Implementation of a distributed video sharing architecture for wireless networks with heterogeneous links

Υλοποίηση κατανεμημένης αρχιτεκτονικής διαμοιρασμού βίντεο σε ασύρματα τοπικά δίκτυα με ετερογενή χαρακτηριστικά διασυνδέσεων

Διπλωματική Εργασία
του ΔΟΥΛΗ Μιχαήλ

Επιβλέποντες
ΑΡΓΥΡΙΟΥ Αντώνιος
ΤΑΣΣΙΟΥΛΑΣ Λέανδρος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
Τμήμα Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών

2013

This page is intentionally left blank

Acknowledgments

I would like to thank my supervisor Prof. Antonios Argyriou who guided me with determination and patience, from the moment we started to discuss the topic of my Thesis up until the completion of the last page of this report. Furthermore, I am grateful to my co-supervisor Prof. Leandros Tassiulas and all the Department's professors for the invaluable knowledge that they offered to me throughout my studies.

This page is intentionally left blank

Περίληψη

Την περίοδο που συντάσσεται το παρόν, το Διαδίκτυο έχει εξελιχθεί σε ένα παράδεισο διαμοιρασμού περιεχομένου πολυμέσων. Παράλληλα, οι εξελίξεις στον τομέα υλικού (ολοκληρωμένα κυκλώματα, ευκρινείς οθόνες αφής μικρού μεγέθους κ.α.) και η ανάπτυξη εύχρηστου και αξιόπιστου λογισμικού για φορητές συσκευές οδήγησαν στη ραγδαία αύξηση της χρήσης των τοπικών ασύρματων δικτύων. Τα τοπικά αυτά δίκτυα φέρουν συνήθως ετερογενή χαρακτηριστικά, κυρίως ως προς την επεξεργαστική ισχύ των κόμβων τους και το *bandwidth* των επιμέρους συνδέσεων. Συνεπώς οι χρήστες του δικτύου έχουν διακριτές απαιτήσεις, οι οποίες όμως αφορούν στο ένα -κοινό- περιεχόμενο πολυμέσων. Σε αυτή την κατεύθυνση εργαστήκαμε για να υλοποιήσουμε μια κατανεμημένη αρχιτεκτονική διαμοιρασμού βίντεο για δίκτυα τέτοιου τύπου. Η αρχιτεκτονική που υλοποιήσαμε εκμεταλλεύεται τον υψηλό ρυθμό μετάδοσης συνδέσμων μεταξύ κόμβων, συνδεδεμένων σε τοπικά ασύρματα δίκτυα, ενώ παράλληλα εγγυάται την απόδοση του κλασσικού μοντέλου πελάτη-εξυπηρετητή. Παράλληλα μέσα από μια σειρά πειραμάτων αποδείξαμε πως η προσέγγυσή μας προσφέρει σημαντική βελτίωση στην απόδοση διαμοιρασμού βίντεο, εστιάζοντας στις παραμέτρους, εκείνες, που επηρεάζουν τα μέγιστα τη συμπεριφορά της υλοποιημένης αρχιτεκτονικής.

This page is intentionally left blank

Contents

Chapter 1

Introduction

The invention of the motion picture camera in 1888, allowed the modern man, for the first time in human history, to capture and store single pictures on a single reel. Since then, video has served as a medium for entertainment and communication [1]. Television broadcasting gave an extra boost in the popularity of video. Recorded shows or live events were made available to millions of people around the world through over-the-air broadcasts and cable signals.

At the end of the millennium, Internet and World Wide Web offered a new boost in the popularity of video. Initially, demands for text, pictures and document exchange were typical for the provided services which were available at the time, such as Web browsing and file transfer. Soon enough though, individuals started to experiment with the transfer of multimedia data– mainly sound and video– over the Internet. Furthermore, reinforced by technological advancements such as processing power, video compression techniques and higher networking speeds real-time multimedia transfer over the Internet gained significant attention during the last ten years.

Streaming is a technology that allows the user to transfer video over the net in real-time, without waiting for the entire file to be downloaded [2]. This technology is based on the idea that a video file can be separated into small parts. These parts are transmitted in succession to the end-user and then combined together again in order to playback the video, without waiting for all the parts to be delivered. Therefore, streaming offers nearly instantaneous playback of a video file in spite of it's size. This size undependability, plus the fact that end users do not spare any storage space for streaming a video, combined with the rise of broadband Internet connections resulted in a major boost in streaming popularity [3].

This popularity made video distribution to be the primary reason behind the major growth of data traffic over wireless networks. Moreover, an increase by two orders of

magnitude compared with the current volume of data is expected within the next four years [4].

Despite the advancements in wireless networks and wireless data services, this increased demand is likely to lead in a breakdown of cellular and wireless systems. This problem can be addressed by bringing the video content closer to the end users. This can be achieved by caching popular video files and then through localized communication exploit their proximity in order to improve video throughput [5]. The cooperation of coexisting heterogeneous wireless networks promises great potential, especially in our study case of localized communication, i.e video distribution via short-range links among the user terminals.

The architecture which we implemented for the purposes of this Thesis embodies many of the above characteristics. Our system exploits the short-range/high-rate connections between the mobile devices of a local WAN in order to stream video files. The system offers actually a generic framework for simulating networks with many terminals. At the same time it facilitates configurable heterogeneous links and is designed in a way to be expandable.

In the second Chapter we present in brief some general principles, rules, techniques and mechanisms regarding the basic knowledge that this Thesis is based on. In Chapter 3 we combine this knowledge and offer an overview of the distributed architecture, including a brief discussion about connection options, a basic example and the abstract form that we worked on, during the development of the Thesis. Chapter 4 accommodates all the details that we consider as critical for having a better understanding of the implementation of the system. Initially we present an overview of the simulation tool Omnet++. Furthermore we discuss the functionality of the different components of the system highlighting the critical parts.

The behavior of the implemented framework is tested in Chapter 5. We conduct a number of experiments aiming into validating the correct behavior of the system under certain circumstances. We, further, focus on some interesting parameters offering a broader view of the challenges that emerge. Finally in Chapter 6 we present our conclusions.

Chapter 2

Background Knowledge

In this Chapter we present the principles, rules methods and mechanisms of various schemes and systems, including both wireless networks and video streaming in order to introduce some basic knowledge that we used during the development of the implementation of the architecture.

In recent times, the expansion of wireless Local Area Networks(LAN) has been immense. University campuses, cafeterías, hospitals and hotels have all been hotspots for IEEE802.11 based wireless LANs. Furthermore, nowadays nearly all of the mobile devices come with a type of 802.11 network interface. At the same time, cellular wide access wireless networks have emerged and are gaining popularity. These heterogeneous wireless networks coexist, but have many differences regarding data rates, coverage, mobility and so on.

Local access wireless networks have, in general, high data rates and relatively small power consumption on their terminals, but support limited mobility and coverage. On the contrary, wide access networks have large coverage and mobility, while suffering from lower data rates and high power consumption. Therefore, these two heterogeneous wireless networks complement each other and could be both efficient in case they operate in a cooperative way [6].

2.1 Cooperative Networks

Cooperative networking is a scientific issue that has many different understandings. Cooperation can be distinguished into two major categories. Macro and Micro cooperation.

Macro cooperation involves wireless terminals, wireless routers and access points which all collaborate in order to exploit shorter radio propagation and increase the data rate.

In micro cooperation the cooperating units may be algorithms, processing units or functional parts. The gains of the micro cooperation can vary from increasing the bandwidth of the backbone link, improving the reliability of the wireless channel and lowering the power consumption of the terminal (end-user device).

Macro and micro cooperation are actually two approaches on the same issue of cooperative networking. These approaches can be finally integrated together.

2.1.1 Micro Cooperative Architecture & Scenarios

Fig. 1 and Fig. 2 demonstrate two typical architectures of the different cooperation categories that we discussed above. We are going to focus on the micro cooperative network architecture of Fig. 2. This version facilitates a number of terminals that are connected locally in a wireless network. At the same time they are connected individually with an access point/base station using a different wireless standard, for instance cellular reception. These terminals have to be multi-modality ones, i.e. they should be able to communicate simultaneously with the access point/base station and the peer terminals, with long and short range links respectively.

In this type of cooperative networks, the multi-modality terminals and the base station usually operate and communicate down to packet granularity. Therefore, the architecture of this network is actually very similar to the one of a centrally controlled p2p network. There are many scenarios regarding the operation of the micro cooperative networks which are discussed in [6]. One of them which is very interesting and useful for our work is that of the Unicast Transport/Multicast Servise. In this scenario the mobile terminals have multicast service and this service is transported through unicast links to the other terminals (peers).

The same team conducted a number of experiments by implementing cooperative network prototypes to illustrate the potential of such type of networks.

In particular, they assumed that a server hosts some digital content which can be accessed by mobile terminals over cellular interface. At the same time, the architecture offers the possibility to the mobile terminals to collaborate and exchange disjointed parts of the digital content over short-range links in case they manage to find a peer.

Two commercial Nokia N70 devices were used to illustrate this approach. The devices communicated to each other via bluetooth and were connected to the base station using GPRS. The base station provided two download possibilities to the mobile terminals. A stand-alone download, where the content was available in a single large file and a split file download, where the content was divided into two equally large

files. In the stand-alone download each terminal downloads the file in a given time T with data rate R. In the cooperative scenario though, where the terminals download a smaller file from the base station and then exchange the missing file with their neighbors the download time became nearly half (~T/2). This simple experiment demonstrated that **through cooperation higher data rates are achievable without increasing the complexity.**

2.2 Video Streaming in general

Video Streaming refers to the constant delivery of video content from a provider, to an end-user that instantly proceed to playback [7] . The client (end-user) can start to playback the media before the whole video file has been transmitted. Therefore, the end user can enjoy nearly instantaneous playback despite the size of the video.

The generic video streaming architecture can be divided into six areas; *media compression, application-layer QoS control, media distribution services, streaming servers, media synchronization at the receiver side,* and *streaming media protocols* [8]. In order for the video to be transmitted over a network it is necessary to be compressed into a digital stream with a reduced bit rate.

This task is performed by various applications known as encoders. Decoders decompress the compressed stream. These two types of applications combined together form *the codec*. MPEG-2, MPEG-4, H-264 (also known as AVC-Advanced Video Coding) are some of the well-known and most used codecs [9].

The coded video streams consist of successive pictures called inter- or intra-frames. Inter frames are frames that result from Inter frame prediction. Inter frame prediction tries to exploit the temporal redundancy between neighboring frames, thus leading to a higher compression rate. There are two types of Inter-frame, the B-Frames and the P-Frames. The main difference between the two types of frames is the reference frame they use for the Inter prediction. Intra-frames are frames that do not depend on any other –neighboring or not– frames of the video stream. They are known as I-Frames.

The Group of Pictures is a structured sequence of Intra- and Inter- frames. The typical GoP structure is IBBPBBP... . The Inter prediction dictates that among the frames of a Group of Pictures there are many dependencies. Fig. 3 illustrates these dependencies in a typical GoP [10].

Video streaming, also, involves various transport protocols that packetize the compressed bit-streams and send the packets to the clients that requested the given video file. Since video streaming is affected by transport delays and packet losses many network support mechanisms have emerged in order to cope with them. Among those mechanisms are the application level multicast and **content replication (caching) which improves the scalability** [8]**.**

During our work on this subject we took into consideration many of the above. Of course we also had to maintain the complexity of our project in a permissible limit. Hence, some simplifications were imposed.

2.3 Peer-To-Peer Video Streaming

In the introductory chapter we briefly reviewed the evolution of video streaming through the Internet. The World Wide Web can be viewed abstractly as a collection of millions of servers and clients that form a distributed system in order to provide access to associated documents and files.

The client-server model that was initially employed in the WWW proved to be inadequate. This was due to the fact that resources are usually concentrated in a small number of nodes, while in order to provide continuous service and permissible responsiveness, complex fault-tolerant and load-balance algorithms have to be employed. Additionally the limitations set on network bandwidth worsened the situation even more. These two important issues lead the researchers to explore alternative models and schemes which allocate processing load and network bandwidth among all the nodes that form a distributed system [11].

P2p networking exploits the computer power, the network connectivity and the storage in a way that allows users to leverage their collective power to the benefit of all. The nodes of a p2p network have equal roles thus they are often called *peers.* The peers of a p2p network *cooperate* in a *distributed* manner in order to achieve the desired objective.

One of the key features of a p2p system is that every peer contributes resources such as processing power, storage and of course bandwidth. Therefore, the overall system capacity increases as more nodes/peers join the system, making it's scalability significantly better than the one of the client-server model.

Those intrinsic characteristics make the P2P model a potential candidate to solve various problems in multimedia streaming over the Internet [12]. A peer can act simultaneously as a client and as a server. This allows a peer who is downloading a video file to upload it to other peers at the same time. Therefore according to [13] p2p streaming decreases considerably the bandwidth needs of the source. **The objective of this type of streaming is to maximize the video quality on individual peers despite the bandwidth heterogeneity and irregularity of the links among them. This objective then is closely related with our work.** In our implementation we have included some features and functionality from p2p streaming systems such as Chainsaw [14].

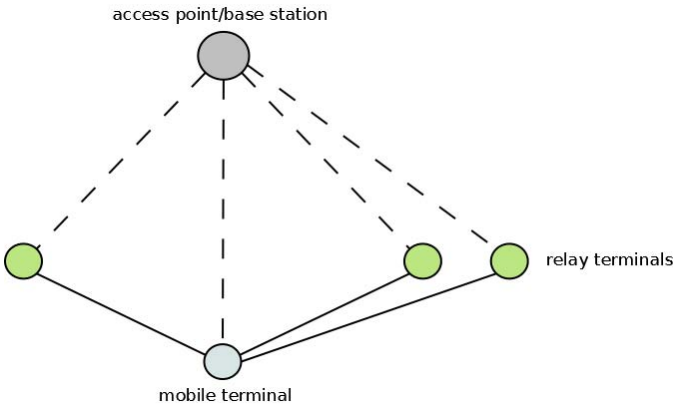

Figure 1. Macro cooperative architecture

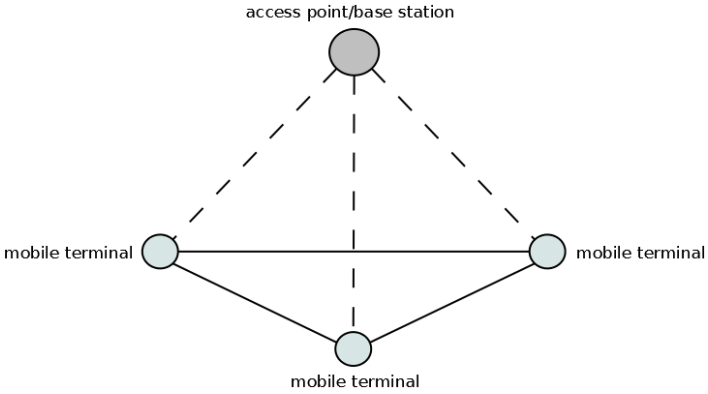

Figure 2. Micro cooperative architecture

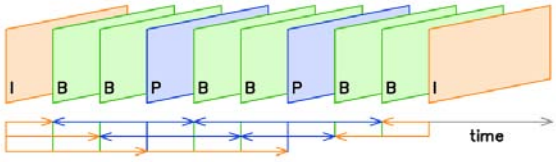

Figure 3. GoP structure [10]

This page is intentionally left blank

Chapter 3

Architecture Overview

In this section we are going to discuss the basic features of the distributed video sharing architecture that we propose.

Our architecture aims into fulfilling the requirements of video streaming, i.e. high data rates and low latency, by exploiting the advantages offered by new schemes of networking and communication such as micro cooperative networks and p2p systems. By combining various elements, methods and disciplines of these schemes we propose and implement a distributed video sharing architecture for wireless networks with heterogeneous links.

In this chapter we are going to study a generic architecture of a wireless LAN consisting of a number of end-user devices such as laptops, smartphones and tablets. Moreover, we try to analyze the possible connections of the devices, examine a specific architecture and it's abstracted version on which our implementation is developed and finally present details regarding our algorithms and techniques used in it.

3.1 Connection Options, Channels and Data Rates

As we have previously seen from the literature regarding the evolution of heterogeneous wireless networks and cooperative networks the options of connecting the mobile devices with each other and the access point/base station are numerous. Cellular reception, bluetooth and Wi-Fi are some of the standards that have been tested from researchers. Table 1 shows some connection options and features of them.

As we discussed in previous chapters, IEEE802.11 is the most popular standard for wireless communications. This connection option forms a basic framework that we can

work on, which at the same time is plausible and based on contemporary popular wireless standards. The devices that form the wireless network in our initial framework are connected to an access point via an IEEE802.11.b connection. Furthermore, they are connected to each other via IEEE802.11.a.

So far we have discussed the wireless part of our architecture. A part of it though is wired. In this part resides the video server which is connected to the access point via the Internet. This connection is considered as a standard DSL Internet connection. More connection options and details will be discussed in Chapter 5.

802.11	Frequency(GHz)	Max. Data Rate (Mbit/s)	Outdoor Range(m)
a	5	54	120
b	2.4	11	140
g	2.4	54	140
n	2.4/5	72.2	250

Table 1

3.2 Basic Framework

The architecture is based on a framework consisting of a local wireless network and a remote video server. A number of mobile devices such as laptops, tablets and smartphones may be connected to the wireless network. Fig. 4 illustrates the framework we had in mind when we started working on the generic architecture of this project.

The particular wireless network of the figure consists of 4 devices connected on an access point (blue links). The devices of this particular figure are two laptops, one tablet PC and one smartphone. The access point connects over the Internet with a video server. The dotted line shows the wired and the wireless segment of our architecture.

3.2.1 Principles, Methods and Algorithms

The contemporary model of video transfer is that of the client-server one. Fig. 5 demonstrates such a transfer taking place into the familiar system that we are discussing. Particularly, the user of the smartphone requests a video from the video server. A connection between the smartphone and the server is established and then the server starts to stream the video file. We notice that the gray rectangles that resemble the packets are indexed in a way to illustrate that they are streamed *in order.* The

packets travel from the server through the Internet and then through the wireless link (blue line).

The key difference between this model and the one of the proposed architecture is that *user devices may request and –in case they are available– receive data/packets from other devices of the local WAN.*

Fig. 6 illustrates a system that allows a node of the local wireless network which is already receiving data/packets to share them with another node of the local network. This is a core function of our system. The packets are transmitted via a wireless link (red line) which connects the smartphone *directly* with the laptop and has no relevance to the standard WiFi connection between the laptop and the access point (blue line). This feature is based on the micro cooperative architecture that we discussed in Chapter 2.

By combining the two previous approaches we have a distributed system which receives data from both the Internet (video server) and peers that have them available. Fig. 7 shows a distributed system that operates in the way we described. In particular, the laptop user requests a video file from the smartphone. The first frames/packets of the video stream to the laptop (orange rectangles with id's 0 and 1). In this scenario some frames are missing from the smartphone. This can occur for a number of reasons. For instance, these frames may part of the enhancement layer that is not necessary for a low quality version of the video that the smartphone user requested, or they could be missing due to packet loss. The missing frames then can be transferred from the video server to the laptop through the other wireless link (blue line) and be combined for a high quality version of the video.

Our design gives priority on the transfer of data from the peers at the local WAN. This is a core feature due to the fact that with this system we try to exploit the *proximity* of the data residing in nodes that are connected via *short-range/high-bandwidth* links.

Moreover, a node of the local WAN is, also, able to request data from the video server. This characteristic ensures that each mobile device/node will definitely receive video frames at some time -assuming that the channel connecting the node with the access point does not drop any packets. Additionally, this feature is absolutely essential because initially we assume that none of the nodes has any video frames, thus the first node requests the video stream from the video server.

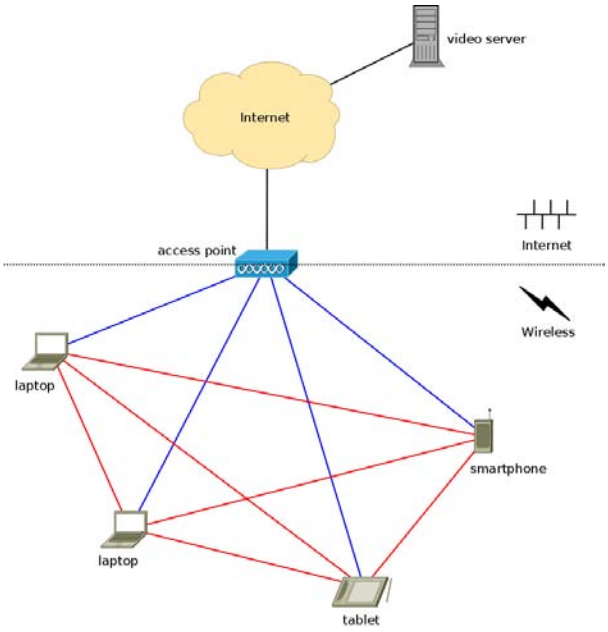

Figure 4. Architecture with 4 mobile devices

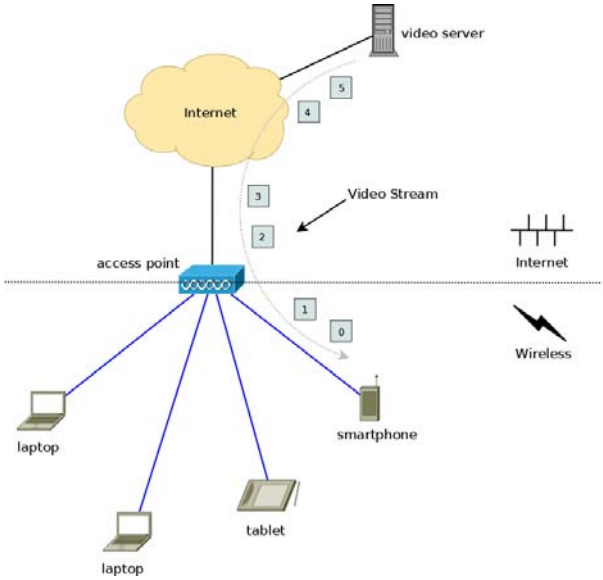

Figure 5. Standalone streaming according to the client-server model

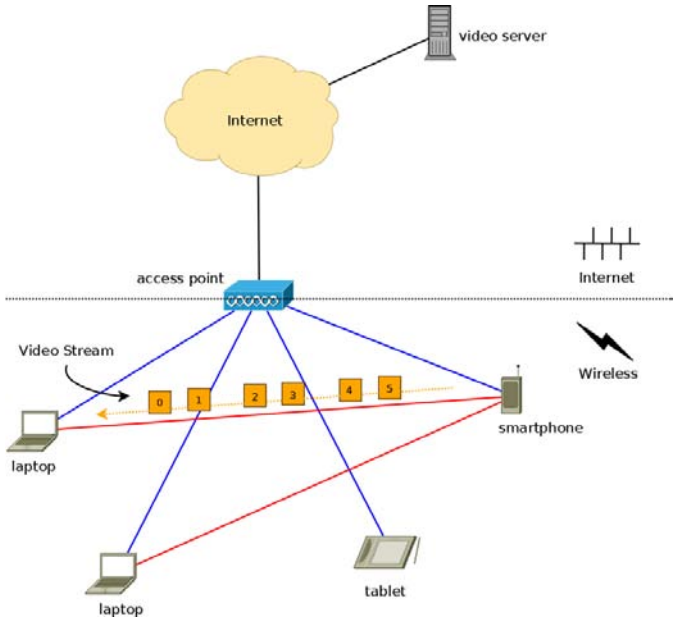

Figure 6. Streaming from a peer

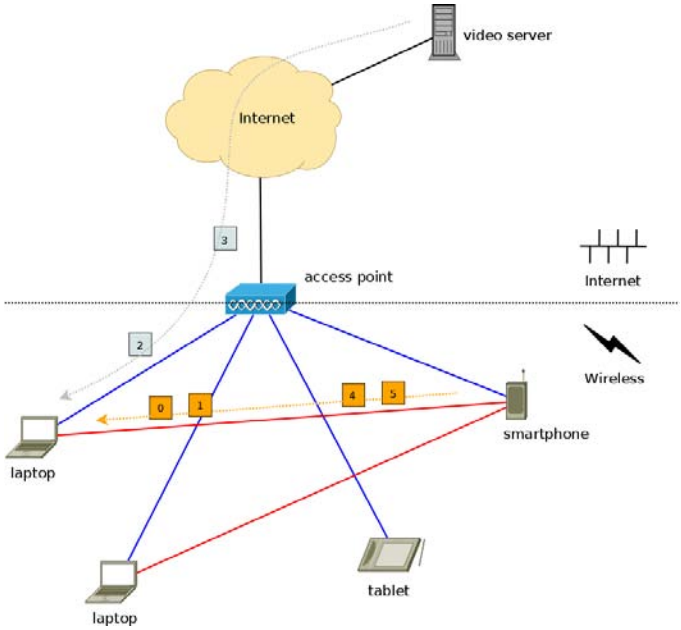

Figure 7. Distributed streaming

3.2.2 Abstract form of the Architecture

Although the framework presented in Fig. 4 is very intuitive and offers a fair understanding of the components and connections of our architecture, we ultimately had to come up with a more abstract version of it.

This version reduces and factors out many details so that we can focus on the features of the framework that we are interested in. Fig. 8 illustrates the abstract form of the architecture of Fig. 4 .

Initially, we notice that there is no distinction among the mobile devices, thus they are labeled simply as *terminals*. The *access point* remains as a component. It's role is auxiliary and is limited on an elementary level of routing between the terminals and the *base station.* An individual could argue that in the abstract form of the architecture that we examine, the access point could be eliminated without any obvious consequences regarding the coherence of our system, but the module itself serves a structural necessity. It helps us to facilitate two types of heterogeneous links, one representing the wireless links that connect to the terminals and the other representing the connection to the Internet the base station. The latter is actually the video server that we have seen in previous illustrations. We named this component in a way that reflects it's generic overseeing role in the whole system. In essence the base station embodies a fraction of the operations of the video server, plus some new functionality regarding the distributed nature of our system.

Finally, the abstract form of the system demonstrates three types of links. The blue and the red ones are wireless links, but have different colors due to the fact that we regard them as heterogeneous, i.e. links with different data rates, delay and packet drop rate. The black link represents the connection of the local WAN to the Internet and thus the base station. Again, we are going to present more details regarding the links in the next Chapter.

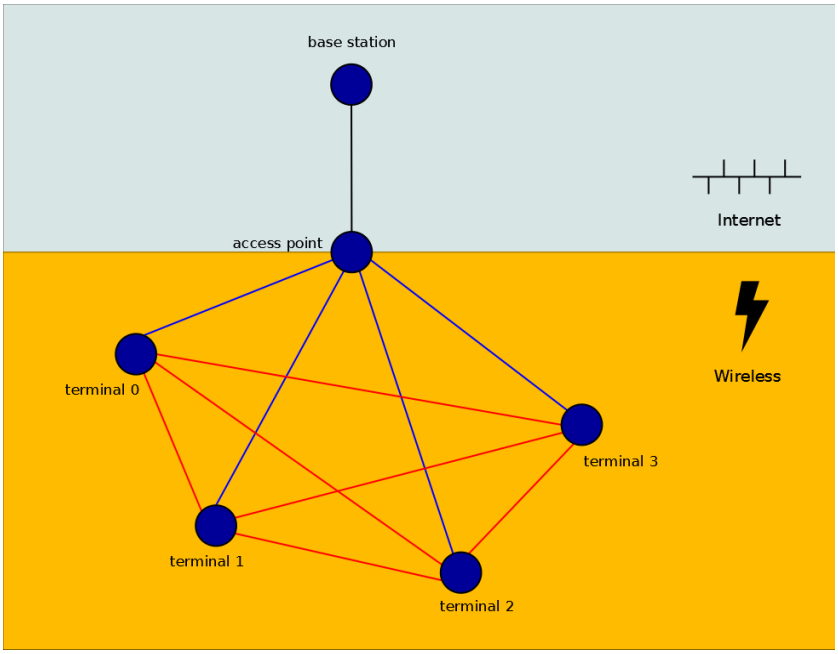

Figure 8. Abstract form of the architecture

This page is intentionally left blank

Chapter 4

Implementation

In order to verify the feasibility of our distributed video sharing architecture we used the OMNET++ network simulator to design, simulate and define our system's features in detail.

Initially, in this Chapter we present an overview of the tool we used to implement our system, the Omnet++ network simulator. Then, we discuss the key features of our system in detail. Subsequently, we examine various details regarding the structure and organization of our implementation as well as the functionality of each module and the behavior of the whole system in some intriguing scenarios.

4.1 An Overview of OMNET++ Network Simulator

Omnet++ is described from it's official site as "an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. 'Network' is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queuing networks, and so on" [15].

Omnet++ is a network simulator that initiated by András Varga in Technical University of Budapest, Department of Telecommunications (BME-HIT). It is developed in C++ and it is free to use for Academic and non-profit projects. All these features made Omnet++ an ideal tool for academic research, boosting it's popularity in the global scientific community.

Omnet++ is not a simulator itself. Instead, it offers a set of tools that form an infrastructure for simulating networks. The most fundamental ingredient of Omnet++ is it's component architecture. Simulation models are assembled from components

labeled as *modules.* These modules can be written in a way to be reusable, thus forming a generic and flexible architecture. Modules have parameters that can customize their behavior. The modules of the lowest hierarchy level are called simple modules. Simple modules are written in C++ and make use of the simulation library [15].

Along with the modularity and the flexibility of the simulation components Omnet++ offers great scalability. Modules can be connected to each other through gates and combine to form *compount modules*. The number of hierarchy levels is unlimited. Model structure is described in Omnet++'s NED language.

Modules communicate with each other through a message passing mechanism. Messages can carry arbitrary data structures and can be delivered directly or through a predefined path to their destination module, making Omnet++ ideal for modeling both wire and wireless communication systems. Fig. 9 illustrates an abstract layout of simple and compount modules, connections and gates.

The timing simulation is very simple. The "local simulation time" of a module advances only when a message is received. The message can arrive either from another module or from the receiving module itself through the *selfmessage* feature –mainly used to implement timers.

Gates are the main communication interfaces of the modules. Incoming messages are delivered on the input gates, while outcoming messages depart from the output gates. A *connection* links two gates. Additionally, *channels* are C++ classes, which encapsulate parameters and behavior regarding connections, such as data rate, propagation delay and packet error rate. These parameters are very useful in case a physical link is to be modeled.

The user defines the structure of the model in NED language descriptions (Network Description). NED lets the user declare simple modules, and connect and assemble them into compound modules.

Finally, Omnet++ comes with a build-in mechanism to record simulation results and collect statistics.

All these features, along with the GUI for simulation execution and the IDE based on the Eclipse platform, conveyed us in choosing the Omnet++ as the most suitable simulator for implementing our architecture.

4.2 Modeling the System's Components

4.2.1 Network Description – Structure and Organization

In Chapter 3 we discussed an abstract form of the architecture which lets us focus on the important features of it, while reducing any abundant details. Our implementation is

based upon this abstract form, regarding both the system's structure and the functionality of each module.

The system's structure and organization is described in a NED file. As we can observe from the abstract form of the architecture three basic components stand out and therefore we model these components in our implementation as three simple modules. The NED file then includes the declaration of three simple modules, the *terminal,* the *base station* and the *access point*.

The terminals have one gate for connection to the access point and N-1 gates for connection to the other terminals in the local WAN –where N is the total number of terminals in the WAN. The base station has only one input/output gate that connects it to the access point. The access point has the most gates among the three simple modules because it's routing role places it in the center of our system, functioning as a "bridge" between the Internet and the local wireless network. The access point accommodates one gate for connection to the base station and N other gates for connecting with all the terminals.

The NED file also contains the definition of the channels that connect the modules of our system. If we go back again to the abstract picture of our system we notice that there are three colors of lines, the red, the blue and the black ones. Therefore we modeled three channels, each with different characteristics. All three of these channels are extended from the *ned.DatarateChannel* class provided by Omnet++'s simulation library. Fig. 10 demonstrates the names of the particular channels and the parameters that they inherit from the DatarateChannel class. *WirelessA* and *WirelessB* serve as wireless links between the access point and the terminals and among the terminals themselves respectively. We can define each channel's parameters separately and model the heterogeneous links of our system with precision and flexibility. *ap2Internet* is the channel that models the Internet connection. Again we can modify the parameters accordingly in order to experiment with different data rates and delays. Fig. 11 shows the GUI output of the network overview of our system for a local WAN of three terminals.

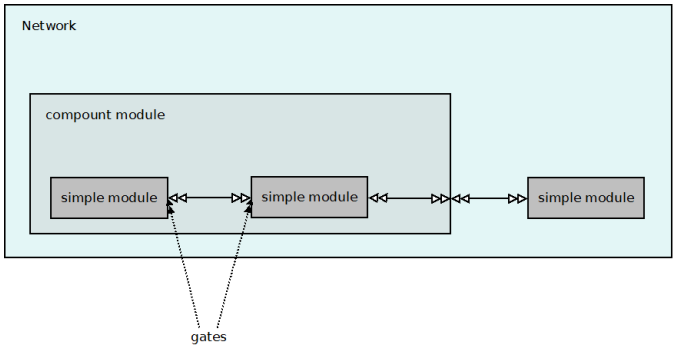

Figure 9. Simulation modules layout

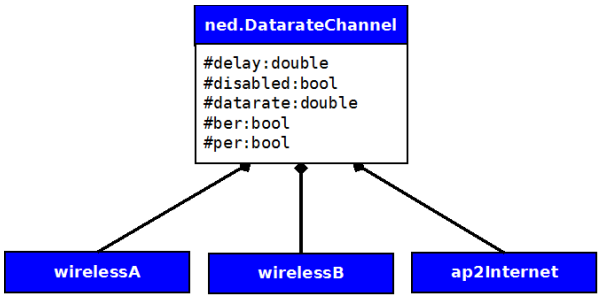

Figure 10. Channel sublassing

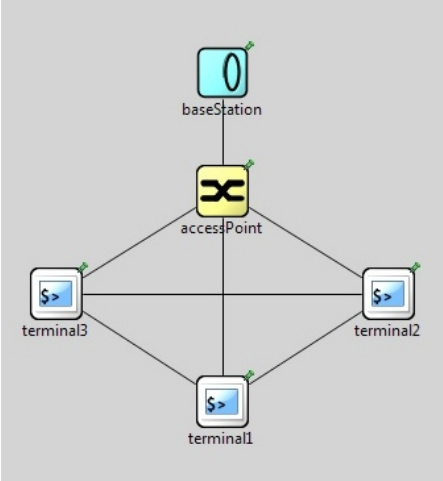

Figure 11. Network topology for 3 terminals

4.2.2 Simulation Modules – Parameters and Functionality

After finishing the design of the network structure in the NED file we have to implement the functionality of each module. This is done by writing the code in a separate C++ file, which is later linked with the NED and the simulation files (.ned and .ini respectively).

Simple modules are represented by a C++ class which is subclassed from the *cSimpleModule* class provided by the simulation library. The modules then are registered by the ***Define_Module()*** macro. In every module we redefine two methods from the *cSimpleModule* class; ***initialize()*** which is invoked from the simulation kernel just once at the beginning of every simulation, and ***handleMessage()*** which is invoked whenever a message arrives at the module. In our implementation we had to add some extra methods in every module. These methods are auxiliary and mainly generate or forward the various messages of our simulation. Fig. 12 illustrates the modules and the extra methods or parameters that each of them uses.

As we mentioned in the overview of Omnet++, modules communicate through messages. The messages are represented as objects of the *cMessage* class or it's subclasses. Messages that are sent or scheduled (*selfmessages*) are held by the simulation kernel in a list which maintains their order until their time comes to be delivered to the modules via the ***handleMessage()*** method. When a message arrives in a module the programmer is responsible for writing the code accordingly in order to establish the behavior of the module. Therefore, the main functionality of our modules is implemented in the redefined ***handleMessage()*** method. Moreover, for our implementation we subclassed and created a number of new message classes due to the varying amount and types of data that we had to attach to them. They are depicted in Fig. 13 where we can also view each message's parameters.

Now we are going to study in detail the implementation of each module. Let us begin with the module that embodies the core –and arguably the most complex– functionality, the terminal.

Each terminal maintains a struct that stores information about:

- the number of video files that the terminal requests.
- the number of video frames of each video file that are stored and ready for playback.
- the exact frames that have already been requested, either from another terminal or from the base station. This is very useful in order to avoid double requests of the same frame.
- the availability of *every* video frame in *every* terminal.

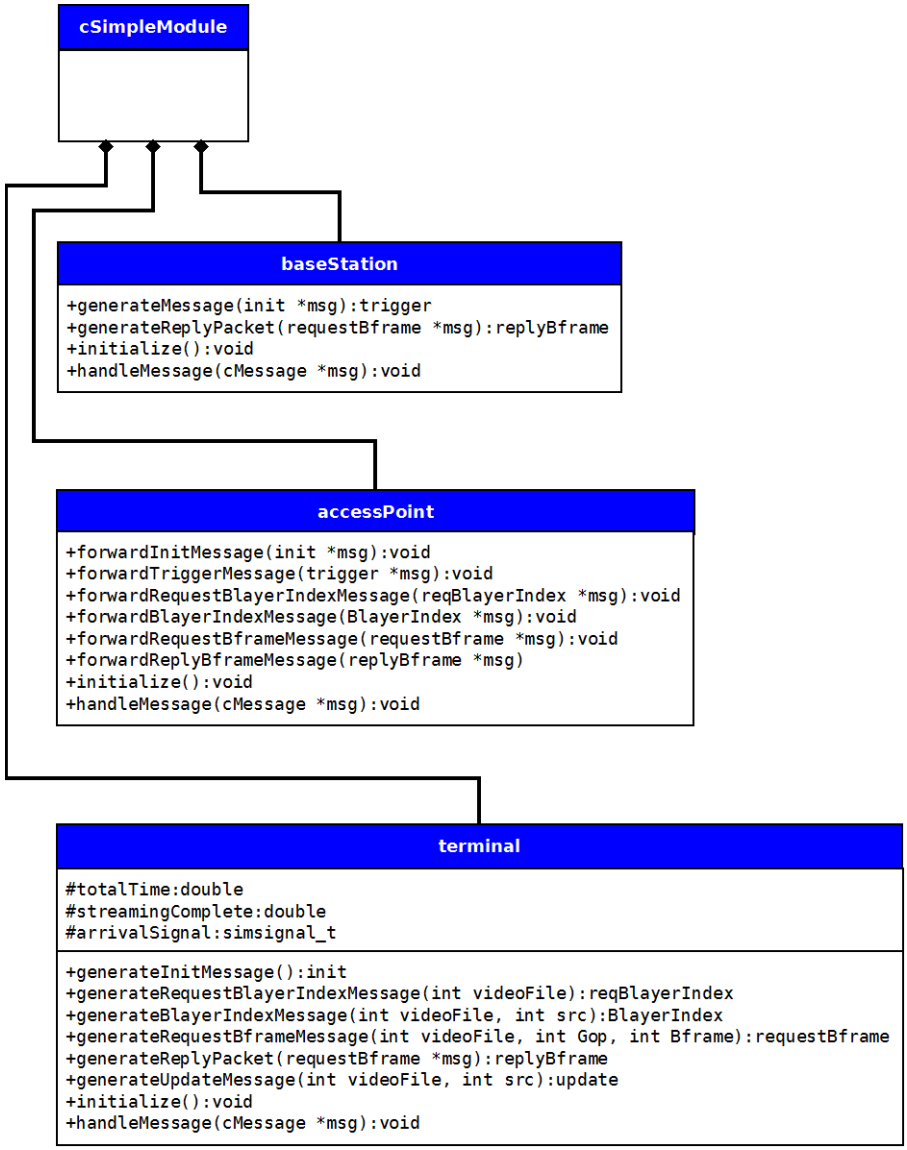

Figure 12. Simulation modules subclassing

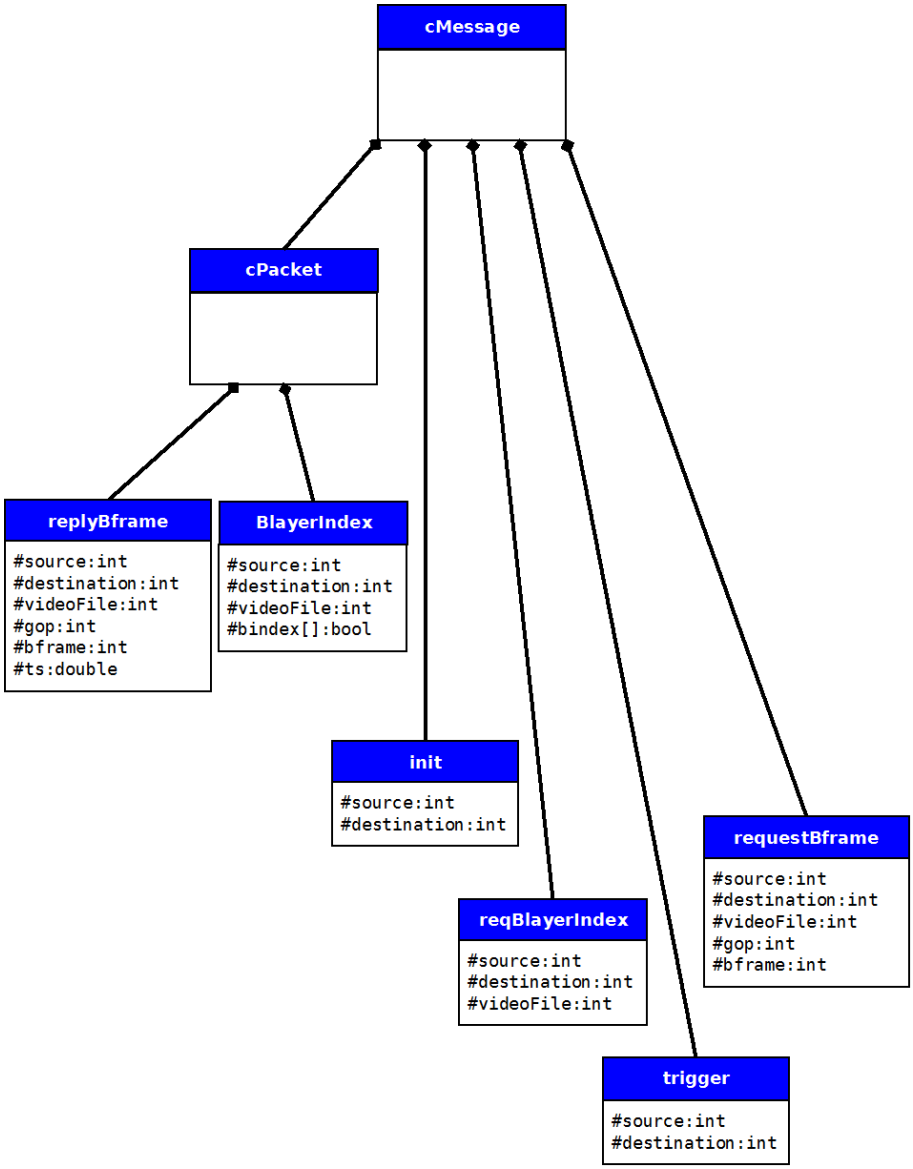

Figure 13. Message subclassing and the extra parameters we used for our implementation

For the implementation we consider that every video file is divided into groups of pictures, called ***GoP***s. Every GoP consists of a number of ***bframes***. This assumption is not 100% accurate regarding the video streaming and coding we discussed in Chapter 2. However, it offers a simple modeling, close to the actual layered video coding. The code is written in a way that both the number of GoPs and bframes can be defined easily, offering a flexible base for experimentation.

The behavior of the terminal module depends on what message it receives. When a message arrives, the terminal updates it's struct with the data attached to the message and depending on the situation it sends new messages. Fig. 14 demonstrates the functionality of the module as a state machine. Now we are going to study each state.

Idle. This is the initial state of each module after the invoking of the ***initialize()*** method. In this method we initialize the struct of each terminal, where we can implicitly set which terminals we want to act as seeders. Also we can set explicitly when each terminal will start to request a video file. This happens by sending an "*initialize*" message to the base station.

When a terminal receives a "*trigger*" message from the base station it moves to the next state. The initialization process and the message exchange between the terminals and the base station is depicted in Fig. 15.

State 1. As we described earlier each terminal requests the video stream from other terminals first. Therefore, it is necessary to know which terminals have the content available. This is established by exchanging "*index*" messages. An index message is a message that carries an index of a video file. The index is a boolean array which indicates what bframes a certain terminal has stored.

In this state if the terminal is not a seeder it requests via a "*REQ BlayerIndex*" message an index from all the other terminals.

When a terminal receives a "*REQ BlayerIndex*" message from any other terminal it moves to the next state.

State 2. In this state a terminal generates a "*RPL BlayerIndex*" message that attaches the index of the requested video file. Fig. 16 shows an index for a video file with 3 GoPs, each one consisting of 5 bframes. The terminal that generated this particular index has only the first GoP of the video available. The message then is sent directly to the interested terminal.

When a terminal receives a "*RPL BlayerIndex*" message it moves to the next state.

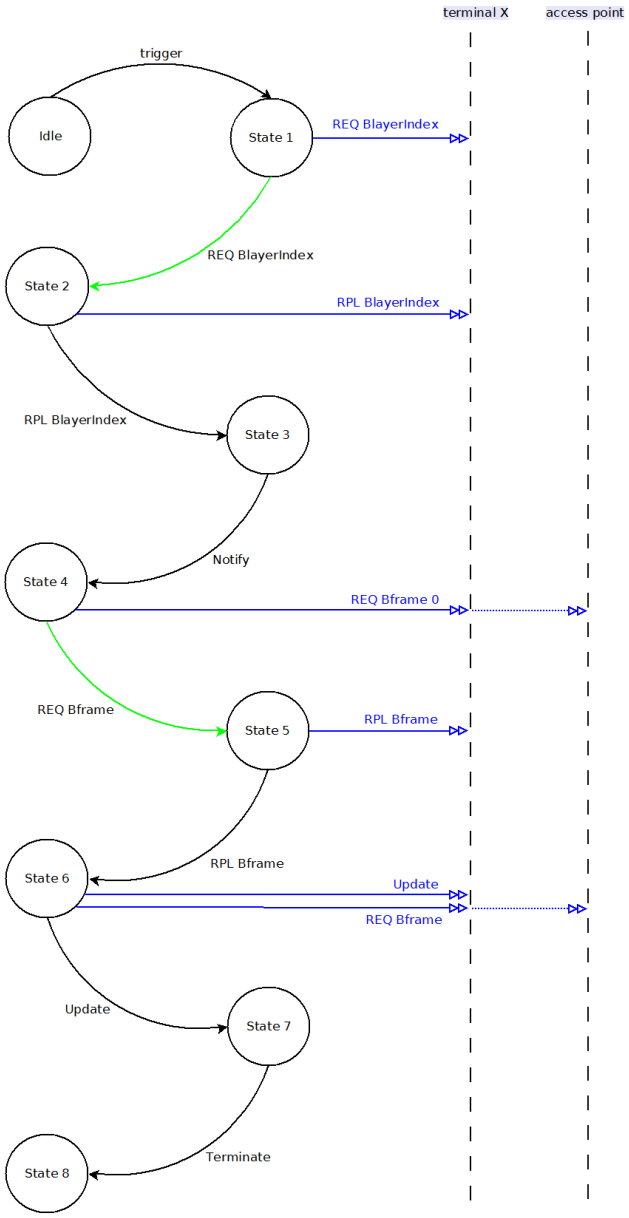

Figure 14. Terminal state machine

State 3. In our implementation a terminal retrieves the information from the "*RPL BlayerIndex*" message it receives and then updates the ***available2request*** field of its struct. Available2request is a two-dimension array that stores information regarding the availability of GoPs on each terminal. In general if a GoP is marked as available on a terminal then the streaming of the frames of this GoP may begin. Table 2 illustrates a particular instance of the available2request field of terminal 1. Please notice that the line which corresponds to the terminal's id (1) is highlighted with blue color. Also, notice that all the GoPs of this line are marked as non-available to request from this terminal. In our implementation we assume that the line that corresponds to the terminal itself, has by default marked all the GoPs with 0. In the instance of Table 2 if terminal 1 needs to request GoP 1, it could choose between terminal 2 and 3. If it needs to request GoP 4, though, it will scan the respective column and after finding all the entries as zeros will eventually send a request message to the base station.

The terminal continues to the next state only when all the peers of the local WAN has sent an Index reply. We implement this by scheduling a *"Notify"* self-message. Fig. 17 demonstrates the sequence of messages that the terminals exchange in order to obtain the video Indexes.

		GoP				
		0	**1**	**2**	**3**	**4**
Terminal	**0**	*0*	*0*	*0*	*0*	*0*
	1	*0*	*0*	*0*	*0*	*0*
	2	*1*	*1*	*1*	*1*	*0*
	3	*1*	*1*	*0*	*0*	*0*

Table 2

State 4. When a terminal is in this state it is certain that it has all the necessary information in order to start requesting bframes. It is worth noticing here that GoPs and bframes are requested, either from a peer or the base station, sequentially. In this state we handle the first frame of the first GoP of the video file. The terminal checks if the GoP is available in the peers through a simple iteration of them. Subsequently, it sends a "*REQ Bframe*" message to the first peer that has the first GoP available. If no peers have this GoP in their cache the terminal sends the message to the base station. A pseudo-code that summarizes the above is given in Algorithm 1.

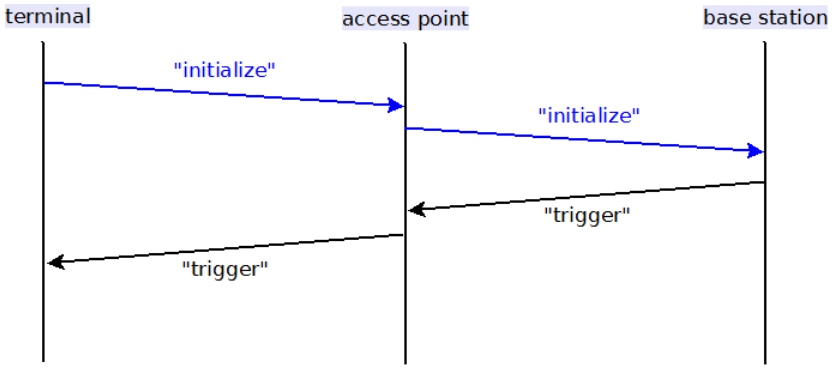

Figure 15. Initialization sequence

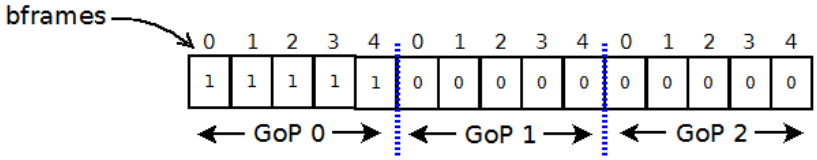

Figure 16. Index structure

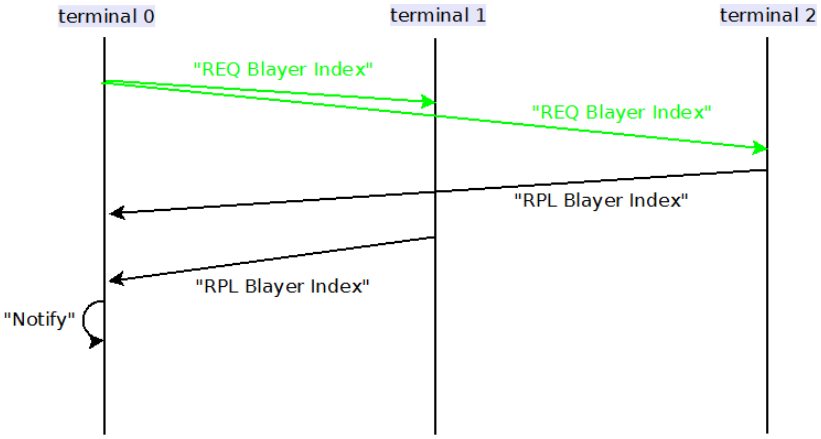

Figure 17. Index exchange sequence

Algorithm 1 Request the first bframe of the first GoP of a video file

Input: number of terminals in the local WAN, *terminals*

```
for each t in terminals do
    if available2request[t][0] then
        "REQ Bframe" ← generateRequestBframeMessage(0)
        sendMessage("REQ Bframe", t)
    else
        "REQ Bframe" ← generateRequestBframeMessage(0)
        sendMessage("REQ Bframe", base station)
    endif
endfor
```

State 5. In case the terminal receives a request message for a frame it generates a reply message which is subclassed from the *cPacket* library class. This message has extra parameters that are essential for the purposes of our implementation. Particularly, we have the ability to set the length of the message in bytes or bits, thus the links with different data rates can be simulated properly.

The terminal's function in this state is rather straightforward. It simply sends the generated message ("*RPL Bframe*") back to the sender.

State 6. When a reply message arrives at the terminal a sequence of checks takes place in order to establish the correct behavior of the system. Our implementation has some dynamic elements that enhance the performance of the architecture exploiting it's distributed and real-time characteristics.

The system embodies an update mechanism which is deployed when a terminal reaches a certain amount of cached GoPs. We call this amount the *Window of Availability* (WoA). If the GoPs that the terminal has received cover the WoA, the terminal sends an "*Update*" message to all the other terminals in order to inform them that it has bframes available for streaming. Algorithm 2 shows a pseudo-code that summarizes all the above.

In our implementation a terminal requests a bframe only when it has received the previous one. An alternative approach would be for the terminals (and the base station) to exchange acknowledgment messages. However, this would increase the number of exchanging messages, thus the congestion on our network.

Every time a terminal receives a "*RPL Bframe"* message it checks to verify whether the Bframe is the last one of the current GoP. If the Bframe concludes the GoP, we proceed to the next GoP and check if this is the last Gop of the video file. In that case the terminal stores the simulation time in a local variable, which will help us with the statistics collection. Furthermore, we schedule a self-message (*"terminate*") which will

make the transition of the terminal to the final State 8. If the current GoP is not complete yet, we choose the next bframe generate a new request message. In order to establish from where to send the new message, the terminal checks it's **available2request** struct. If the GoP that accommodates the Bframe is not cached in any of the peers, the terminal requests it from the base station. Otherwise it selects randomly a peer and forwards the newly generated "*REQ Bframe*" message to that one. Algorithm 3 summarizes in pseudo-code the above procedure, from the moment a reply message arrives at the terminal, up until the moment the terminal requests the next bframe. An interesting scenario that involves an update is depicted in Fig. 18.

Algorithm 2 Send Update Messages

Input: number of terminals in the local WAN, *terminals &*
window of availability, *WoA &*
counter for delivered GoPs to the terminal, *accumulatedGoPs.*

receiveMessage("*RPL Bframe"*)
accumulatedGoPs←*accumulatedGoPs+1*

if *accumulatedGoPs=WoA* **then**
accumulatedGoPs← 0
"Update" ← generateUpdateMessage*()*
for each *t* in *terminals* **do**
send*("Update", t)*
endfor
endif

Algorithm 3 Request next bframe

Input: total number of Groups of Pictures of the video file, *NoGoPs &*
number of terminals in the local WAN, *terminals &*
the id of the current GoP extracted by the received "*RPL Bframe"* message, *gop*

receiveMessage("*RPL Bframe"*)
A ← currentGoPHasMissingFrames(*gop*)
if A **then**
nextBframe ← findFirstMissingBframeOfGoP(*gop*)
else
gop ← *gop+1*
endif

```
if gop != NoGoPs then
    for each t in terminals do
        if available2request[t][gop] then
            gopNotCachedInTerminal ← False
        else
            gopNotCachedInTerminal ← True
        endif
    endfor
    if gopNotCachedInTerminal then
        "REQ Bframe" ← generateRequestBframeMessage(nextBframe)
        sendMessage("REQ Bframe", base station)
    else
        destination ← rand()%terminals
        while ((destination == self)&& (!(available2request[destination][gop])))
            destination ← rand()%terminals
        endwhile
        "REQ Bframe" ← generateRequestBframeMessage(nextBframe)
        sendMessage("REQ Bframe", terminal)
    endif
else
    finished ← getSimulationTime()
    self-message("Terminate")
endif
```

State 7. This is the state of a terminal that has just received an "*Update*" message. The message carries an index which is exactly the same as the index that the peers exchange after the initialization process. The receiver terminal updates the ***available2request*** field that is associated with the sender. The process is the same as in State 3.

State 8. In this state the terminal just checks whether all the other terminals have finished the streaming and if it is so, it ends the simulation by invoking the **endSimulation()** method.

The next module that we are going to study is the base station. As we can assume simply by observing the Fig. 19 that illustrates the states of the module and the transitions among them, this module is less complex. The base station module has also a struct that acts like a local database, useful for storing information regarding the video files. Here, we only had to maintain data regarding the structure of the video files, particularly the number of GoPs and bframes per video file.

In a matter of fact, we could say that although the name of this module implies a key role to the architecture, in reality it acts like a slave, i.e. this module does not iniate

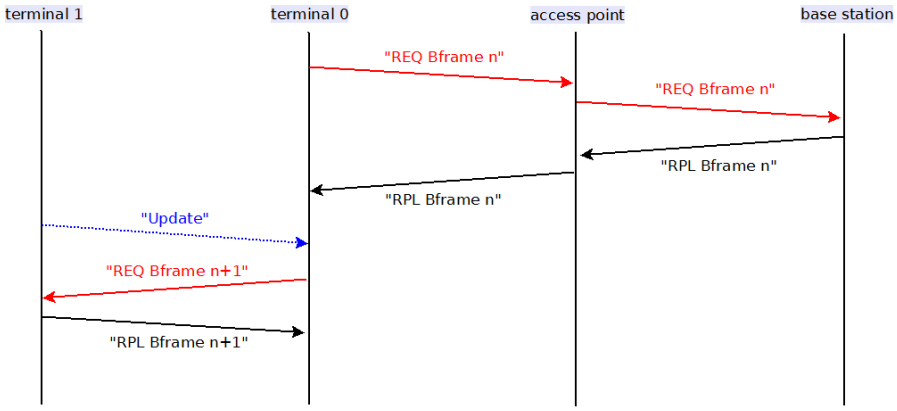

Figure 18. Update scenario sequence

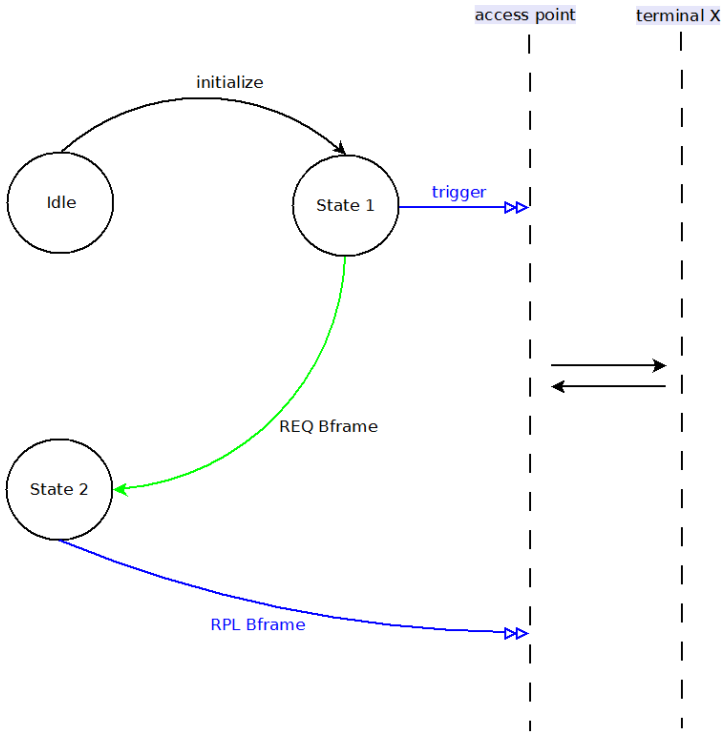

Figure 19. Base station state machine

any action in our system. It's behavior is limited in initiating a video stream and sending reply messages to the interested terminals. Therefore, the functionality is rather limited. The following three states summarize it for a better understanding.

Idle. In this state the base station has just finished the initialization process. This process involves the video files that this module has in memory. The base station then, waits for an "*initialize*" message from any of the terminals.

State 1. When an "*initialize*" message arrives on the base station the module invokes a method that generates a "trigger" message. The message then is sent back to the original sender through the access point, triggering the start of a video streaming.

State 2. The base station in this state behaves exactly as the terminal in state 5. If it receives a request for a bframe from a certain terminal, it invokes a "*RPL bframe*" message generator method which sets the parameters of the reply packet, i.e. packet length, appropriately. The base station attaches the id of the original sender on the reply and forwards the message to the access point.

The last simulation module of our implementation is the access point. As we have mentioned in Chapter 3 this module's role is auxiliary. It's only contribution revolves around the routing of messages that the terminals exchange with the base station. Again, we demonstrate the different states of this module in Fig. 20 .

Idle. The ***initialize()*** method of this simulation module is actually blank, because there is nothing to be initialized. It does not maintain any kind of information regarding the video files or anything else relevant to our architecture.

State 1 & **State 3**. These are the two states in which the access point receives messages from the terminals and forwards them to the base station. The forwarding is done by invoking methods for each case.

State 2 & **State 4**. In these two states the access point forwards messages from the base station back to the terminals. Again, the forwarding is performed by invoking the appropriate methods. An interesting observation regarding all the states of this module is that in every state the access point forwards a message that caused the transition on the current state.

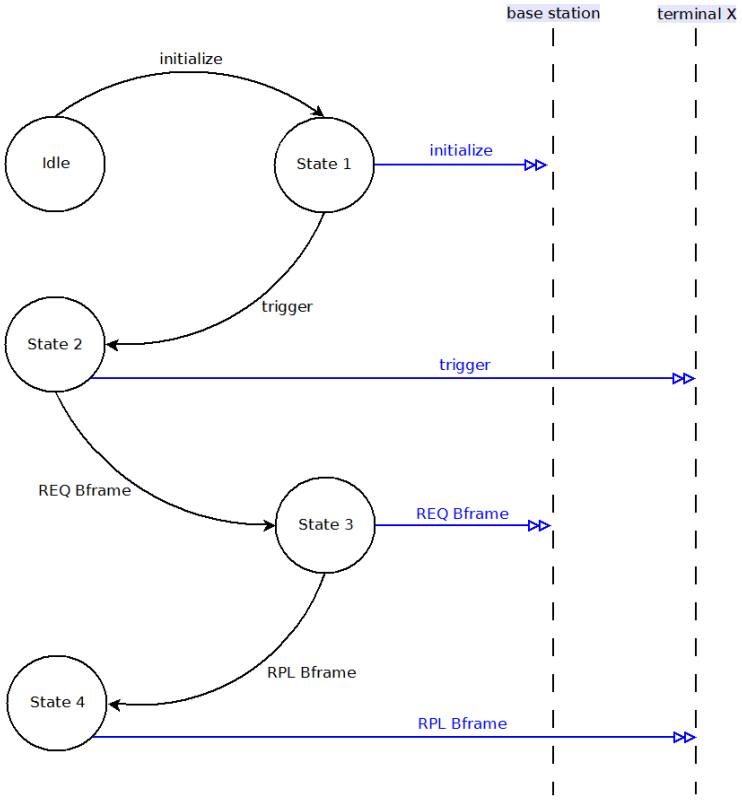

Figure 20. Access point state machine

Chapter 5

Experiments

Despite the fact that the main purpose of this Thesis is to implement a generic framework for a distributed video sharing architecture for heterogeneous wireless networks, one final goal is a brief analysis of the behavior of our system under various conditions. These conditions regard structural parameters of the system such as data rates or the number of terminals and, also, other parameters such as the Window of Availability.

In order to verify the correct behavior on each occasion and validate the functionality of our implementation we conducted a series of experiments. These experiments involved some alternations to the C++ code of our modules. However, Omnet++ offers a result recording and statistics collection mechanism that helped us to limit those alternations, thus maintaining the complexity of our code in permissible levels.

5.1 Validating the system's behavior

Initially we tested whether the system actually works as it was supposed to. We initiated a network consisting of three terminals. We assumed that the video file consists of 20 GoPs and each GoP accommodates 5 bframes. The packet length that we used in all our experiments is set to 1475 bytes. This makes the size of a GoP 7375 Bytes. We should note here that this number has not connection at all with GoP sizes of coding methods such as MPEG-4. It's purpose is to facilitate a reasonable packet size in order to help us test our system in permissible time limits.

In this first experiment we tested two scenarios. In the first scenario, all three

terminals request the video file at the same time. Additionally we deactivate the update mechanism. Therefore, all the terminals start to stream the file in the conventional client-server model. Every terminal requests a bframe and when it receives it proceeds with the next request. In the second scenario we test the fully cooperative case, which dictates that a terminal joins the network while the other two have already streamed the video file and have it cached. From now on we are going to refer to the first scenario as the *standalone* and to the second one as the *fully cooperative*.

The results of both simulations were plotted on the same diagram Fig. 21. The time of the vertical axis is the accumulated time of the simulation, i.e. the sum of the completion times of the three terminals. On the horizontal axis we have various *AP2Internet* data rates. As we move rightwards on this axis the data rates increase. In all occasions the fully cooperative streaming is faster than the standalone one. A very interesting observation is the steady difference in accumulated times of the two systems no matter how big the data rate of the Internet connection is. The real improvement then relies on the approach of our design and not on connection speeds.

5.2 Experimenting with the Window of Availability

In the next series of experiments we concentrated on a critical parameter of our implementation. *Window of Availability* serves as an update indicator affecting the dynamic nature of the architecture. The lower the *WoA* is, the more dynamic our system is. Excluding this parameter for this sequence of experiments we used again the same parameters as before. Our scenario in this case involves three terminals who do not request a stream simultaneously, but sequentially with some seconds separating the requests. We record the finish times of each one and accumulate them to take the diagram of Fig. 22. The *WoA* is illustrated as the percentage of the total number of the video's GoPs. We observe that for small window sizes the times remain low and as the window increases, less updates are sent to the terminals, thus many of them do not have information about the availability of GoPs on terminals and they request them from the base station. If the update mechanism is completely disabled (No WoA) the accumulated times are the same as the standalone scenario of the first phase of experimentation, which is actually the worst case scenario.

This gain in performance though comes at a cost. Small window equals with many updates, which equals with many "*Update*" messages exchange on the network. During this phase of experimentation we modified the C++ code of the terminal modules in order to record the number of update messages exchanged between the terminals for. We run again the same scenario for the same *WoA* values. The results are illustrated in Fig. 23 which confirms our claim. Small windows increase the number of messages on the local network.

From this diagram though we can infer another important issue. When a terminal completes it's Window of availability, sends an update message. In order for the GoPs of the window to be available for streaming, they have to be cached on the terminal. This puts caching on the frame, too, and it can be combined with research towards distributed caching. Finding a trade off between these parameters is an emerging challenge [5] .

5.3 Measuring the latency

An important quality measure regarding video streaming is the latency, especially the one of the first GoPs, which dictates the playback startup. In this phase of experimentation we took advantage of Omnet++'s record result and statistics mechanism. We used it to record the delay of the reply packets. The delay was measured by timestamps attached to the reply messages.

We simulated the same scenario as in the previous phase. This time we opted for a different representation of the results. We present the delay of each terminal in a separate diagram. This alternation offers a better insight on the systems dynamic nature.

In Fig. 24 we observe the delay of reply packets arriving on terminal 0. This is the first terminal that initiates a request. The delays for a given Internet data rate are the same regardless of the *WoA'*s size. This is due to the fact that the first terminal always requests and receives packets from the base station. The update messages are in fact delivered, but the GoPs they revolve have already been streamed from the base station. The variation on results in this case regards the decreased delays when the Internet connection gets faster which is absolutely justified.

The delays of terminal 1, which is the second one to request the particular stream are depicted in Fig. 25. In this case we observe that there are variations among different *WoA* values, which indicates that this terminal does stream from other peers (in particular it streams from terminal 0). As we can infer from the fourth column standalone downloads from the base station deteriorate the latency. The differences of delay among simulations with different Internet connection speeds are now smaller. This provides another proof for our claim that terminal 1 streams from peers, since the Internet's speed impact is rather limited.

Fig. 26 corroborates the conclusion that we made above. Terminal 2 which is the last one to start the streaming enjoys the smallest delay of all the terminals in all simulations, due to the fact that it exploits the cached GoPs on the other two terminals and streams them through the high data rate short-range links. As well as before, in this case the Internet speed has a very limited impact, smaller than that of terminal 2, which is again justified.

5.4 Experimenting with data rates

In the next phase we experimented with the structure parameters, the data rate of the Internet connection and the wireless channels.

We simulated both the above scenarios with different *AP2Internet* channel data rates. The data rates represent the speeds of modern ADSL and VDSL connections. Moreover, the data rates of the wireless channels are 72 and 54 Mbps for *WirelessA* and *WirelessB* respectively. These values maintain the condition that the local bandwidth (*WirelessA*) is always higher than the bandwidth to the Internet (*WirelessB*). The results for five different connections are illustrated in Fig. 27. As we can observe the finish time of the particular terminal of the system (the one that is not a seeder) is the same in all occasions which is correct due to the fact that the fully cooperative scenario does not involve any streaming from the base station. The line that represents the finish times of the terminal in the standalone scenario is declining as the connection speeds increase. We expect this behavior because higher data rates lessen the transmission time of the packets, thus leading to decreased streaming completion time for the particular terminal.

In Chapter 3 we referred briefly to the various wireless connection options that are available for the local WAN. During our experimentation we tested some different combinations while keeping the *AP2Internet* data rate at 8Mbps and the results are depicted in Fig. 28. An interesting observation on this particular figure is that the second pair of columns demonstrates the smaller difference between the two scenarios and the fourth the biggest. If we take into consideration that the standalone scenario relates only with the *WirelessB* link and the fully cooperative one only with the *WirelessA* link this outcome is expected. Of course in this occasion an individual could argue that we stretched the boundaries of the assumption we made before, i.e. that the local bandwidth should be higher than the bandwidth to the internet. We should keep in mind thought that the *AP2Internet* link still serves as a bottleneck in the connection of a terminal to the Internet, thus we took the initiative to assign equal speeds to the wireless connections.

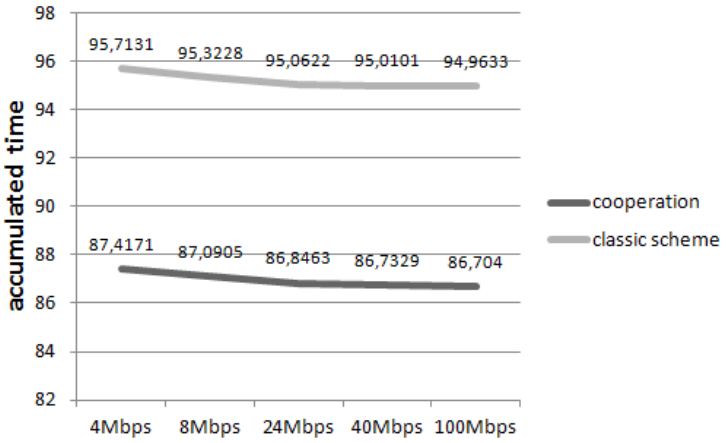

Figure 21. Accumulated time for two different system configurations. The classic-standalone and the distributed-cooperating streaming.

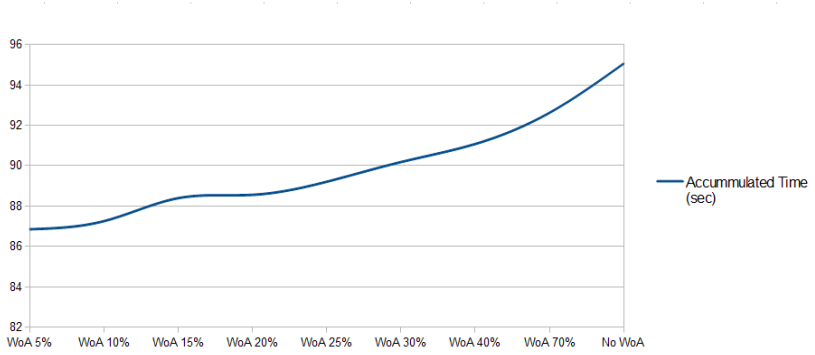

Figure 22. The effect of WoA on the accumulated time of a three-terminal distributed-cooperative system.

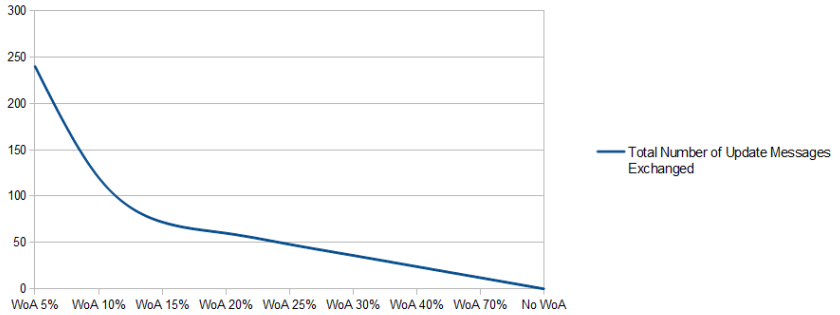

Figure 23. The number of "Update" messages exchanged in a three-terminal distributed configuration for various WoA values.

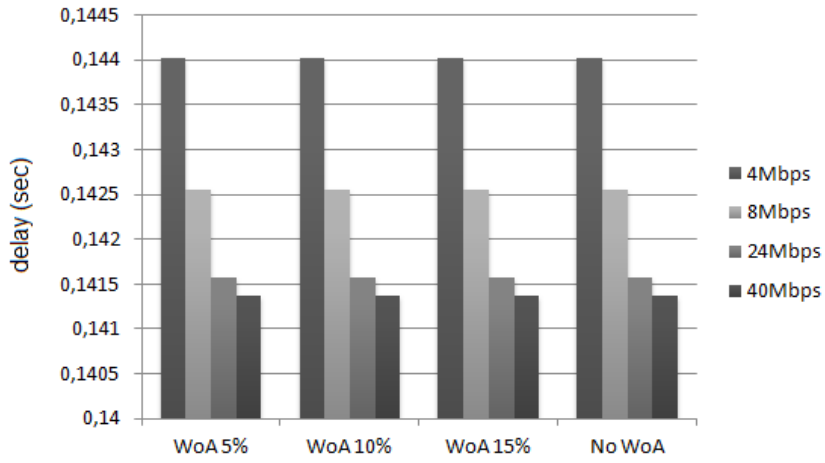

Figure 24. The delay of the first terminal that triggers a stream on a three-terminal distributed configuration, illustrated for various Internet connection speeds and indicative values of WoA.

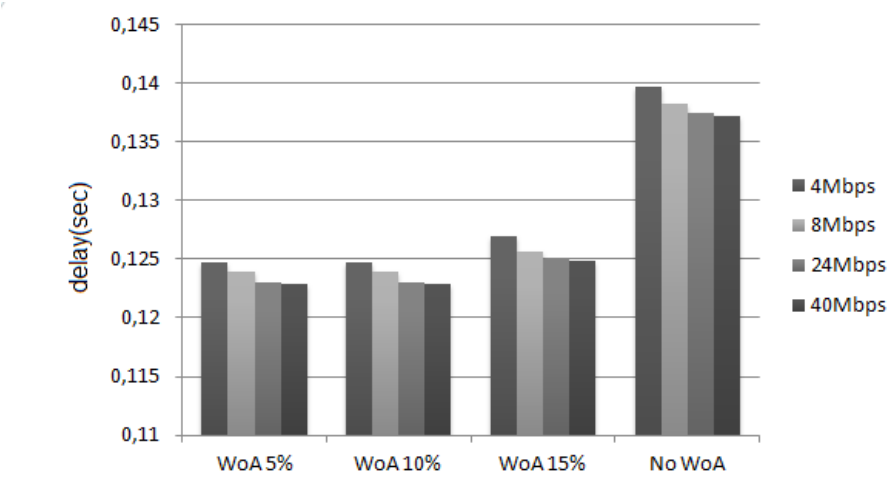

Figure 25. The delay of the second terminal that triggers a stream on a three-terminal distributed configuration, illustrated for various Internet connection speeds and indicative values of WoA.

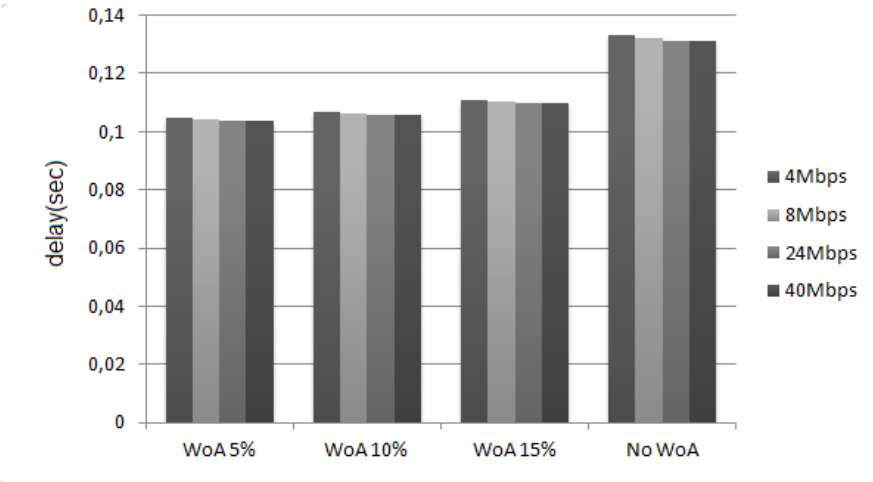

Figure 26. The delay of the third terminal that triggers a stream on a three-terminal distributed configuration, illustrated for various Internet connection speeds and indicative values of WoA.

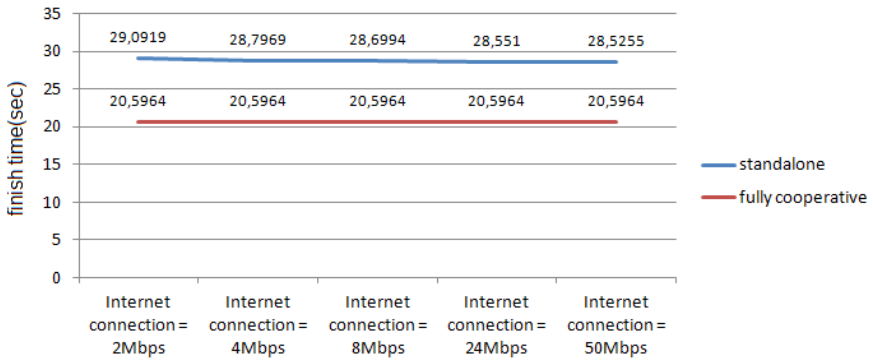

Figure 27. The effect of the Internet connection speed on the finish time of terminal 2 (the second that starts to request the video) of a three-terminal distributed system run for both configurations.

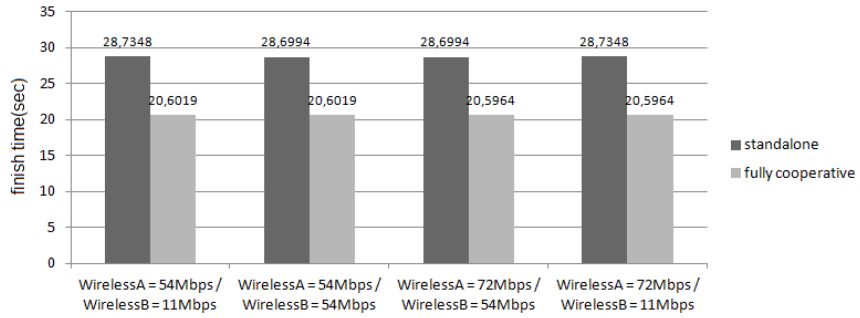

Figure 28. The effect of the combination of various Wireless data rates on the finish time of terminal 2 (the second that starts to request the video) of a three-terminal distributed system run for both configurations.

This page is intentionally left blank

Chapter 6

Conclusions

In this final section we restate the observations that we made after experimenting with our implementation and we conclude.

The rise of broadband Internet connections and the oversupply of cheap yet powerful mobile devices led to a boost in sharing multimedia content over wireless networks. Furthermore, as we discussed in Chapter 2, research on the heterogeneous wireless networks shows that the development towards fully cooperative networks may facilitate higher data rates without increasing the complexity. Additionally, micro cooperative networks may be regarded as centrally controlled p2p systems. We combined this observation with various elements of existing p2p systems targeted into video streaming and we implemented a distributed video sharing architecture for local wireless networks with heterogeneous links.

Moreover, we conducted a series of experiments in order to verify the behavior of our system under certain conditions. During these experiments we focused on the characteristics that mostly defined our approach and could validate the correct behavior in each occasion. We examined the performance of the distributed architecture under various data rate combinations. For all possible combinations the results indicate that our approach does offer an advantage when compared to the contemporary client-server streaming model for local WANs. Further, since latency is a decisive factor for video streaming performance we showed that that our system decreases it. The most interesting outcome of these experiments though, is the impact of the update mechanism that our implementation embodies. Although it is essential for the distributed function of the system, an aggressive configuration of this mechanism may increase dramatically the number of messages on the network. Finally, this mechanism can also be linked with issues regarding the impact of caching data in terminals.

With this work we showed that there is a great potential in approaches regarding video sharing that are targeted into cooperative networking for wireless networks.

Bibliography

[1] INTERNAP, The next big wave in convergence – Building the right foundation for Internet TV. *Streaming Media Magazine*. Retrieved May 5, 2010 from www.internap.com .

[2] Telecom, ATIS Telecom Glossary 2007, American National Standard for Telecommunications. Retrieved June 1, 2010 from http://www.atis.org, 2007.

[3] B. Dutson, The European Streaming Industry: Clearing the Barriers to Growth. Re-trieved 12, May 2010 from http://www.streamingmedia.com/ .

[4] http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537 /ns705/ns827/white/paper/c11-520862.html .

[5] K Shanmugam, N Golrezaei, A G. Dimakis, A. F. Molisch, G. Caire,FemtoCaching: Wireless Video Content Delivery through Distributed Caching Helpers, *In proceedings of the 33rd Annual IEEE International Conference on Computer Communications*, 2012.

[6] Qi Zhang, Frank H.P. Fitzek, Marcos Katz, Evolution of Heterogeneous Wireless Networks: Towards Cooperative Networks, *In proceedings of the 3rd International Conference of the Center for Information and Communication Technologies*, 2006.

[7] http://en.wikipedia.org/wiki/Streaming_media. Streaming media, Wikipedia page.

[8] S. M. Thampi, A Review On P2P Streaming, published in eprintarXiv:1304.1235, 04/2013.

[9] http://en.wikipedia.org/wiki/Video_codec. Video Codecs, Wikipedia page.

[10] http://en.wikipedia.org/wiki/Group_of_pictures. Group of Pictures structure, Wi-kipedia page.

[11] Aberer, K. & Hauswirth, M.,An overview on peer-to-peer information systems, Proc. *Workshop on Distributed Data and Structures*, pp. 171-188, March, 2002

[12] Meddour, D., Mushtaq, M., & Ahmed, T., Open Issues in P2P Multimedia Streaming, *MULTICOMM 2006 proceedings*, pp.43-48, 2006.

[13] Liu, J., Rao, S. G., Li, B. & Zhang, H., Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast, Proc. *IEEE*, Vol. 96 Issue 1, pp. 11-24, 2008.

[14] V. Pai, K. Tamilmani, V. Sambamurthy, K. Kumar, and A. Mohr, B Chainsaw: Eliminating trees from overlay multicast, *Proc. 4th Int. Workshop Peer-to-Peer Systems (IPTPS)*, Feb. 2005.

[15] http://www.omnetpp.org/. Omnet++ Network Simulator webpage.