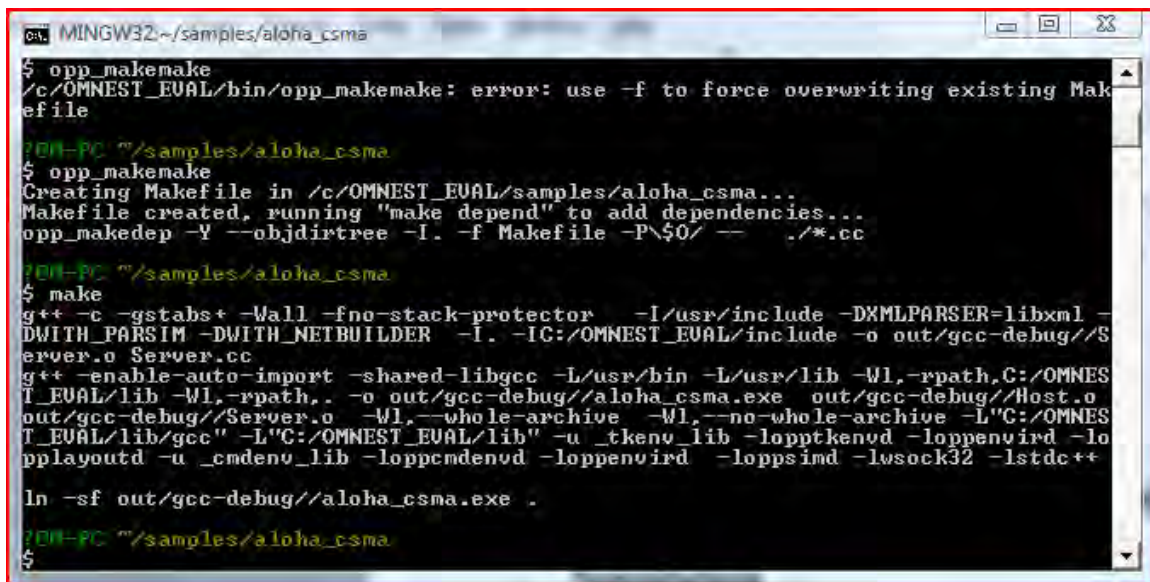


Figure 41: Error Probability results obtained after the conduction of the OFDM simulation

As far as Error Probability is concerned and the frequency of appearance, in the OFDM case, as it can be easily observed by the above diagram (Figure 41), medium performance is achieved. This can be supported by the results that obtained. As it is shown in the diagram of Figure 41 the OFDM case exhibits a variety of fluctuations. For low error probability values variations and fluctuations are observed in the system. After the value of approximately 0.32 of error probability, the frequencies are increased and the system reaches its highest value at high error probability values. After this peak, around 0.5 of error probability, lower frequencies on the right of the peak are observed which correspond to higher error probabilities. From all these results that are obtained by the conduction of the simulation of error probability for the OFDM case it can be supported that medium performance is achieved as far as error probability is concerned. By the various fluctuations of the curve of the above diagram (Figure 41), it can be safely stated that the system does not reach high stability levels in the OFDM case due to these fluctuations that are observed. The system is not stable due to the fact that the curve is fluctuated at various values of error probability and at various frequencies all the time. As it was previously mentioned, the system reaches its highest values at relatively high error probability values. Additionally, it remains in the high error probability area for relatively high frequency values and this is a very significant factor for the stability of the system because the errors occur more often and this is an undesirable condition. It has to be mentioned that medium performance is noticed in both OFDM and TDMA but OFDM seems to be slightly better, because the maximum value of frequencies is located on slightly higher values of error probability in TDMA.

3.5 Simulation of Carrier Sense Multiple Access CSMA

The Simulation of a simple network using Carrier Sense Multiple Access is achieved through the basis model of Aloha network of OMNEST EVAL or OMNET++. In this case the network is supplemented with the addition of CSMA in a simple way. In the following Figure (Figure 42) the running of CSMA Aloha simulation through executable is illustrated.



```

MINGW32 ~/samples/aloha_csma
$ opp_makemake
/c/OMNEST_EVAL/bin/opp_makemake: error: use -f to force overwriting existing Makefile

?D00-PC ~/samples/aloha_csma
$ opp_makemake
Creating Makefile in /c/OMNEST_EVAL/samples/aloha_csma...
Makefile created, running "make depend" to add dependencies...
opp_makedep -Y --objdirtree -I. -f Makefile -P\$/ -- /*.cc

?D00-PC ~/samples/aloha_csma
$ make
g++ -c -gstabs+ -Wall -fno-stack-protector -I/usr/include -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_NETBUILDER -I. -IC:/OMNEST_EVAL/include -o out/gcc-debug/Server.o Server.cc
g++ -enable-auto-import -shared-libgcc -L/usr/bin -L/usr/lib -Wl,-rpath,C:/OMNEST_EVAL/lib -Wl,-rpath,. -o out/gcc-debug/aloha_csma.exe out/gcc-debug/Host.o out/gcc-debug/Server.o -Wl,-whole-archive -Wl,-no-whole-archive -L"C:/OMNEST_EVAL/lib/gcc" -L"C:/OMNEST_EVAL/lib" -u _tkenv_lib -lopptkenvd -loppenvird -lopplayoutd -u _cmdenv_lib -loppcmdenvd -loppenvird -loppsind -lwssock32 -lstdc++
ln -sf out/gcc-debug/aloha_csma.exe .

?D00-PC ~/samples/aloha_csma
$
  
```

Figure 42: Running CSMA Aloha Simulation through executable

In the following Figure (Figure 43) the files that exist in the File of CSMA Aloha example can be easily observed:

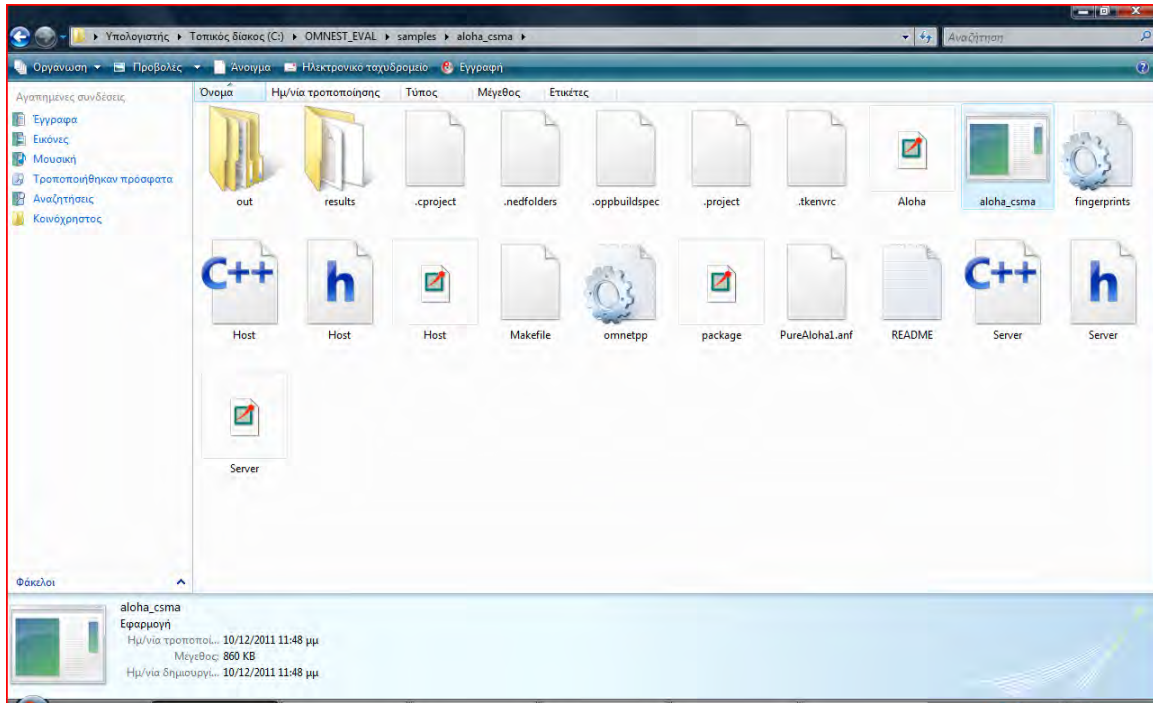


Figure 43: Files in the file of CSMA Aloha example

The code that is used in each one of the files that exist in the CSMA Aloha example separately is presented in the appendix of this project.

In the following Figure (Figure 44) an example of a run of the CSMA Simulation is illustrated.

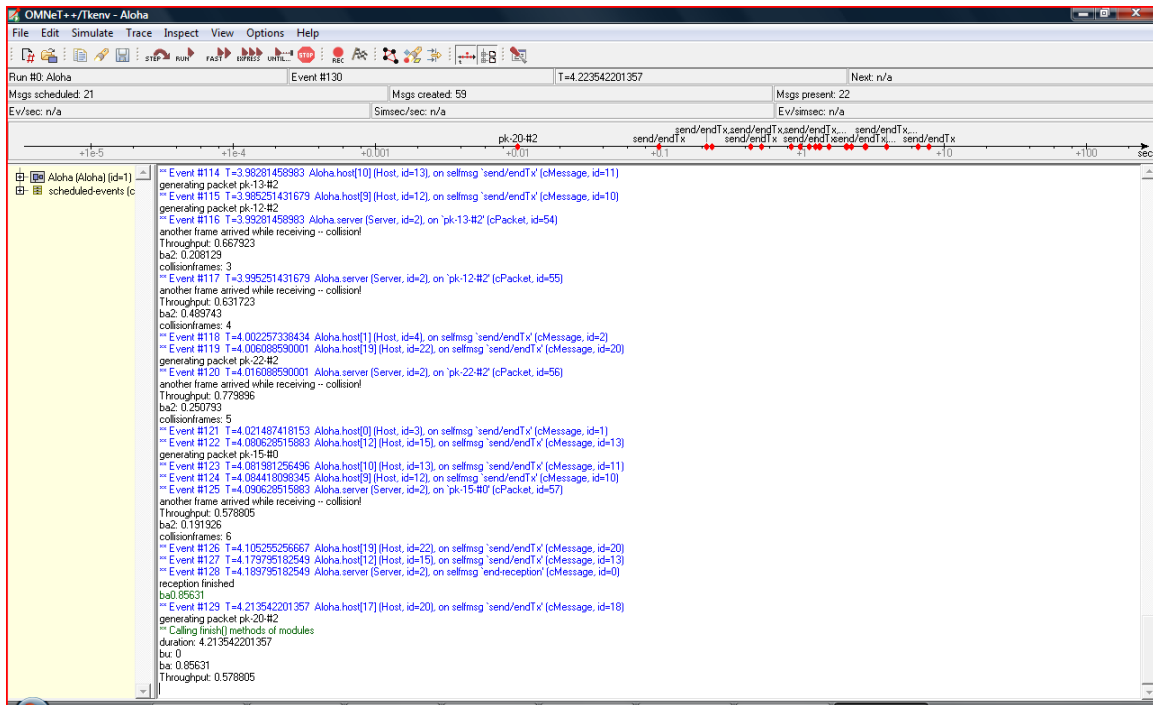


Figure 44: Example of a run of a CSMA executable

Consequently, according to the channel Throughput which expresses the possibility of the successful transfer of the whole message, we calculated the success with respect to the sum of the messages being transmitted in the CSMA case.

The experiment was conducted a number of times and we gathered some vital results as shown in the diagrams that are illustrated in the Figures below.

In the following Figure (Figure 45) the **Throughput** that is derived over the offered load is clearly illustrated:

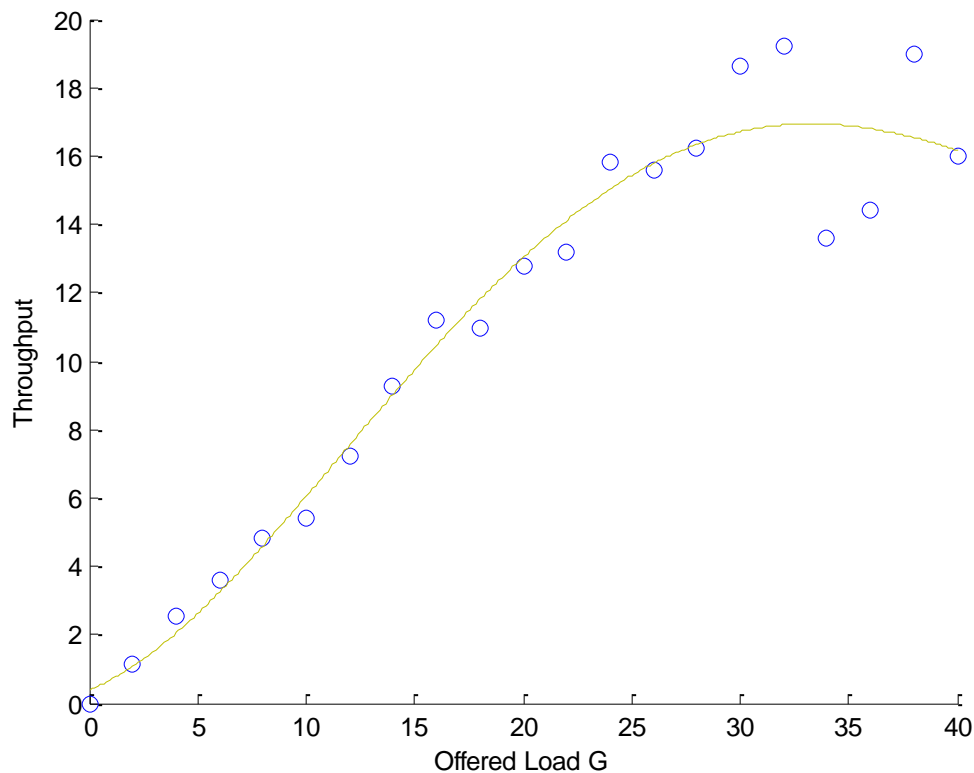


Figure 45: Throughput results obtained after the conduction of the CSMA simulation

As it can be easily observed by the above diagram (Figure 45), the peak of the curve in the CSMA case is transferred to a bigger load, which corresponds to an offered load of 33 packets. This price is the biggest compared to all the algorithms and multiplexing techniques (TDMA, CDMA and OFDM) that were previously mentioned. This means that our system remains stable for bigger amount of offered load and as a consequence the throughput reaches higher levels compared to those of the previous algorithms. Furthermore, it has to be mentioned that after the peak of the curve, it remains almost stable and as a consequence it means that even bigger load can be serviced and in wider range. In our example, with our parameters, we would probably choose the CSMA multiplexing technique as the most effective one among all the aforementioned (TDMA, CDMA, OFDM).

In the following Figure (Figure 46) the **Performance over Latency** that is achieved can be easily observed:

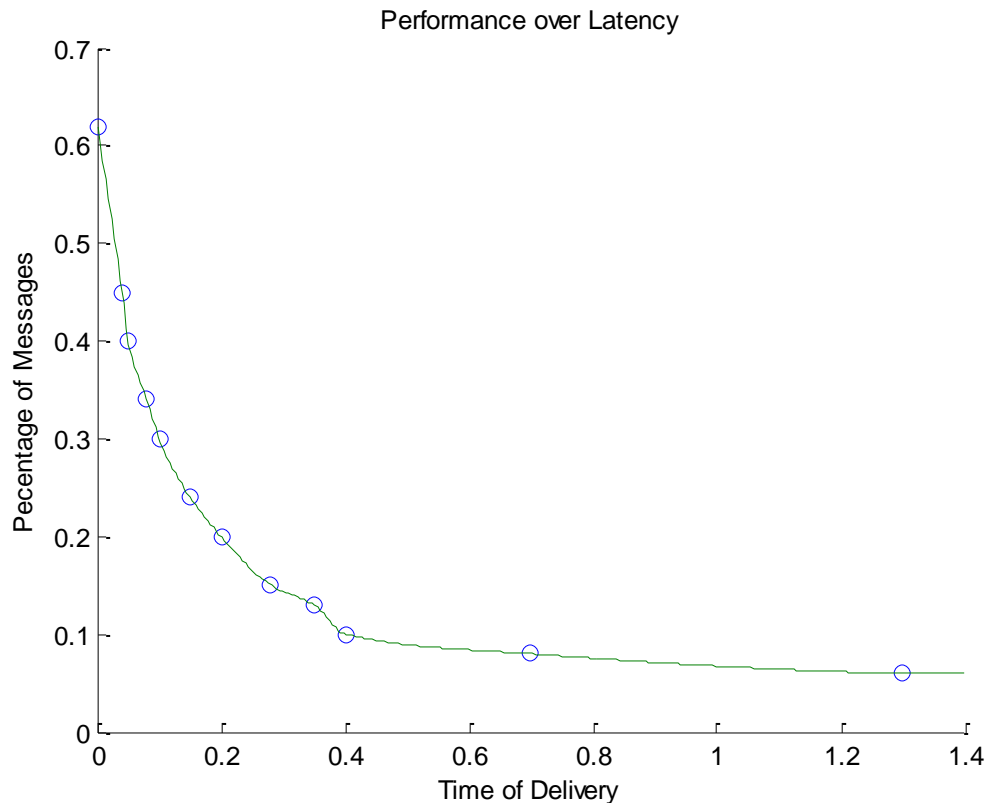


Figure 46: Performance over Latency results obtained after the conduction of the CSMA simulation

As far as Performance over Latency is concerned and the delivery of the messages, in the CSMA case, it can be easily observed by the above diagram (Figure 46), that the performance that is achieved in OFDM has also to exhibit significant and interesting results as far as the operation of CSMA technique. To be more specific, as it is shown in the above diagram (Figure 46) for smaller periods of delivery time the percentage of messages that are delivered successfully reaches high prices with its best, peak, to be observed at a value of 0.62 for a delivery time value around of 0.015 to 0.1. On the other hand, as the delivery time reaches higher levels, the percentage of messages that are

delivered successfully is reduced and at a value of 0.4 for delivery time the percentage corresponds to a value of about 0.1. After this value the system tends to be more stable but, as in the CDMA and OFDM case, the percentage may be almost stable but always remain messages for being delivered. There are similarities in the Performance over Latency scenario in CDMA, OFDM and CSMA techniques. These similarities indicate the fact that all these aforementioned techniques operate in a similar way in this case, but compared to TDMA it can be safely stated that TDMA outperforms better than CDMA, OFDM and CSMA due to the results that obtained by the simulation.

In the following Figure (Figure 47) the **Error Probability** that arises can be easily observed:

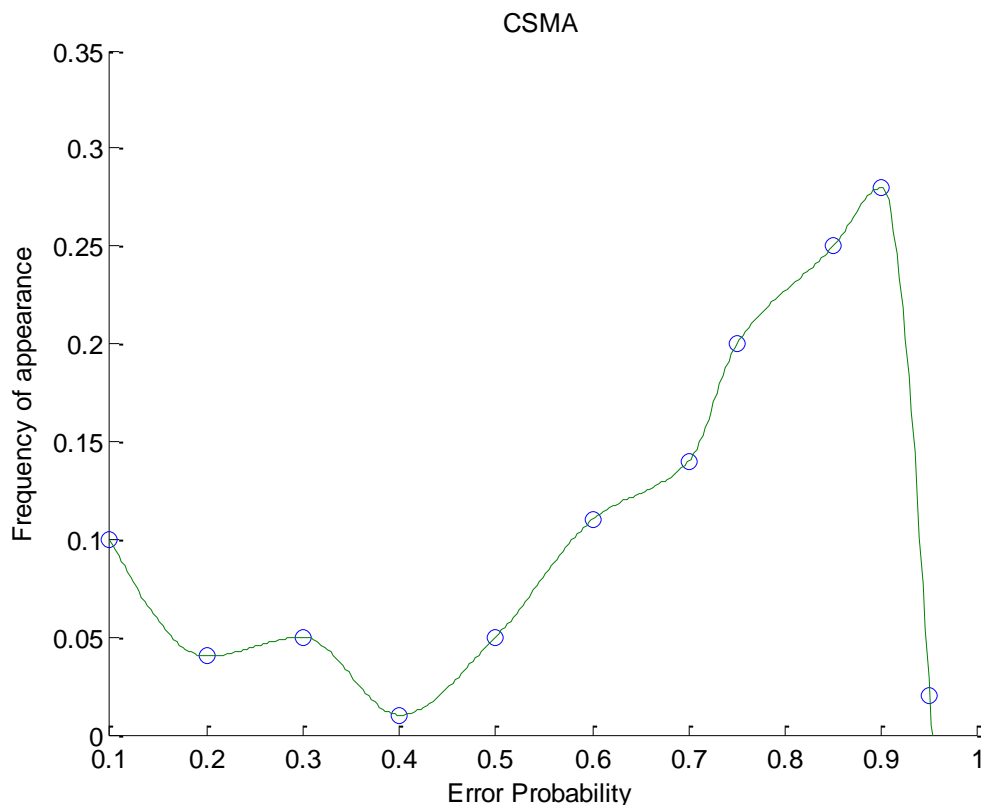


Figure 47: Error Probability results obtained after the conduction of the CSMA simulation

As far as Error Probability is concerned and the frequency of appearance, in the CSMA case, as it can be easily observed by the above diagram (Figure 47), the worst performance compared to all the other techniques is achieved. This can be supported by the results that obtained. As it is shown in the diagram of Figure 47 the CSMA case exhibits a variety of fluctuations. For low error probability values, variations and fluctuations are observed in the system. For the error probability values between 0.1 and 0.4 various fluctuations are observed in the system, where the error probability is increased and the frequency of appearance fluctuates in relatively low levels. Beyond this value of approximately 0.4 of error probability, the frequencies are rapidly increased and the system reaches its highest value at high error probability values. The system reaches its peak at the value of approximately 0.92 of error probability, which corresponds at a high frequency level. Furthermore, on the right of the peak are observed lower frequencies which correspond to higher error probabilities. The main point that has to be mentioned is that errors occur very often in the CSMA case, as it is clearly indicated by the conduction of our simulation. From all these results that were obtained, it can be supported that the worst performance is achieved as far as error probability is concerned in the CSMA case. The system is not stable due to the fact that the curve is fluctuated at high values of error probability and at various frequencies all the time. As it was previously mentioned, the system reaches its highest values at very high error probability values. Additionally, it remains in the high error probability area for high frequency values and this is a very significant factor for the stability of the system because the errors occur more often and this is an undesirable situation for the system.

3.6 Comparison of the Simulation Results

In this section we will offer a summary of the results that are finally obtained after the conduction of the simulation for different implementation scenarios for the four algorithms (TDMA, OFDM, CDMA and CSMA). As far as the results that are obtained we have to mention the following:

As far as **Throughput** is concerned, **CSMA** is the best algorithm with **OFDM** to reach similar levels of Throughput but with a bit worse results. The price of CSMA is the biggest compared to all the algorithms and multiplexing techniques (TDMA, CDMA and OFDM) that were also examined in this scenario. This means that, with the use of CSMA, our system remains stable for bigger amount of offered load and as a consequence the throughput reaches higher levels compared to those of the previous algorithms. Furthermore, it has to be mentioned that after the peak of the curve, that shown in the corresponding diagram of Throughput simulation for CSMA, it remained almost stable and as a consequence this means that even bigger load can be serviced and in wider range through this technique.

As far as **Performance over Latency** is concerned and the delivery of the messages, we chose to observe the time span between the start of the receiving of the message and the end of the message transmission, just before the message is erased from the system. In that case we notice that the best performance belongs to **TDMA**. It can be easily observed by the corresponding diagram of Performance over Latency simulation for TDMA that the performance that is achieved in TDMA has to exhibit vital and important results. To be more specific, for smaller periods of delivery time the percentage of messages that are delivered successfully reaches high prices. On the other hand, as the delivery time reaches higher levels the percentage of messages that have to be delivered is reduced, which means that the system shows more stable in comparison with the others that arise from the other algorithms (CDMA, OFDM and CSMA) that were examined in the corresponding Performance over Latency scenario.

As far as **Error Probability** is concerned, the best performance is observed in **CDMA**. Stable values are observed in the first section, having the maximum in low error probability. On the other hand low enough frequencies on the right correspond to higher error probabilities. Medium performance is noticed both OFDM and TDMA in this scenario. **OFDM** seems slightly better, because the maximum value of frequencies is located on slightly higher values of error probability in TDMA. Last but not least, the worst case in error probability performance seems to be CSMA. From all the results that were obtained, it can be supported that the system is not stable due to the fact that the curve is fluctuated at high values of error probability and at various frequencies all the time. As it was previously mentioned in the simulation, the system reaches its highest values at very high error probability values.

In the following Table (Table 3) the main research findings that were obtained after the conduction of the simulation can be easily shown. The best, most effective, algorithm that corresponds to each simulation parameter separately is clearly observed in the Table 3 bellow:

	TDMA	OFDM	CDMA	CSMA
Throughput		X*		X
Performance over Latency	X			
Error Probability		X*	X	

* exhibits similar, with the best one technique, but slightly worse results

Table 3: Best algorithm for each Parameter separately

It has to be mentioned that as far as the Throughput and Error Probability parameter is concerned, we have also selected the **OFDM** technique along with CSMA in the first case and CDMA in the second one, due to the fact that in both cases it exhibited similar results with CSMA and CDMA but these two, CSMA and CDMA, seemed to be slightly better.

Conclusion

The 4th Generation of telecommunications promises greater capacities along with wider frequency areas and greater transmission rates. In order to achieve the efficient transmission of the messages from one antenna to the other, through a usually noisy channel, mechanics have invented a series of multiplexing algorithms in order to distinguish the messages that arrive to a receiver combined with other messages as well as noise. The property of correctly retrieving an errorless message is called reconfigurability.

Reconfigurability is attempted with a series of measures in 4th Generation telecommunications. Sometimes we change our hardware, for example adding multiple antennas in the receiver and measuring multiple beams and sometimes we use a variety of algorithms as a means of distinguishing the messages via software. Among the most famous algorithms are Time Division Multiple Access, **TDMA**, Code Division Multiple Access, **CDMA**, Orthogonal Frequency Division Multiplexing, **OFDM**, as well as Carrier Sense Multiple Access, **CSMA**. Each of them works in a different way on the analog signal received by an antenna. TDMA utilizes every time interval with different time interval prefixes in order to send and receive distinguishable messages. OFDM multiplies every signal with a carrier orthogonal to other carriers in terms of frequency. As a result, every signal lies in a different frequency interval. Multiplication also takes place in CDMA, but with a unique code for each signal which comprises of ones and minus ones, for example -1, -1, 1, 1, -1. This widens the signal in terms of frequency and makes each signal again almost orthogonal to the other. Finally, it has been proven that an efficient allocation of channels and antennas and a careful use of them, as in CSMA, can give us competitive results.

All four multiplexing algorithms aim at greater reconfigurability. Consequently, in order to achieve further reconfigurability, we should combine them. The combination can take place in different ways and in different orders. Firstly, in our multiplexing procedure

comes CSMA multiplexing so as to take advantage of the various channels of the transmission. Secondly, we perform TDMA multiplexing to divide time into intervals. The next step is OFDM multiplexing in order to take signals almost orthogonal to one another. Lastly, a code of ones and minus ones is multiplied with each signal to take it unique, according to CDMA. Each algorithm operates in a different way in a communication system and various benefits can be achieved by their combination. In this way, we hope that with the implementation of this combined algorithm to a system we will logically take advantage as good and as fast as possible of the benefits of every multiplexing algorithm and further reconfigurability can be successfully achieved.

We have tried to simulate the results of those four algorithms with help of an open – source network simulation tool called OMNEST. OMNEST is an object-oriented modular discrete event network simulation framework, working in cooperation with C++. It has a generic architecture, so it can be used in various problem domains such as modeling of wired and wireless communication networks, modeling of queuing networks, evaluating performance aspects of complex software systems and in general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages. We have based our analysis on **aloha network**, which is a network of mobile phones and a server, where messages coming from the mobile phones are produced exponentially.

In our results, we have included a **Throughput** analysis, a **Performance over latency** graph as well as a **Probability of Error** measurement, comparing the four multiplexing algorithms mentioned above. Our arithmetic results have shown that all four algorithms have displayed competitive results. However, each of them has its advantages in terms of the three parameters measured. TDMA has proven to be the fastest in our experiment however CDMA and OFDM seem to manage better as far as Error Probability is concerned, where both CSMA and TDMA are left behind. Let us also note that our results rely on our specific parameters choice and our system, nevertheless, in terms of Throughput, they agree with previously conducted experiments as CSMA and OFDM are usually proven to be more efficient in that areas.

References

- [1] Gazis Vangelis, Houssos Nikos, Alonistioti Nancy, Merakos Lazaros Communication Networks Laboratory, University of Athens, Athens, Greece, Reconfiguration Management in 4G Mobile Networks: Requirements, Process and Architecture
- [2] Rhituparna Paul, Nishat Kabir and Tahnia Farheen, 4G Mobile Architecture
- [3] Keith E.Nolan, Reconfigurable OFDM Systems
- [4] Day, J. D. and H. Zimmermann (1983). "The OSI reference model." Proceedings of the IEEE **71**(12): 1334-1340.
- [5] Bluetooth, S. (2005). "Specification of the Bluetooth system." Core, version 1: 2005-2010.
- [6] Williams, S. (2000). "IrDA: past, present and future." Personal Communications, IEEE **7**(1): 11-19.
- [7] Leavitt, N. (2007). "For Wireless USB, the Future Starts Now." Computer **40**(7): 14-16.
- [8] Alliance, Z. B. (2005). "Zigbee specification." ZigBee document 053474r06, version 1.
- [9] Ochsner, H. (1989). DECT-digital European cordless telecommunications, IEEE.
- [10] Khun-Jush, J., P. Schramm, et al. (1999). Structure and performance of the HIPERLAN/2 physical layer, IEEE.

- [11] Bianchi, G. (2000). "Performance analysis of the IEEE 802.11 distributed coordination function." Selected Areas in Communications, IEEE Journal on **18**(3): 535-547.
- [12] Hoymann, C., M. Puttner, et al. (2003). The HIPERMAN standard-a performance analysis.
- [13] Ghosh, A., D. R. Wolter, et al. (2005). "Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential." Communications Magazine, IEEE **43**(2): 129-136.
- [14] Mouly, M. and M. B. Pautet (1992). The GSM system for mobile communications, Telecom Publishing.
- [15] Agashe, P., R. Rezaifar, et al. (2004). "cdma2000® high rate broadcast packet data air interface design." Communications Magazine, IEEE **42**(2): 83-89.
- [16] Dahlman, E., B. Gudmundson, et al. (1998). "UMTS/IMT-2000 based on wideband CDMA." Communications Magazine, IEEE **36**(9): 70-80.
- [17] Dahlman, E. (2008). 3G evolution: HSPA and LTE for mobile broadband, Academic Press.
- [18] Day, J. D. and H. Zimmermann (1983). "The OSI reference model." Proceedings of the IEEE **71**(12): 1334-1340.
- [19] Toh, C. K. K. (2001). Ad Hoc Wireless Networks: Protocols and Systems, Prentice Hall PTR Upper Saddle River, NJ, USA.

- [20] Jung, P., P. W. Baier, et al. (1993). "Advantages of CDMA and spread spectrum techniques over FDMA and TDMA in cellular mobile radio applications." Vehicular Technology, IEEE Transactions on **42**(3): 357-364.
- [21] Ergen, M., D. Lee, et al. (2004). "WTRP-wireless token ring protocol." Vehicular Technology, IEEE Transactions on **53**(6): 1863-1881.
- [22] Roberts, L. G. (1975). "ALOHA packet system with and without slots and capture." ACM SIGCOMM Computer Communication Review **5**(2): 28-42.
- [23] Nutt, G. and D. Bayer (1982). "Performance of CSMA/CD networks under combined voice and data loads." Communications, IEEE Transactions on **30**(1): 6-11.
- [24] Karn, P. (1990). MACA-a new channel access method for packet radio.
- [25] Bianchi, G., L. Fratta, et al. (1996). Performance evaluation and enhancement of the CSMA/CA MAC protocol for 802.11 wireless LANs, IEEE.
- [26] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Code: Turbo Code," *Proceedings of the 1993 IEEE International Conference on Communications*, 1064–1070 (Geneva, Switzerland, May 1993).
- [27] M. R. Shane and R. Wesel, "Parallel Concatenated Turbo Codes for Continuous Phase Modulation," *Proceedings of 2000 IEEE*
- [28] G. Lui and K. Tsai, "A Soft-Output Viterbi Algorithm Demodulation for Pre-coded Binary CPM Signal," to appear in *Proceedings of 20th AIAA International Satellite System Conference* (May 2002).

- [29] D. Sklar and C. C. Wang, "On the Performance of High Rate Turbo Codes," to be published in *Proceedings of 2002 IEEE Aerospace Conference* (March 2002).
- [30] B. Vucetic and J. Yuan, *Turbo Codes* (Kluwer Academic Publishers, Boston, 2000).
- [31] C. C. Wang, "Mitigating the Error Floor for Turbo Codes," *Proceedings of Globecom '98* (November 1998)
- [32] C. C. Wang, "Improving Faded Turbo Code Performance Using Biased Channel Side Information," *Proceedings of Military Communications Conference 1999* (October 1999)
- [33] C. C. Wang and D. Sklar, "A Novel Metric Transformation to Improve Performance of the Turbo Coded System with DPSK Modulation in Fading Environments," *Proceedings of Military Communications Conference 2001* (October 2001)
- [34] C. C. Wang and D. Sklar, "Performance of the Turbo Coded System with DPSK Modulation Using Enhanced Decoding Metrics and Matched Channel Side Information," to appear in *Proceedings of 2002 International Communications Conference* (April 2002)
- [35] Sinem Coleri Ergen and Pravin Varaiya, TDMA Scheduling Algorithms for Wireless Sensor Networks
- [36] Serkan Ender Hakyemez, Scheduling algorithms in Wireless CDMA Networks
- [37] Jo Woon Chong, Dan Keun Sung, , and Youngchul Sung, "Cross-Layer Performance Analysis for CSMA/CA Protocols: Impact of Imperfect Sensing"
- [38] L. Kleinrock and F. Tobagi, "Packet switching in radio channels:Part I—Carrier sense multiple-access modes and their throughput-delaycharacteristics," *IEEE Trans. Commun.*, vol. COM-23, no. 2, pp. 1400–1416, Dec. 1975.

- [39] H. Takagi and L. Kleinrock, "Throughput analysis for persistent CSMA systems," *IEEE Trans. Commun.*, vol. COM-33, no. 7, pp. 627–638, Jul. 1985.
- [40] Metcalfe, R. M. and D. R. Boggs (1976). "Ethernet: distributed packet switching for local computer networks." *Commun. ACM* **19**(7): 395-404.
- [41] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11 WG, Jun. 1997.
- [42] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std. 802.15.4, 2003.
- [43] Schiller J., *Mobile communications*, 1st edition, Addison – Wesley, London, 2000
- [44] C. Y. Jung, H. Y. Hwang, D. K. Sung, and G. U. Hwang, "Enhanced Markov chains model and throughput analysis of the slotted CSMA/CA of the IEEE 802.15.4 under an unsaturated traffic condition," *IEEE Trans. Veh. Technol.*, vol. 58, no. 1, pp. 473–478, Jan. 2009.
- [45] J.W. Chong, Y. Sung, and D. K. Sung, "Cross-layer performance analysis for CSMA/CA systems: Impact of imperfect sensing," in *Proc. IEEE SPAWC*, Jul. 2008, pp. 96–100.
- [46] J. W. Chong, Y. Sung, and D. K. Sung, "Analysis of CSMA/CA systems under carrier sensing error: Throughput, delay and sensitivity," in *Proc. IEEE Globecom*, Dec. 2008, pp. 1–6.
- [47] E. Ziouva and T. Antonakopoulos, "CSMA/CA performance under high traffic conditions: Throughput and delay analysis," *Comput. Commun.*, vol. 25, no. 3, pp. 313–321, Feb. 2002.

- [48] C. H. Foh and J. W. Tantra, "Comments on IEEE 802.11 saturation throughput analysis with freezing of backoff counters," *IEEE Commun. Lett.*, vol. 9, no. 2, pp. 130–132, Feb. 2005.
- [49] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 3, pp. 535–547, Mar. 2000.
- [50] G. Bianchi and I. Tinnirello, "Remarks on IEEE 802.11 DCF performance analysis," *IEEE Commun. Lett.*, vol. 9, no. 8, pp. 765–767, Aug. 2005.
- [51] H. Wu, Y. Peng, K. Long, S. Cheng, and J. Ma, "Performance of reliable transport protocol over IEEE 802.11 wireless LANs: Analysis and enhancement," in *Proc. IEEE INFOCOM*, 2002, vol. 2, pp. 599–607.
- [52] Andrisano, O.; Grandi, G.; Raffaelli, C.; , "Analytical model of busy channel multiple access (BCMA) for packet radio networks in a local environment," *Vehicular Technology, IEEE Transactions on* , vol.39, no.4, pp.299-307, Nov 1990
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=61351&isnumber=2233>
- [53] Chakraborty, S.S.; Wager, S.; "The inhibit sense multiple access with polling (ISMA/P): a MAC-RLP combined procedure for packet mode data over noninterleaved channel for GSM-GPRS services," *Vehicular Technology Conference, 1996. 'Mobile Technology for the Human Race', IEEE 46th* , vol.2, no., pp.776-780 vol.2, 28 Apr-1 May 1996
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=501417&isnumber=10813>
- [54] F. A. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels: Part II - the Hidden Terminal Problem in Carrier Sense Multiple-Access Modes and the Busy-Tone Solution," *IEEE Trans. Commun.*, vol. COM-23, no. 12, pp. 1417-1433, 1975.

- [55] Jack M. Holtzman, "A simple, Accurate Method to Calculate Spread Spectrum Multiple-Access Error Probabilities," *IEEE Trans. On Communications*, vol. 40, no. 3, March 1992.
- [56] T.Yamazato , T. Sato , H. Okada , M. Katayama, and A. Ogawa "Throughput and Delay Analysis of DS/SSMA Unslotted ALOHA by Non-Perfect Capture," *Proc. IEEE Int. Conf. Universal Personal Communications*, 1995.
- [57] R.K. Morrow and J.S. Lehnert "Bit-to-bit error dependence in slotted DS/SSMA packet systems with random signature sequences," *IEEE Trans. On Communications*, vol. 37 pp. 1052-1061 Oct. 1989.
- [58] Ali T.Koc, Ozgur Ozdemir ,Murat Torlak and William P.Osborne, Opnet Simulation of Random Access CDMA Networks.
- [59] Livia Ruiz, Gerard Baldwin and Ronan Farrell, Reconugurable Radio Testbed, Irish Signal and Systems Conference, Dublin, June 28-30, 2006.
- [60] Analog Devices "Direct Conversion Quadrature Demodulator". Graphics TCP 30, 12/20.
- [61] John G. Proakis, "Digital Communications, 3rd Edition"

Appendix of the Code

TDMA

Server.cc

```
#include "Server.h"

namespace aloha {
int bu=0;
double ba=0;
double ba2=0;
double rt=0;
double uee;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;
Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
    collisionLengthSignal = registerSignal("collisionLength");

    emit(receiveSignal, 0.0);
    emit(receiveBeginSignal, 0.0);

    if (ev.isGUI())
        getDisplayString().setTagArg("i2",0,"x_off");
}
```

```

}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        cPacket *pkt = check_and_cast<cPacket *>(msg);

        ASSERT(pkt->isReceptionStart());
        simtime_t endReceptionTime = simTime() + pkt->getDuration();
        emit(receiveBeginSignal, ++receiveCounter);

        if (!channelBusy)
        {
            EV << "started receiving\n";
            recvStartTime = simTime();
            channelBusy = true;
            scheduleAt(endReceptionTime, endRxEvent);
        }
    }
}

```

```

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_yellow");
            getDisplayString().setTagArg("t",0,"RECEIVE");
            getDisplayString().setTagArg("t",2,"#808000");
        }
    }
    else
    {
        EV << "another frame arrived while receiving --
collision!\n";

        if (currentCollisionNumFrames==0)
            currentCollisionNumFrames = 2;
        else
            currentCollisionNumFrames++;
        //

        // our interv.
        bu=bu+currentCollisionNumFrames;
        // this counter is not set to zero every time
        // it grows along with every packet failure
        // our interv.
        if (endReceptionTime > endRxEvent->getArrivalTime())
        {
            cancelEvent(endRxEvent);
            scheduleAt(endReceptionTime, endRxEvent);
        }

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_red");
            getDisplayString().setTagArg("t",0,"COLLISION");
            getDisplayString().setTagArg("t",2,"#800000");
            char buf[32];
            sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
            bubble(buf);
        }
    }
    channelBusy = true;
    delete pkt;
}

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl; // total number of collisions
    recordScalar("duration", simTime());
}
};

```

```

#include "Server.h"

namespace aloha {
int bu=0;
Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
    collisionLengthSignal = registerSignal("collisionLength");

    emit(receiveSignal, 0.0);
    emit(receiveBeginSignal, 0.0);

    if (ev.isGUI())
        getDisplayString().setTagArg("i2",0,"x_off");
}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the original
message received
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
        }
    }
}

```



```

        emit(receiveSignal, 0);
    }
    else
    {
        // start of collision at recvStartTime
        cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
        emit(collisionSignal, &tmp);

        emit(collisionLengthSignal, dt);
    }

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    emit(receiveBeginSignal, receiveCounter);

    // update network graphics
    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_off");
        getDisplayString().setTagArg("t",0,"");
    }
}
else
{
    cPacket *pkt = check_and_cast<cPacket *>(msg);

    ASSERT(pkt->isReceptionStart());
    simtime_t endReceptionTime = simTime() + pkt->getDuration();

    emit(receiveBeginSignal, ++receiveCounter);

    if (!channelBusy)
    {
        EV << "started receiving\n";
        recvStartTime = simTime();
        channelBusy = true;
        scheduleAt(endReceptionTime, endRxEvent);

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_yellow");
            getDisplayString().setTagArg("t",0,"RECEIVE");
            getDisplayString().setTagArg("t",2,"#808000");
        }
    }
    else
    {
        EV << "another frame arrived while receiving --
collision!\n";

        if (currentCollisionNumFrames==0)
            currentCollisionNumFrames = 2;
        else
            currentCollisionNumFrames++;
        // our interv.
        ERPR=erf(sqrt(2*ba));
    }
}
}

```

```

EV << "Error Prob"<<ERPR<<endl;

//bu=bu+currentCollisionNumFrames;
bu=snrInfo.rcvdPower / noiseLevel;
// our interv.
if (endReceptionTime > endRxEvent->getArrivalTime())
{
    cancelEvent(endRxEvent);
    scheduleAt(endReceptionTime, endRxEvent);
}

// update network graphics
if (ev.isGUI())
{
    getDisplayString().setTagArg("i2",0,"x_red");
    getDisplayString().setTagArg("t",0,"COLLISION");
    getDisplayString().setTagArg("t",2,"#800000");
    char buf[32];
    sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
    bubble(buf);
}
}
channelBusy = true;
delete pkt;
}
}

void Server::finish()
{
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "Time of Delivery " << simTime() - rcvStartTime<<endl;
    recordScalar("duration", simTime());
    recordScalar(bu);
}
};

```

The code of Server.h file is the following

Server.h

```

#ifndef __ALOHA_SERVER_H_
#define __ALOHA_SERVER_H_

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha server; see NED file for more info.

```

```

*/
class Server : public cSimpleModule
{double snr;
  private:
    // state variables, event pointers
    bool channelBusy;
    cMessage *endRxEvent;
    long currentCollisionNumFrames;
    long receiveCounter;
    simtime_t recvStartTime;
    //double snr;
    // statistics
    simsignal_t receiveBeginSignal;
    simsignal_t receiveSignal;
    simsignal_t collisionLengthSignal;
    simsignal_t collisionSignal;

  public:
    Server();
    virtual ~Server();

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

}; //namespace

#endif

```

The code of Host.cc file is the following:

Host.cc

```

#include "Host.h"
namespace aloha {

Define_Module(Host);

Host::Host()
{
    endTxEvent = NULL;
}

Host::~~Host()
{
    cancelAndDelete(endTxEvent);
}

```

```

void Host::initialize()
{
    server = simulation.getModuleByPath("server");
    if (!server) error("server not found");

    txRate = par("txRate");
    //
    snr = par("snr");
    radioDelay = par("radioDelay");
    iaTime = &par("iaTime");
    pkLenBits = &par("pkLenBits");

    slotTime = par("slotTime");
    isSlotted = slotTime>0;
    WATCH(slotTime);
    WATCH(isSlotted);

    endTxEvent = new cMessage("send/endTx");
    state = IDLE;
    pkCounter = 0;
    WATCH((int&)state);
    WATCH(pkCounter);

    if (ev.isGUI())
        getDisplayString().setTagArg("t",2,"#808000");

    scheduleAt(getNextTransmissionTime(), endTxEvent);
}

void Host::handleMessage(cMessage *msg)
{
    ASSERT(msg==endTxEvent);

    if (state==IDLE)
    {
        // generate packet and schedule timer when it ends
        char pname[40];
        sprintf(pname,"pk-%d-#%d", getId(), pkCounter++);
        EV << "generating packet " << pname << endl;

        state = TRANSMIT;

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"yellow");
            getDisplayString().setTagArg("t",0,"TRANSMIT");
        }

        cPacket *pk = new cPacket(pname);
        pk->setBitLength(pkLenBits->longValue());
        // manual pg 112
        snr=par("snr");
        simtime_t duration = pk->getBitLength() / txRate;
        sendDirect(pk, radioDelay , duration, server->gate("in"));
    }
}

```

```

        scheduleAt(simTime()+duration, endTxEvent);
    }
    else if (state==TRANSMIT)
    {
        // endTxEvent indicates end of transmission
        state = IDLE;

        // schedule next sending
        scheduleAt(getNextTransmissionTime(), endTxEvent);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        error("invalid state");
    }
}

simtime_t Host::getNextTransmissionTime()
{
    simtime_t t = simTime()+iaTime->doubleValue();

    if (!isSlotted)
        return t;
    else
        // align time of next transmission to a slot boundary
        return slotTime * ceil(t/slotTime);
}

}; //namespace

```

The code of Host.h file is the following:

Host.h

```

#ifndef __ALOHA_HOST_H_
#define __ALOHA_HOST_H_

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha host; see NED file for more info.
 */
class Host : public cSimpleModule
{
public:
    double snr;
private:

```

```

// parameters
simtime_t radioDelay;
double txRate;
cPar *iaTime;
cPar *pkLenBits;
simtime_t slotTime;
bool isSlotted;
//

// state variables, event pointers etc
cModule *server;
cMessage *endTxEvent;
enum {IDLE=0, TRANSMIT=1} state;
int pkCounter;

public:
Host();
virtual ~Host();
protected:
virtual void initialize();
virtual void handleMessage(cMessage *msg);
simtime_t getNextTransmissionTime();

};

}; //namespace

#endif

```

The code of Aloha.ned is the following:

Aloha.ned

```

network Aloha
{
    parameters:
        int numHosts; // number of hosts
        double txRate @unit(bps); // transmission rate
        double slotTime @unit(ms); // zero means no slots (pure Aloha)
        volatile double snr;
        @display("bgi=background/terrain,s");
    submodules:
        server: Server
        {snr=snr;};

        host[numHosts]: Host {
            txRate = txRate;
            slotTime = slotTime;
            snr=snr;
        }
}

```

The code of Server.ned is the following:

Server.ned

```

simple Server
{
    parameters:
        @display("i=device/antennatower_1");
        @signal[receiveBegin] (type="long"); // increases with each new
frame arriving to the server and drops to 0 if the channel becomes
finally idle
        @signal[receive] (type="long"); // for successful receptions
(non-collisions): 1 at the start of the reception, 0 at the end of the
reception
        @signal[collision] (type="long"); // the number of collided
frames at the beginning of the collision period
        @signal[collisionLength] (type="double"); // the length of the
last collision period at the end of the collision period

        @statistic[receiveBegin] (source="receiveBegin"; record=vector;
interpolationmode=sample-hold; title="receive begin");
        @statistic[channelUtilization] (source="timeavg(receive)";
record=last; interpolationmode=linear; title="channel utilization");
        @statistic[collisionMultiplicity] (source=collision;
record=vector?, histogram; title="collision multiplicity");

@statistic[collisionLength] (record=vector?, histogram, mean, sum, max;
title="collision length");
        @statistic[receivedFrames] (source="sum(receive)"; record=last;
title="received frames");
        @statistic[collidedFrames] (source="sum(collision)"; record=last;
title="collided frames");
        volatile double snr;
        gates:
        input in @directIn;
}

```

And finally the code of Host.ned is the following:

Host.ned

```

simple Host
{
    parameters:
        double txRate @unit(bps); // transmission rate
        double radioDelay @unit(s); // propagation delay of radio

    link
        volatile int pkLenBits @unit(b); // packet length in bits
        volatile double iaTime @unit(s); // packet interarrival time
}

```

```

        double slotTime @unit(s);           // zero means no slots (pure
Aloha)
        volatile double snr;
        @display("i=device/pc_s");
    }

```

CDMA

Server.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 1992-2008 Andras Varga
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// `license' for details on this and other legal matters.
//

#include "Server.h"

namespace aloha {
int bu=0;
double ba=0;
double ba2=0;
double rt=0;
double uee;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;
Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);
}

```



```

currentCollisionNumFrames = 0;
receiveCounter = 0;
WATCH(currentCollisionNumFrames);

receiveBeginSignal = registerSignal("receiveBegin");
receiveSignal = registerSignal("receive");
collisionSignal = registerSignal("collision");
collisionLengthSignal = registerSignal("collisionLength");

emit(receiveSignal, 0.0);
emit(receiveBeginSignal, 0.0);

if (ev.isGUI())
    getDisplayString().setTagArg("i2",0,"x_off");
}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
}

```

```

}
else
{
    cPacket *pkt = check_and_cast<cPacket *>(msg);

    ASSERT(pkt->isReceptionStart());
    simtime_t endReceptionTime = simTime() + pkt->getDuration();
    emit(receiveBeginSignal, ++receiveCounter);

    if (!channelBusy)
    {
        EV << "started receiving\n";
        recvStartTime = simTime();
        channelBusy = true;
        scheduleAt(endReceptionTime, endRxEvent);

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_yellow");
            getDisplayString().setTagArg("t",0,"RECEIVE");
            getDisplayString().setTagArg("t",2,"#808000");
        }
    }
    else
    {
        EV << "another frame arrived while receiving --
collision!\n";

        if (currentCollisionNumFrames==0)
            currentCollisionNumFrames = 2;
        else
            currentCollisionNumFrames++;
        //
        uee=(double)currentCollisionNumFrames;
        mk=(Nee*uee)/6;
        sik=sqrt((uee/4)*((23*pow(Nee,2)/360+(Nee-1)*(0.05+(uee-
1)/36))));
        rt=(2/3)*erf(pow(Nee,2)/(2*mk+0.5*(pow(Nee,2)/ba)))+(
1/6)*erf(pow(Nee,2)/(2*(mk+(sik*sqrt(3)))+0.5*(pow(Nee,2)/ba))
+(1/6)*erf(pow(Nee,2)/(2*(mk-(sik*sqrt(3)))+0.5*(pow(Nee,2)/ba)));
        Thr=la*(pow(la,uee)/uee)*exp(-la)*(1-rt); // estimation of
throughput
        // our interv.
        bu=bu+currentCollisionNumFrames;
        ba2=par("snr");
        EV << "ba2: " << ba2<<endl;
        // our interv.
        if (endReceptionTime > endRxEvent->getArrivalTime())
        {
            cancelEvent(endRxEvent);
            scheduleAt(endReceptionTime, endRxEvent);
        }

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_red");

```

```

        getDisplayString().setTagArg("t",0,"COLLISION");
        getDisplayString().setTagArg("t",2,"#800000");
        char buf[32];
        sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
        bubble(buf);
    }
}
channelBusy = true;
delete pkt;
}
}

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "ba: " << ba<<endl;
    EV << "Throughput: " << Thr<<endl;
    recordScalar("duration", simTime());
}

};

```

```

#include "Server.h"

namespace aloha {
int bu=0;
double ba=0;
double ba2=0;
double rt=0;
double uee;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;
Define_Module(Server);

```

```

Server::Server()
{
    endRxEvent = NULL;
}

Server::~~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;
}

```

```

gate("in")->setDeliverOnReceptionStart(true);

currentCollisionNumFrames = 0;
receiveCounter = 0;
WATCH(currentCollisionNumFrames);

receiveBeginSignal = registerSignal("receiveBegin");
receiveSignal = registerSignal("receive");
collisionSignal = registerSignal("collision");
collisionLengthSignal = registerSignal("collisionLength");

emit(receiveSignal, 0.0);
emit(receiveBeginSignal, 0.0);

if (ev.isGUI())
    getDisplayString().setTagArg("i2",0,"x_off");
}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");

```

```

        getDisplayString().setTagArg("t",0,"");
    }
}
else
{
    cPacket *pkt = check_and_cast<cPacket *>(msg);

    ASSERT(pkt->isReceptionStart());
    simtime_t endReceptionTime = simTime() + pkt->getDuration();
    emit(receiveBeginSignal, ++receiveCounter);

    if (!channelBusy)
    {
        EV << "started receiving\n";
        recvStartTime = simTime();
        channelBusy = true;
        scheduleAt(endReceptionTime, endRxEvent);

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_yellow");
            getDisplayString().setTagArg("t",0,"RECEIVE");
            getDisplayString().setTagArg("t",2,"#808000");
        }
    }
    else
    {
        EV << "another frame arrived while receiving --
collision!\n";

        if (currentCollisionNumFrames==0)
            currentCollisionNumFrames = 2;
        else
            currentCollisionNumFrames++;
        //
        uee=(double)currentCollisionNumFrames;
        mk=(Nee*uee)/6;
        sik=sqrt((uee/4)*((23*pow(Nee,2)/360+(Nee-1)*(0.05+(uee-1)/36))));
        rt=(2/3)*erf(pow(Nee,2)/(2*mk+0.5*(pow(Nee,2)/ba)))+(1/6)*erf(pow(Nee,2)/(2*(mk+(sik*sqrt(3)))+0.5*(pow(Nee,2)/ba)))+(1/6)*erf(pow(Nee,2)/(2*(mk-(sik*sqrt(3)))+0.5*(pow(Nee,2)/ba)));
        Thr=la*(pow(la,uee)/uee)*exp(-la)*(1-rt); // estimation of throughput
        // our interv.
        bu=bu+currentCollisionNumFrames;
        ba2=par("snr");
        EV << "Error Probability " << rt<<endl;
        EV << "ba2: " << ba2<<endl;
        // our interv.
        if (endReceptionTime > endRxEvent->getArrivalTime())
        {
            cancelEvent(endRxEvent);
            scheduleAt(endReceptionTime, endRxEvent);
        }

        // update network graphics

```

```

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_red");
            getDisplayString().setTagArg("t",0,"COLLISION");
            getDisplayString().setTagArg("t",2,"#800000");
            char buf[32];
            sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
            bubble(buf);
        }
    }
    channelBusy = true;
    delete pkt;
}
}

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "Time of Delivery " << simTime() - recvStartTime<<endl;
    EV << "Throughput: " << Thr<<endl;
    recordScalar("duration", simTime());
}

```

The code of Server.h file is the following:

Server.h

```

#ifndef __ALOHA_SERVER_H
#define __ALOHA_SERVER_H

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha server; see NED file for more info.
 */
class Server : public cSimpleModule
{
public:
    double snr;
private:
    // state variables, event pointers
    bool channelBusy;
    cMessage *endRxEvent;
    long currentCollisionNumFrames;
    long receiveCounter;
    simtime_t recvStartTime;
    //double snr;
    // statistics
    simsignal_t receiveBeginSignal;
    simsignal_t receiveSignal;
    simsignal_t collisionLengthSignal;
    simsignal_t collisionSignal;
}
}

```

```

public:
    Server();
    virtual ~Server();

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

}; //namespace

#endif

```

The code of Host.cc file is the following:

Host.cc

```

#include "Host.h"
namespace aloha {

Define_Module(Host);

Host::Host()
{
    endTxEvent = NULL;
}

Host::~Host()
{
    cancelAndDelete(endTxEvent);
}

void Host::initialize()
{
    server = simulation.getModuleByPath("server");
    if (!server) error("server not found");

    txRate = par("txRate");
    //
    snr = par("snr");
    radioDelay = par("radioDelay");
    iaTime = &par("iaTime");
    pkLenBits = &par("pkLenBits");

    slotTime = par("slotTime");
    isSlotted = slotTime>0;
    WATCH(slotTime);
    WATCH(isSlotted);
}

```

```

endTxEvent = new cMessage("send/endTx");
state = IDLE;
pkCounter = 0;
WATCH((int&)state);
WATCH(pkCounter);

if (ev.isGUI())
    getDisplayString().setTagArg("t",2,"#808000");

scheduleAt(getNextTransmissionTime(), endTxEvent);
}

void Host::handleMessage(cMessage *msg)
{
    ASSERT(msg==endTxEvent);
    if (state==IDLE)
    {
        // generate packet and schedule timer when it ends
        char pkname[40];
        sprintf(pkname,"pk-%d-#%d", getId(), pkCounter++);
        EV << "generating packet " << pkname << endl;

        state = TRANSMIT;

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"yellow");
            getDisplayString().setTagArg("t",0,"TRANSMIT");
        }

        cPacket *pk = new cPacket(pkname);
        pk->setBitLength(pkLenBits->longValue());
        // manual pg 112
        snr=par("snr");
        simtime_t duration = pk->getBitLength() / txRate;
        sendDirect(pk, radioDelay , duration, server->gate("in"));

        scheduleAt(simTime()+duration, endTxEvent);
    }
    else if (state==TRANSMIT)
    {
        // endTxEvent indicates end of transmission
        state = IDLE;

        // schedule next sending
        scheduleAt(getNextTransmissionTime(), endTxEvent);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else

```



```

    {
        error("invalid state");
    }
}

simtime_t Host::getNextTransmissionTime()
{
    simtime_t t = simTime()+iaTime->doubleValue();

    if (!isSlotted)
        return t;
    else
        // align time of next transmission to a slot boundary
        return slotTime * ceil(t/slotTime);
}
};

```

The code of Host.h file is the following:

Host.h

```

#ifndef __ALOHA_HOST_H_
#define __ALOHA_HOST_H_

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha host; see NED file for more info.
 */
class Host : public cSimpleModule
{
public:
    double snr;
private:
    // parameters
    simtime_t radioDelay;
    double txRate;
    cPar *iaTime;
    cPar *pkLenBits;
    simtime_t slotTime;
    bool isSlotted;
    //

    // state variables, event pointers etc
    cModule *server;
    cMessage *endTxEvent;
    enum {IDLE=0, TRANSMIT=1} state;
    int pkCounter;

public:
    Host();
    virtual ~Host();
protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

```

```

    simtime_t getNextTransmissionTime();
};

}; //namespace

#endif

```

The code of Aloha.ned is the following:

Aloha.ned

```

network Aloha
{
    parameters:
        int numHosts; // number of hosts
        double txRate @unit(bps); // transmission rate
        double slotTime @unit(ms); // zero means no slots (pure Aloha)
        volatile double snr;
        @display("bgi=background/terrain,s");
    submodules:
        server: Server
        {snr=snr;};

        host[numHosts]: Host {
            txRate = txRate;
            slotTime = slotTime;
            snr=snr;
        }
}

```

The code of Server.ned is the following:

Server.ned

```

simple Server
{
    parameters:
        @display("i=device/antennatower_1");
        @signal[receiveBegin] (type="long"); // increases with each new
        frame arriving to the server and drops to 0 if the channel becomes
        finally idle
        @signal[receive] (type="long"); // for successful receptions
        (non-collisions): 1 at the start of the reception, 0 at the end of the
        reception
}

```

```

    @signal[collision] (type="long"); // the number of collided
frames at the beginning of the collision period
    @signal[collisionLength] (type="double"); // the length of the
last collision period at the end of the collision period

    @statistic[receiveBegin] (source="receiveBegin"; record=vector;
interpolationmode=sample-hold; title="receive begin");
    @statistic[channelUtilization] (source="timeavg(receive)";
record=last; interpolationmode=linear; title="channel utilization");
    @statistic[collisionMultiplicity] (source=collision;
record=vector?, histogram; title="collision multiplicity");

@statistic[collisionLength] (record=vector?, histogram, mean, sum, max;
title="collision length");
    @statistic[receivedFrames] (source="sum(receive)"; record=last;
title="received frames");
    @statistic[collidedFrames] (source="sum(collision)"; record=last;
title="collided frames");
    volatile double snr;
    gates:
    input in @directIn;
}

```

And finally the code of Host.ned is the following:

Host.ned

```

simple Host
{
    parameters:
        double txRate @unit(bps); // transmission rate
        double radioDelay @unit(s); // propagation delay of radio
link
        volatile int pkLenBits @unit(b); // packet length in bits
        volatile double iaTime @unit(s); // packet interarrival time
        double slotTime @unit(s); // zero means no slots (pure
Aloha)
        volatile double snr;
        @display("i=device/pc_s");
}

```

OFDM

Server.cc code

```

#include "Server.h"

namespace aloha {
int bu=0;
double ba=0;
double ba2=0;
double rt=0;
double uee=0;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;
Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
    collisionLengthSignal = registerSignal("collisionLength");

    emit(receiveSignal, 0.0);
    emit(receiveBeginSignal, 0.0);

    if (ev.isGUI())
        getDisplayString().setTagArg("i2", 0, "x_off");
}

```

```

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        cPacket *pkt = check_and_cast<cPacket *>(msg);

        ASSERT(pkt->isReceptionStart());
        simtime_t endReceptionTime = simTime() + pkt->getDuration();
        emit(receiveBeginSignal, ++receiveCounter);

        if (!channelBusy)
        {
            EV << "started receiving\n";
            recvStartTime = simTime();
            channelBusy = true;
            scheduleAt(endReceptionTime, endRxEvent);

            if (ev.isGUI())
            {

```

```

        getDisplayString().setTagArg("i2",0,"x_yellow");
        getDisplayString().setTagArg("t",0,"RECEIVE");
        getDisplayString().setTagArg("t",2,"#808000");
    }
}
else
{
    EV << "another frame arrived while receiving --
collision!\n";

    if (currentCollisionNumFrames==0)
        currentCollisionNumFrames = 2;
    else
        currentCollisionNumFrames++;

    // Pe error probability in M=16 QAM constellation
    // ofdm simulation
    uee=par("snr"); // first read the snr od this current
interference user
    ba2=ba2+ uee; // and update ba, the enrgy of
the interferers
    mk=ba/ba2; // calculating SNR level, our energy over
interferers
    mk=10*log10(mk);
    rt=(2-1/sqrt(16))*erf(sqrt((3/(2*15))*mk));
    Thr=la*(pow(la,uee)/ uee)*exp(-la)*(1-rt); // estimation of
throughput
    EV << "Throughput: " << Thr<<endl;
    bu=currentCollisionNumFrames;
    EV << "ba2: " << ba2<<endl; // on the screen for control
    EV << "collisionframes: " << bu<<endl;
    if (endReceptionTime > endRxEvent->getArrivalTime())
    {
        cancelEvent(endRxEvent);
        scheduleAt(endReceptionTime, endRxEvent);
    }

    // update network graphics
    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_red");
        getDisplayString().setTagArg("t",0,"COLLISION");
        getDisplayString().setTagArg("t",2,"#800000");
        char buf[32];
        sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
        bubble(buf);
    }
}
channelBusy = true;
delete pkt;
bu=0;
}
}

void Server::finish()
{
    rt=erf(ba);

```

```

EV << "duration: " << simTime()<<endl;
EV << "bu: " << bu<<endl;
EV << "ba: " << ba<<endl;
EV << "Throughput: " << Thr<<endl;
recordScalar("duration", simTime());
}

};

#include "Server.h"

namespace aloha {
int bu=0;
double ba=0;
double ba2=0;
double rt=0;
double uee=0;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;
Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
    collisionLengthSignal = registerSignal("collisionLength");

    emit(receiveSignal, 0.0);
    emit(receiveBeginSignal, 0.0);
}

```

```

    if (ev.isGUI())
        getDisplayString().setTagArg("i2",0,"x_off");
}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        cPacket *pkt = check_and_cast<cPacket *>(msg);

        ASSERT(pkt->isReceptionStart());
        simtime_t endReceptionTime = simTime() + pkt->getDuration();
        emit(receiveBeginSignal, ++receiveCounter);

        if (!channelBusy)
        {
            EV << "started receiving\n";
            recvStartTime = simTime();

```



```

channelBusy = true;
scheduleAt(endReceptionTime, endRxEvent);

if (ev.isGUI())
{
    getDisplayString().setTagArg("i2",0,"x_yellow");
    getDisplayString().setTagArg("t",0,"RECEIVE");
    getDisplayString().setTagArg("t",2,"#808000");
}
}
else
{
    EV << "another frame arrived while receiving --
collision!\n";

    if (currentCollisionNumFrames==0)
        currentCollisionNumFrames = 2;
    else
        currentCollisionNumFrames++;

    // Pe error probability in M=16 QAM constellation
    // ofdm simulation
    uee=par("snr"); // first read the snr od this current
interference user
    ba2=ba2+ uee; // and update ba, the enrgy of
the interferers
    mk=ba/ba2; // calculating SNR level, our energy over
interferers
    mk=10*log10(mk);
    rt=(2-1/sqrt(16))*erf(sqrt((3/(2*15))*mk));
    Thr=1a*(pow(1a,uee)/ uee)*exp(-1a)*(1-rt); // estimation of
throughput
    //EV << "Throughput: " << Thr<<endl;
    // Error Probability
    EV << "Probab.Error: " << rt<<endl;
    bu=bu+currentCollisionNumFrames;
    EV << "ba2: " << ba2<<endl; // on the screen for control
    EV << "collisionframes: " << bu<<endl;
    if (endReceptionTime > endRxEvent->getArrivalTime())
    {
        cancelEvent(endRxEvent);
        scheduleAt(endReceptionTime, endRxEvent);
    }

    // update network graphics
    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_red");
        getDisplayString().setTagArg("t",0,"COLLISION");
        getDisplayString().setTagArg("t",2,"#800000");
        char buf[32];
        sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
        bubble(buf);
    }
}
}
channelBusy = true;

```

```

EV << "TimeofDelivery: " << -recvStartTime +simTime() <<endl;

    delete pkt;
    bu=0;
}
}

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "ba: " << ba<<endl;
    EV << "Throughput: " << Thr<<endl;
    recordScalar("duration", simTime());
}
};

```

The code of Server.h is the following:

Server.h

```

#ifndef __ALOHA_SERVER_H
#define __ALOHA_SERVER_H

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha server; see NED file for more info.
 */
class Server : public cSimpleModule
{
public:
    double snr;
private:
    // state variables, event pointers
    bool channelBusy;
    cMessage *endRxEvent;
    long currentCollisionNumFrames;
    long receiveCounter;
    simtime_t recvStartTime;
    //double snr;
    // statistics
    simsignal_t receiveBeginSignal;
    simsignal_t receiveSignal;
    simsignal_t collisionLengthSignal;
    simsignal_t collisionSignal;

public:
    Server();
    virtual ~Server();

protected:

```

```

    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

}; //namespace

#endif

```

The code of Host.cc is the following:

Host.cc

```

#include "Host.h"
namespace aloha {

Define_Module(Host);

Host::Host()
{
    endTxEvent = NULL;
}

Host::~Host()
{
    cancelAndDelete(endTxEvent);
}

void Host::initialize()
{
    server = simulation.getModuleByPath("server");
    if (!server) error("server not found");

    txRate = par("txRate");
    //
    snr = par("snr");
    radioDelay = par("radioDelay");
    iaTime = &par("iaTime");
    pkLenBits = &par("pkLenBits");

    slotTime = par("slotTime");
    isSlotted = slotTime>0;
    WATCH(slotTime);
    WATCH(isSlotted);

    endTxEvent = new cMessage("send/endTx");
    state = IDLE;
    pkCounter = 0;
}

```

```

WATCH((int&) state);
WATCH(pkCounter);

if (ev.isGUI())
    getDisplayString().setTagArg("t",2,"#808000");

scheduleAt(getNextTransmissionTime(), endTxEvent);
}

void Host::handleMessage(cMessage *msg)
{
    ASSERT(msg==endTxEvent);

    if (state==IDLE)
    {
        // generate packet and schedule timer when it ends
        char pkname[40];
        sprintf(pkname,"pk-%d-#%d", getId(), pkCounter++);
        EV << "generating packet " << pkname << endl;

        state = TRANSMIT;

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"yellow");
            getDisplayString().setTagArg("t",0,"TRANSMIT");
        }

        cPacket *pk = new cPacket(pkname);
        pk->setBitLength(pkLenBits->longValue());
        // manual pg 112
        snr=par("snr");
        simtime_t duration = pk->getBitLength() / txRate;
        sendDirect(pk, radioDelay , duration, server->gate("in"));

        scheduleAt(simTime()+duration, endTxEvent);
    }
    else if (state==TRANSMIT)
    {
        // endTxEvent indicates end of transmission
        state = IDLE;

        // schedule next sending
        scheduleAt(getNextTransmissionTime(), endTxEvent);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        error("invalid state");
    }
}

```

```

    }
}

simtime_t Host::getNextTransmissionTime()
{
    simtime_t t = simTime()+iaTime->doubleValue();

    if (!isSlotted)
        return t;
    else
        // align time of next transmission to a slot boundary
        return slotTime * ceil(t/slotTime);
}

}; //namespace

```

The code of Host.h is the following:

Host.h

```

#ifndef __ALOHA_HOST_H_
#define __ALOHA_HOST_H_

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha host; see NED file for more info.
 */
class Host : public cSimpleModule
{double snr;
private:
    // parameters
    simtime_t radioDelay;
    double txRate;
    cPar *iaTime;
    cPar *pkLenBits;
    simtime_t slotTime;
    bool isSlotted;
    //

    // state variables, event pointers etc
    cModule *server;
    cMessage *endTxEvent;
    enum {IDLE=0, TRANSMIT=1} state;
    int pkCounter;

public:
    Host();
    virtual ~Host();

```

```

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    simtime_t getNextTransmissionTime();

};

}; //namespace

#endif

```

The code of Aloha.ned is the following:

Aloha.ned

```

network Aloha
{
    parameters:
        int numHosts; // number of hosts
        double txRate @unit(bps); // transmission rate
        double slotTime @unit(ms); // zero means no slots (pure Aloha)
        volatile double snr;
        @display("bgi=background/terrain,s");
    submodules:
        server: Server
        {snr=snr;};

        host[numHosts]: Host {
            txRate = txRate;
            slotTime = slotTime;
            snr=snr;
        }
}

```

The code of Server.ned is the following:

Server.ned

```

simple Server
{
    parameters:
        @display("i=device/antennatower_1");
        @signal[receiveBegin](type="long"); // increases with each new
        frame arriving to the server and drops to 0 if the channel becomes
        finally idle
}

```

```

    @signal[receive](type="long"); // for successful receptions
    (non-collisions): 1 at the start of the reception, 0 at the end of the
    reception
    @signal[collision](type="long"); // the number of collided
    frames at the beginning of the collision period
    @signal[collisionLength](type="double"); // the length of the
    last collision period at the end of the collision period

    @statistic[receiveBegin](source="receiveBegin"; record=vector;
    interpolationmode=sample-hold; title="receive begin");
    @statistic[channelUtilization](source="timeavg(receive)";
    record=last; interpolationmode=linear; title="channel utilization");
    @statistic[collisionMultiplicity](source=collision;
    record=vector?, histogram; title="collision multiplicity");

@statistic[collisionLength](record=vector?, histogram, mean, sum, max;
title="collision length");
    @statistic[receivedFrames](source="sum(receive)"; record=last;
title="received frames");
    @statistic[collidedFrames](source="sum(collision)"; record=last;
title="collided frames");
    volatile double snr;
    gates:
    input in @directIn;
}

```

And finally the code of Host.ned is the following:

Host.ned

```

simple Host
{
    parameters:
        double txRate @unit(bps); // transmission rate
        double radioDelay @unit(s); // propagation delay of radio
link
        volatile int pkLenBits @unit(b); // packet length in bits
        volatile double iaTime @unit(s); // packet interarrival time
        double slotTime @unit(s); // zero means no slots (pure
Aloha)
        volatile double snr;
        @display("i=device/pc_s");
}

```

CSMA

Server.cc

```

#include "Server.h"
#include <math.h>

namespace aloha {
int bu=0;
double ba=0;
double x;
double ba2=0;
double rt=0;
double uee=0;
double la=2;
double Thr;
int Nee=4;
double mk;
double sik;

Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
    collisionLengthSignal = registerSignal("collisionLength");

    emit(receiveSignal, 0.0);
    emit(receiveBeginSignal, 0.0);

    if (ev.isGUI())
        getDisplayString().setTagArg("i2", 0, "x_off");
}

```



```

}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        cPacket *pkt = check_and_cast<cPacket *>(msg);

        ASSERT(pkt->isReceptionStart());
        simtime_t endReceptionTime = simTime() + pkt->getDuration();
        emit(receiveBeginSignal, ++receiveCounter);

        if (!channelBusy)
        {
            EV << "started receiving\n";
            recvStartTime = simTime();
            channelBusy = true;
            scheduleAt(endReceptionTime, endRxEvent);
        }
    }
}

```

```

        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_yellow");
            getDisplayString().setTagArg("t",0,"RECEIVE");
            getDisplayString().setTagArg("t",2,"#808000");
        }
    }
else
{
    EV << "another frame arrived while receiving --
collision!\n";

    if (currentCollisionNumFrames==0)
        currentCollisionNumFrames = 2;
    else
        currentCollisionNumFrames++;

    // Pe error probability in M=16 QAM constellation
    // ofdm simulation
    uee=par("snr"); // first read the snr of this current
interference user
    ba2=(pow((uee+ba2),2)); // and update ba, the
enrgy of the interferers
    mk=pow(ba,2) + ba2; // calculating SNR level, our energy over
interferers,  $\sigma^2 + \sigma_1^2$ 
    x=lgamma(0.5/mk);
    Thr=la*(pow(la,uee)/uee)*exp(-la)*(1-x); // estimation of
throughput
    EV << "Throughput: " << Thr<<endl;
    bu=bu+currentCollisionNumFrames;
    EV << "ba2: " << ba2<<endl; // on the screen for control
    EV << "collisionframes: " << bu<<endl;
    if (endReceptionTime > endRxEvent->getArrivalTime())
    {
        cancelEvent(endRxEvent);
        scheduleAt(endReceptionTime, endRxEvent);
    }

    // update network graphics
    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_red");
        getDisplayString().setTagArg("t",0,"COLLISION");
        getDisplayString().setTagArg("t",2,"#800000");
        char buf[32];
        sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
        bubble(buf);
    }
}
channelBusy = true;
delete pkt;
bu=0;
ba2=0;
}
}
}

```

```

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "ba: " << ba<<endl;
    EV << "Throughput: " << Thr<<endl;
    recordScalar("duration", simTime());
}

};

#include "Server.h"
#include <math.h>

namespace aloha {
int bu=0;
double ba=0;
double x;
double ba2=0;
double rt=0;
double uee=0;
double la=2;
double Thr;
double bi=0;
int Nee=4;
double mk;
double sik;

Define_Module(Server);

Server::Server()
{
    endRxEvent = NULL;
}

Server::~Server()
{
    cancelAndDelete(endRxEvent);
}

void Server::initialize()
{
    endRxEvent = new cMessage("end-reception");
    channelBusy = false;

    gate("in")->setDeliverOnReceptionStart(true);

    currentCollisionNumFrames = 0;
    receiveCounter = 0;
    WATCH(currentCollisionNumFrames);

    receiveBeginSignal = registerSignal("receiveBegin");
    receiveSignal = registerSignal("receive");
    collisionSignal = registerSignal("collision");
}

```

```

collisionLengthSignal = registerSignal("collisionLength");

emit(receiveSignal, 0.0);
emit(receiveBeginSignal, 0.0);

if (ev.isGUI())
    getDisplayString().setTagArg("i2",0,"x_off");
}

void Server::handleMessage(cMessage *msg)
{
    if (msg==endRxEvent)
    {
        EV << "reception finished\n";
        channelBusy = false;
        ba=par("snr"); // reads one time the parameter snr of the
original message received
        EV << "ba"<<ba;
        // update statistics
        simtime_t dt = simTime() - recvStartTime;
        if (currentCollisionNumFrames==0)
        {
            // start of reception at recvStartTime
            cTimestampedValue tmp(recvStartTime, 11);
            emit(receiveSignal, &tmp);
            // end of reception now
            emit(receiveSignal, 0);
        }
        else
        {
            // start of collision at recvStartTime
            cTimestampedValue tmp(recvStartTime,
currentCollisionNumFrames);
            emit(collisionSignal, &tmp);

            emit(collisionLengthSignal, dt);
        }

        currentCollisionNumFrames = 0;
        receiveCounter = 0;
        emit(receiveBeginSignal, receiveCounter);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i2",0,"x_off");
            getDisplayString().setTagArg("t",0,"");
        }
    }
    else
    {
        cPacket *pkt = check_and_cast<cPacket *>(msg);

        ASSERT(pkt->isReceptionStart());
        simtime_t endReceptionTime = simTime() + pkt->getDuration();
        emit(receiveBeginSignal, ++receiveCounter);
    }
}

```

```

if (!channelBusy)
{
    EV << "started receiving\n";
    recvStartTime = simTime();
    channelBusy = true;
    scheduleAt(endReceptionTime, endRxEvent);

    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_yellow");
        getDisplayString().setTagArg("t",0,"RECEIVE");
        getDisplayString().setTagArg("t",2,"#808000");
    }
}
else
{
    EV << "another frame arrived while receiving --
collision!\n";

    if (currentCollisionNumFrames==0)
        currentCollisionNumFrames = 2;
    else
        currentCollisionNumFrames++;

    // Pe error probability in M=16 QAM constellation
    // ofdm simulation
    uee=par("snr"); // first read the snr of this current
interference user
    ba2=(pow((uee+ba2),2)); // and update ba, the
energy of the interferers
    mk=pow(ba,2) + ba2; // calculating SNR level, our energy over
interferers,  $\sigma^2 + \sigma_1^2$ 
    x=lgamma(0.5/mk);
    Thr=la*(pow(la,uee)/ uee)*exp(-la)*(1-x); // estimation of
throughput
    //EV << "Throughput: " << Thr<<endl;
    EV << "Er.Prob: " << 1-x<<endl;
    bu=bu+currentCollisionNumFrames;
    EV << "ba2: " << ba2<<endl; // on the screen for control
    EV << "collisionframes: " << bu<<endl;
    if (endReceptionTime > endRxEvent->getArrivalTime())
    {
        cancelEvent(endRxEvent);
        scheduleAt(endReceptionTime, endRxEvent);
    }

    // update network graphics
    if (ev.isGUI())
    {
        getDisplayString().setTagArg("i2",0,"x_red");
        getDisplayString().setTagArg("t",0,"COLLISION");
        getDisplayString().setTagArg("t",2,"#800000");
        char buf[32];
        sprintf(buf, "Collision! (%ld frames)",
currentCollisionNumFrames);
        bubble(buf);

```

```

        }
    }
    channelBusy = true;
    EV << "TimeofDelivery: " << -recvStartTime +simTime() <<endl;
    delete pkt;
    bu=0;
    ba2=0;
}

void Server::finish()
{
    rt=erf(ba);
    EV << "duration: " << simTime()<<endl;
    EV << "bu: " << bu<<endl;
    EV << "ba: " << ba<<endl;
    EV << "Throughput: " << Thr<<endl;
    recordScalar("duration", simTime());
}

};

```

The code of Server.h is the following:

Server.h

```

#ifndef __ALOHA_SERVER_H__
#define __ALOHA_SERVER_H__

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha server; see NED file for more info.
 */
class Server : public cSimpleModule
{
public:
    double snr;
private:
    // state variables, event pointers
    bool channelBusy;
    cMessage *endRxEvent;
    long currentCollisionNumFrames;
    long receiveCounter;
    simtime_t recvStartTime;
    //double snr;
    // statistics
    simsignal_t receiveBeginSignal;
    simsignal_t receiveSignal;
    simsignal_t collisionLengthSignal;
    simsignal_t collisionSignal;

public:

```

```

    Server();
    virtual ~Server();

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

}; //namespace

#endif

```

The code of Host.cc is the following:

Host.cc

```

#include "Host.h"
namespace aloha {

Define_Module(Host);

Host::Host()
{
    endTxEvent = NULL;
}

Host::~Host()
{
    cancelAndDelete(endTxEvent);
}

void Host::initialize()
{
    server = simulation.getModuleByPath("server");
    if (!server) error("server not found");

    txRate = par("txRate");
    //
    snr = par("snr");
    radioDelay = par("radioDelay");
    iaTime = &par("iaTime");
    pkLenBits = &par("pkLenBits");

    slotTime = par("slotTime");
    isSlotted = slotTime>0;
    WATCH(slotTime);
    WATCH(isSlotted);

    endTxEvent = new cMessage("send/endTx");
}

```

```

state = IDLE;
pkCounter = 0;
WATCH((int&)state);
WATCH(pkCounter);

if (ev.isGUI())
    getDisplayString().setTagArg("t",2,"#808000");

scheduleAt(getNextTransmissionTime(), endTxEvent);
}

void Host::handleMessage(cMessage *msg)
{
    ASSERT(msg==endTxEvent);

    if (state==IDLE)
    {
        // generate packet and schedule timer when it ends
        char pkname[40];
        sprintf(pkname,"pk-%d-#%d", getId(), pkCounter++);
        EV << "generating packet " << pkname << endl;

        state = TRANSMIT;

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"yellow");
            getDisplayString().setTagArg("t",0,"TRANSMIT");
        }

        cPacket *pk = new cPacket(pkname);
        pk->setBitLength(pkLenBits->longValue());
        // manual pg 112
        snr=par("snr");
        simtime_t duration = pk->getBitLength() / txRate;
        sendDirect(pk, radioDelay , duration, server->gate("in"));

        scheduleAt(simTime()+duration, endTxEvent);
    }
    else if (state==TRANSMIT)
    {
        // endTxEvent indicates end of transmission
        state = IDLE;

        // schedule next sending
        scheduleAt(getNextTransmissionTime(), endTxEvent);

        // update network graphics
        if (ev.isGUI())
        {
            getDisplayString().setTagArg("i",1,"");
            getDisplayString().setTagArg("t",0,"");
        }
    }
}
else

```



```

    {
        error("invalid state");
    }
}

simtime_t Host::getNextTransmissionTime()
{
    simtime_t t = simTime()+iaTime->doubleValue();

    if (!isSlotted)
        return t;
    else
        // align time of next transmission to a slot boundary
        return slotTime * ceil(t/slotTime);
}
};

```

The code of Host.h is the following:

Host.h

```

#ifndef __ALOHA_HOST_H__
#define __ALOHA_HOST_H__

#include <omnetpp.h>

namespace aloha {

/**
 * Aloha host; see NED file for more info.
 */
class Host : public cSimpleModule
{double snr;
private:
    // parameters
    simtime_t radioDelay;
    double txRate;
    cPar *iaTime;
    cPar *pkLenBits;
    simtime_t slotTime;
    bool isSlotted;
    //

    // state variables, event pointers etc
    cModule *server;
    cMessage *endTxEvent;
    enum {IDLE=0, TRANSMIT=1} state;
    int pkCounter;

public:
    Host();
    virtual ~Host();
protected:

```

```

    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    simtime_t getNextTransmissionTime();

};

}; //namespace

#endif

```

The code of Aloha.ned is the following:

Aloha.ned

```

network Aloha
{
    parameters:
        int numHosts; // number of hosts
        double txRate @unit(bps); // transmission rate
        double slotTime @unit(ms); // zero means no slots (pure Aloha)
        volatile double snr;
        @display("bgi=background/terrain,s");
    submodules:
        server: Server
        {snr=snr;};

        host[numHosts]: Host {
            txRate = txRate;
            slotTime = slotTime;
            snr=snr;
        }
}

```

The code of Server.ned is the following:

Server.ned

```

simple Server
{
    parameters:
        @display("i=device/antennatower_1");
        @signal[receiveBegin] (type="long"); // increases with each new
frame arriving to the server and drops to 0 if the channel becomes
finally idle
        @signal[receive] (type="long"); // for successful receptions
(non-collisions): 1 at the start of the reception, 0 at the end of the
reception

```

```

    @signal[collision] (type="long"); // the number of collided
frames at the beginning of the collision period
    @signal[collisionLength] (type="double"); // the length of the
last collision period at the end of the collision period

    @statistic[receiveBegin] (source="receiveBegin"; record=vector;
interpolationmode=sample-hold; title="receive begin");
    @statistic[channelUtilization] (source="timeavg(receive)";
record=last; interpolationmode=linear; title="channel utilization");
    @statistic[collisionMultiplicity] (source=collision;
record=vector?, histogram; title="collision multiplicity");

@statistic[collisionLength] (record=vector?, histogram, mean, sum, max;
title="collision length");
    @statistic[receivedFrames] (source="sum(receive)"; record=last;
title="received frames");
    @statistic[collidedFrames] (source="sum(collision)"; record=last;
title="collided frames");
    volatile double snr;
    gates:
    input in @directIn;
}

```

And finally the code of Host.ned is the following:

Host.ned

```

simple Host
{
    parameters:
        double txRate @unit(bps); // transmission rate
        double radioDelay @unit(s); // propagation delay of radio

link
        volatile int pkLenBits @unit(b); // packet length in bits
        volatile double iaTime @unit(s); // packet interarrival time
        double slotTime @unit(s); // zero means no slots (pure

Aloha)
        volatile double snr;
        @display("i=device/pc_s");
}

```