FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Peer to Peer Multicast Overlay for Smart Content Delivery

Afonso da Rocha Graça

Masters in Informatics and Computing Engineering

Hungarian Supervisor: Róbert Szabó (PhD)

Portuguese Supervisor: Rui Maranhão (PhD)

18^{th} July, 2012

Peer to Peer Multicast Overlay for Smart Content Delivery

Afonso da Rocha Graça

Masters in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Doctor Pedro Alexandre Guimarães Lobo Ferreira do Souto
External Examiner: Doctor José Maria Amaral Fernandes
Supervisor: Doctor Rui Filipe Maranhão de Abreu

18^{th} July, 2012

Abstract

The constant increase in bandwidth and bit rate happening nowadays amongst networks, makes it easier and easier to communicate and distribute information throughout the world. In spite of that, still many of these interactions rely on a *server-client* model, which has been proven to be very suitable for most of said interactions over networks, but still it has a huge drawback, it depends on one entity, the server, and if the server fails, all the model goes along with it. *Peer-to-peer* (P2P) model appeared to remedy this shortcoming but it has not been able to achieve the level of the server-client model.

In this report, a new alternative to both these models will be introduced, described and put up to a performance test, this alternative is called *network coding*. This technique promotes the mixing of data in all the nodes of a network, achieving a greater throughput with the penalty of an added delay. A study across on how network coding works is presented, followed by a test and evaluation trial that presents the performance behaviour of said technique when compared to traditional multicasts and multiple unicasts.

It is proven in this report, that by applying this technique to the various existing networks nowadays (i.e. Internet, LAN, mobile network), these benefit far more than suffer from detriments brought by network coding, like the delay, always achieving faster distribution times.

This report was written in the context of a dissertation work and so it comprises three main parts: the notions of what network coding is and why should it be used; the development of an efficient solution to P2P networks using network coding; the description of the experimental setup, the simulation environment and results obtained by these simulations and consequent evaluation.

Resumo

O constante aumento na largura de banda e velocidade de transferência (i.e. bit rate) que se tem vindo a verificar no universo das redes, torna possível que seja cada vez mais fácil comunicar e distribuir informação por todo o mundo. Apesar desta evolução, ainda muitas dessas interações dependem de um modelo *cliente-servidor*, que se tem assumido e provado como muito apropriado para a maioria desse tipo de interações em redes, mas ainda tem uma desvantagem enorme: depende de uma entidade - o servidor, e se o servidor falhar, todo o modelo falha por conseguinte. O modelo *Peer-to-peer (P2P)* propõe-se para colmatar essa lacuna, mas não tem sido capaz de alcançar o nível penetração no universo das redes do modelo cliente-servidor.

Neste relatório, uma nova alternativa a ambos os modelos será introduzida, descrita e o seu desempenho será posto à prova. Esta alternativa tem o nome de *network coding*. Network coding é uma técnica que promove a mistura de dados em todos os nós de uma rede, permitindo assim um maior *throughput* sob a pena de um atraso adicional. É apresentado um estudo sobre como funciona o *network coding* é apresentado, seguido de um ensaio de testes e posterior avaliação, para registar o comportamento em termos de desempenho da referida técnica quando posta em comparação com as técnicas utilizadas actualmente, isto é, *multicasts* tradicionais e *unicasts* múltiplos.

É provado neste relatório que, ao aplicar esta técnica a diversos tipos de redes existentes hoje em dia (i.e. Internet, *LAN*, rede móvel), estas beneficiam, em larga medida, muito mais do que sofrem dos malefícios por ela criados, como o atraso por exemplo, conseguindo sempre alcançar tempos de distribuição de informação mais céleres do que com as técnicas habituais.

Este relatório foi escrito no contexto de um trabalho de dissertação e, portanto, é composto por três partes principais: esclarecimento das noções necessárias para a compreensão do *network coding* e porque é que deve ser posto em utilização; o desenvolvimento de uma solução eficiente para redes *P2P* aplicando *network coding*; descrição da configuração experimental, do ambiente de simulação e dos resultados obtidos por essas simulações e a sua consequente avaliação.

Acknowledgements

The author of this technical report would like to acknowledge *Balázs Lajtha* (PhD), *Csaba Simon* (PhD), *Mariana de Ascenção* (journalist), *Paula Tavares* (English teacher), *Róbert Szabó* (Hungarian supervisor) and *Rui Maranhão* (Portuguese supervisor) for their help, insight, suggestions and technical support. It goes without saying that without their help, expertise and support, this report would not be the same and it would not have the same value.

The author would also like to leave a brief mention to all his colleagues, family and friends, that throughout their friendship and support made this academic journey worthwhile, acknowledging that this dissertation would not be the same without their presence and influence.

These past five years have been one hell of a ride.

Afonso da Rocha Graça

"Mixing one's wines may be a mistake,
but old and new wisdom mix admirably."

Bertolt Brecht

Contents

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

1.1 Context

Under the environment of a *Masters Dissertation* on *Informatics and Computer Engineering* and two partnering entities, *Budapesti Műszaki és Gazdaságtudományi Egyetem* and *Faculdade de Engenharia da Universidade do Porto*, this report has two main objectives: not only to document all the technical aspects of the dissertation but also to document the development stage of the dissertation, its results and conclusions.

The technical aspects encompass all the technologies applied during the dissertation, which can be found in the state of the art chapter, which documents all the efforts done in the area. This chapter is followed by a detailed explanation of the setup and environment of the task at hand (i.e. description of the simulator, its setup and its modus operandi). Finally there is a chapter on the tests done, the evaluation of these and finally the conclusions taken from the latter.

1.2 Motivation and Objectives

Traditionally, multimedia content delivery relies on a *client-server* model, the clients request the server and the server manages and replies to these requests. This however, can put a tremendous burden on the server side of the model, which can lead to an inefficient network resource utilisation and, above all, creates a point for failure, that is, if the server fails all the model will follow it and fail as well.

One clear architectural solution for this problem is a *P2P* environment, where clients request the other clients and the clients with the appropriate answer to reply to him, achieving a mesh topology and making the model decentralised. Still there are some problems to take into account, such as the content's sensivity to network delays, as lags and jitter can render the user's experience unsatisfactory (i.e. video and audio on an online chat or multiplayer action game over the internet).

The main motivation is the ability to offering a new way to model systems that rely on heterogeneous networks and that are intolerant of delays or errors, making them more reliable and robust, since past researches have shown that network coding can be efficiently used in certain

circumstances to enable multiplayer gaming in a distributed environment (i.e. without relying on client-server communication model) [LBS10], following the idea of trading bandwidth efficiency for delay.

The main objective of this dissertation work is to develop a proof of concept, without being solely basic research, since it applies past knowledge, to understand if this solution is feasible or not in a real world scenario with network heterogeneity (i.e. between mobiles, laptops, servers, access points, etc). By designing and numerically evaluating *overlay organising algorithms* for *network coding*, applied to multicast heterogenous streaming communications.

1.3 Project

This project is being carried out as a research subtask for the *European Institute of Innovation & Technology (EIT)* more specifically for *Knowledge and Innovation Communities (KICs), Information and Communication Technologies (ICT)* Lab. It also has several parternships, including the *University of Stockholm / Ericsson Sweden* and Budapest's *Eötvös Loránd University (ELTE)*.

This project will try to apply *network coding* to the situations mentioned in the previous section and compared its performance to the standards available nowadays. *Network coding* is a technique that at its core, allows and encourages the mixing of data at intermediate network nodes and this allows the network to have a maximum flow of information achieving a larger throughput, being able to competitive.

By trading network efficiency for delay and redundancy, there could be a way of achieving a distribute *multi-to-multi point* interactive communication suite in heterogeneous networks without relying on a *client-server* model.

If this research is promising, the development of a proof of concept under the Android environment (e.g. real time video sharing through mobiles) is to be carried out by the *University of Stockholm* and *Ericsson Sweden*.

ELTE was in charge of providing several real-world topologies, but unfortunately, due to circumstances beyond the author's power, this partnership did not have a fruitful outcome. The alternative to overcome this obstacle will be presented at a later section of the report, concerning the setup of the technology in question.

1.4 Methodologies

This dissertation project followed a pure research on the *network coding* technique, algorithms and preferred topologies, followed by a simulation trial, under a Java environment, and data analysis of these simulations. All these steps of the project were supervised by the PhD student aiding in the work, through discussions and pre-arranged meetings.

After the validation and evaluation of the simulations' results, a scientific paper is to be written on the subject in order to document the research done for future reference.

1.5 Report Structure

Besides the introduction, this dissertation report contains six more chapters.

In chapter 2, the state of the art is present. The technologies to be used will be presented and briefly explained, followed by a presentation of the related works/researches done so far in the field.

In chapter 3, the baseline scenario (i.e. the scenario which will serve as a comparison to the project's simulations) is described, as well as the network coding setup to be used (e.g. topology, segment size, block size, etc).

In chapter 4, the simulator will be documented. After an introduction to the core architecture of the existing simulator developed in [LBS10], the new features and modifications made will be described.

In chapter 5, the runtime of the simulator will be explained. After the *modus operandi* is presented, the testing scenarios to be used in the simulator will be depicted.

In chapter 6, the results of the simulations of the testing scenarios previously introduced will be presented followed by an evaluation concerning our baseline scenario.

In chapter 7, the last chapter of this dissertation, the conclusions taken from the simulations will be stated and a speculation of the possible future work on this field will be presented.

Chapter 2

State of the Art

2.1 Introduction

Communication, whether through a video-conference or a courier pigeon, always had the same paradigm, the information needs a means to reach its recipient, that is, the information itself is independent of the protocol followed to deliver it. Nowadays, with the huge boom in internet communication, it is easy to comprehend that even though the same network can share data streams, the information they carry is independent and that information can travel through different types of channels (i.e. from coaxial cables to wireless channels) before it reaches its destination.

A network is conventionally operated (i.e. routing, data storage, error control) with the objective of avoiding data stream collision, when possible, and so nodes simply forward (and store in some cases) the data they have been given. However, it has been demonstrated[ACLY00] that through mixing data at intermediate nodes, there is a better resource utilisation and a better throughput is visible, achieving the max-flow of said network (i.e. the upper bound of network resource utilisation). This approach was coined as *network coding* and it states and encourages the processing of the data received before it is sent.

2.2 Network Coding

It has been a well established myth in data networking that only data replication is needed at intermediate nodes, discarding any other type of data processing. Allied to this myth, most of the computer networks nowadays use the *store-and-forward* method, in which information, in the form of data packets, is transmitted from the source to its destination. To reach the destination however, these packets are passed through a chain of intermediate nodes, that collect all the packets received through its input channels, stores and forwards them to all its output channels.

But since its full development in [ACLY00], where the actual network coding term was coined, network coding has been proven to have some advantages over the store-and-forward method, thus breaking the myth. A simple example of these advantages can be perceived in figure 2.1, when two parts (A and B) want to communicate through an intermediary (S), and the transfer rate is unitary,

it would take four units of time to fulfil the conversation, on the traditional method, whereas with a simple application of some network coding (logical sum - *XOR*) in the intermediary, the time to fulfil the conversation is reduced by one.

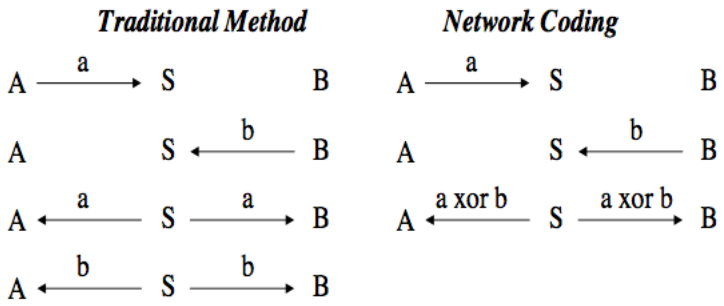
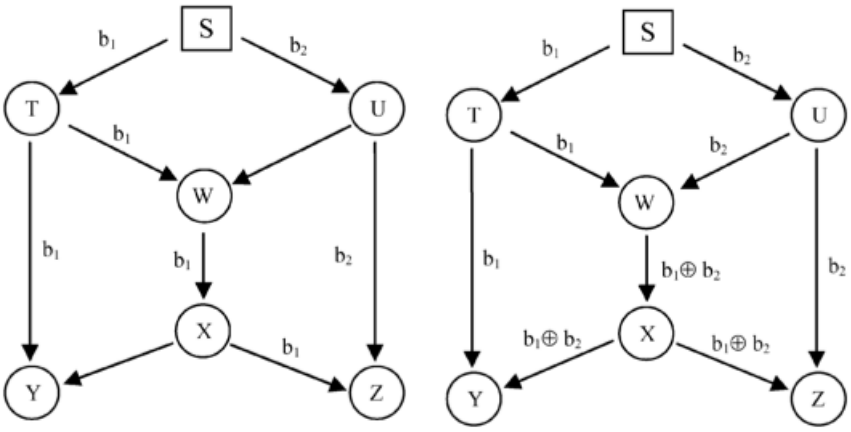


Figure 2.1: Simple network coding application example

When discussing communication networks throughout this chapter, they will take the form of finite directed graphs, such as the ones in figure 2.2, and will be denoted by the letter G. Each graph contains a set of nodes and a set of edges, and hence $G = (V,E)$, with V representing the collection of nodes and E the collection of edges. In the figures, the nodes that have no input channels are called *source nodes*, and they are represented by a square, while the rest are represented with a circle. Nodes that are supposed to receive the information sent from the sources are called *sink nodes*. The channels of the network are considered edges in a graph and when there is a channel from node X to node Y, the edge is called XY. The capacity of transmission between two nodes will be represented by the multiplicity of edges between nodes, from this it can be concluded that each edge has a unit capacity.

To firstly grasp how network coding works, a simple network known as the *butterfly network* (Figure 2.2) is often used. In this network there is a single source node, with two different messages ($b1$ and $b2$) to send to two sink nodes (Y and Z). When analysing figure 2.2(a), a traditional multicast without the use of network coding, it is easy to understand that it is impossible to transmit both messages to both sink nodes in just one round, another round would be needed to transmit the missing messages or a higher capacity on the channel WX to be able to achieve the transmission in one round. Figure 2.2(b) though, shows that node W applies a simple form of coding, deriving the exclusive-OR $b1 \oplus b2$ from the previous received messages (i.e. $b1$ and $b2$) and forwards it throughout the rest of the network. Once it reaches the sink nodes, each of them have the coded message and one original message and with those two, the missing message can be decoded at the sink node.

By applying a simple coding in one intermediate node, the nine channels in all the network are used exactly once, where in the case of the traditional network, without network code, the best case scenario would be adding a channel and using the 10 channels once. This proves that network coding has the potential to reduce latency and energy consumption, since it would run all the channels just once, and because of this it maximises the bit rate of the network.



(a) Traditional (b) Network Coding

Figure 2.2: Butterfly Network

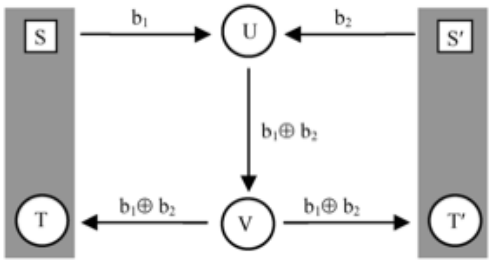


Figure 2.3: Conversation between two devices using network coding

In order to frame this technique in a more realistic environment, figure 2.3 represents two devices, such as computers, smartphones or tablets, each one with a source and a sink node. They communicate with each other through the channel *UV*, sending data from *S* and *S'*, respectively, to *U* and receiving the combination of both parties' data from *V* at *T* and *T'*. When this data is received the desired message can be obtained by decoding the received data with the one that was sent. As it can be seen in figure 2.4 by combining the data sent by each party, a higher bit rate can be achieved than with separate transmission (i.e. a time unit to transfer *b1* and another one to transfer *b2*), thus reducing the downlink bandwidth.

2.2.1 Fixed Network Coding

2.2.1.1 Encoding

Building upon the considerations defined regarding the graph approach to network coding in the previous section, every node in the network possesses a set of incoming channels and a set of outgoing channels, this means that for every node T, In(T) represents the former and Out(T) represents the latter. Concurrently, when referring to source nodes, In(S) represents a set of context

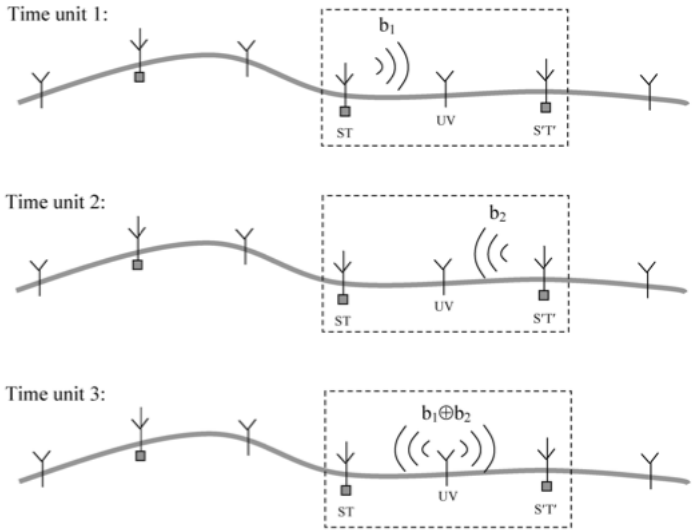


Figure 2.4: Time elapsed on conversation between two devices using network coding

dependent imaginary channels and it is denoted by w. For the butterfly network in figure 2.2(b), the graph would have two imaginary channels for each message (*b1* and *b2*) and $w = 2$.

The unit of data used in a network depends on the *finite field* F chosen. A finite field is a set of elements over which arithmetic operations can be performed in a closed manner (without overflow or underflow). In the previous example, the finite field for the butterfly network is F_2, the set of elements is $\{0,1\}$ and thus the data unit is a bit. The messages to be sent between nodes consist in a group of w data units, that means the messages are w-dimensional row vectors and in this chapter will be denoted as x. A message is propagated throughout a network by transmitting symbols $\tilde{f}_e(x) \in F$ throughout all the channels $e \in E$, this means that a message x is generated at the source node and the corresponding symbols to x transmitted to all Out(S).

It is said that a node encoding function is to simply map all the symbols received through its incoming channels to a symbol for each outgoing channel, which abides the law of commodity, that states that the total volume of outflow from a non-source cannot exceed the total volume of the inflow (i.e. $\text{Out}(T) \leq \text{In}(T)$). The definition of *network code* comes as the encoding mechanism for every channel and it consists of a local encoding mapping for each node of the network and a global encoding mapping for each channel.

Since node processing can bring a heavy burden, which in turn provokes delay, linear combinations over a finite field are of particular interest because they are easy to describe and efficient to compute, invertible and sufficiently rich in terms of processing potential.

Linear global encoding mapping is a w-dimensional column vector f_e, and by multiplying it by the message x generated by S, the symbol $\tilde{f}_e(x)$ to send can be obtained. Analogously, linear local encoding mapping k_e, where $e \in \text{Out}(T)$, is a $|\text{In}(T)|$-dimensional column vector, and by multiplying it by a row vector representing the symbols received at the node T, the mapping of the node is obtained. If all the global encoding mappings are linear, then so will be the local encoding mappings, and vice-versa.

An *adjacent pair* is a pair of channels (d,e), where $d \in \text{In}(T)$ and $e \in \text{Out}(T)$ for every node T, and for every adjacent pair there is a scalar $k_{d,e}$ called *local encoding kernel*, this scalar defines if there is an outgoing channel receiving anything from an incoming channel. At the same time, the local encoding kernel at a node T, is equivalent to the matrix product K_T between $\text{In}(T)$ and $\text{Out}(T)$. Some sort of order has to be assumed among channels when structuring K_T. As expected, the vector f_e of which the global encoding mapping consists, is called the *global encoding kernel* for channel e. This vector is composed by $\sum_{d \in Int(T)} k_{d,e} f_d$, where $e \in \text{Out}(T)$.

When a node T receives the symbols given by $x \cdot f_d, d \in \text{In}(T)$, it uses the linear formula $x \cdot f_e = x \cdot \sum_{d \in Int(T)} k_{d,e} f_d = \sum_{d \in Int(T)} k_{d,e}(x \cdot f_d)$ to encode the symbol to be sent onto each channel $e \in \text{Out}(T)$. It can be concluded that if all the encoding kernels for every channel are known, the global encoding kernels can be calculated recursively, in any upstream-to-downstream order.

Continuing with the example given in the previous section, it is easy to understand how the encoding works. As it can be seen in figure 2.5 there are two imaginary channels *OS* and *OS'*, each carrying one data unit, which in the previous example were the messages b_1 and b_2. From this, and using the methods described previously, it is easy to obtain the local encoding kernels and subsequently the global encoding kernels, as the figure shows.

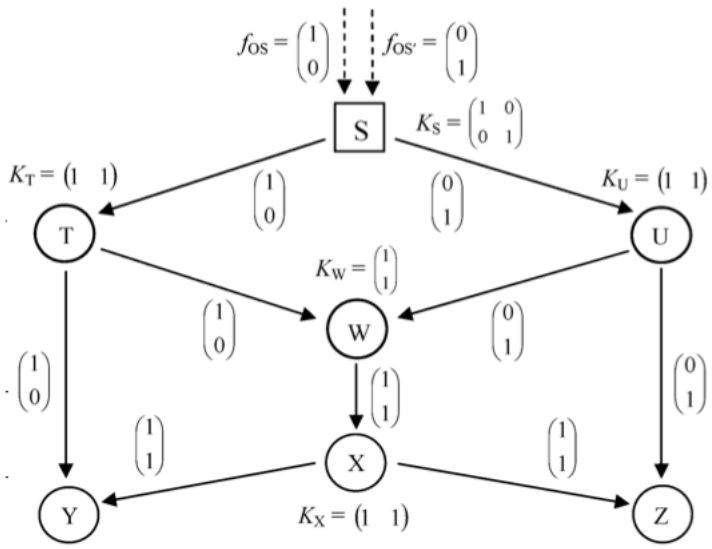


Figure 2.5: Local and global encoding kernels in a butterfly network

2.2.1.2 Decoding

On a fixed network coding scenario, $k_{d,e}$, the local encoding kernel, for all $d,e \in E$ are predefined and included in the global encoding kernels $f_e : e \in E$. This means that when a message is received through an incoming edge of a node, either sink or intermediate, the message was generated by the upstream node, and therefore can be described by the upstream node's encoding function, for that specific outgoing edge.

Also in a fixed scenario, the system is synchronised, meaning that all incoming edges of the node will have a message available to deliver to the node when time comes. Since the kernels in use are linear, $\tilde{f}_e$ is the product of the corresponding global encoding kernel, f_e and the original message x.

When relating to intermediate nodes, the message to be sent to the outgoing edges can be obtained by concatenating each symbol retrieved from the incoming edges. This message, when referring to figure 2.5, on node W is $m_W = M_W \times x$, where M_W is a matrix constructed with the global encoding kernels of the incoming edges. This way the message is re-generated to then send to all the outgoing edges.

On the matter of sink nodes, seeing that these nodes do not have incoming edges, there is no need to create a message, but still, as long as it receives the same number of received messages as w (i.e. imaginary channels), the node also creates a matrix containing the f_e of the incoming edges. This matrix, when inverted and multiplied by the received messages, can decode all the original messages. Again when referring to figure 2.5, when node Y receives both messages from the channels TY and XY, it builds the matrix M_Y and subsequently its inverse M_Y^{-1}, as

$$M_Y = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \text{ and } M_Y^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

By creating a matrix with the received messages and multiplying it by the inverse M_Y^{-1}, it retrieves all the original messages in each line, as

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

As it can be seen, the first line corresponds to OS and the second to OS'.

2.2.2 Random Network Coding

In [HLM+03], *Ho et al* devised a random approach to network coding that proved to be very beneficial when the topology of the network is unknown or changes through time. By combining and transmitting previously received blocks with random and independent generated coefficient, according the the finite field defined, it also proved that the time needed to transmit larger files to peers in the network is reduced. This way data is encoded locally on the source or intermediate without having the need to previously know the topology of the network.

The received combined blocks have a probability of being independent from each other relating the inverse of the size of the finite field, and when each sink node receives as many independent combinations as the number of source messages blocks, it will be able to fully decode the message in question.

2.2.2.1 Encoding

Random network coding is applied as follows: for each input an intermediate node receives, it is generated a random and independent coefficient over the specified finite field. Each coefficient is then multiplied to its corresponding input and all the inputs are added, creating the data block of the message. Before sending the message to the output edges, a block is added to the beginning of the message, containing all the coefficients used to compose the data block.

As an example, let us consider node W from figure 2.6. Using random network coding, W receives packets from channels TW and UW and when it is time to send a message it generates two random coefficients (α_1 and α_2) in the finite field defined, which in this case is $F = 2$, achieving a message $[\alpha_1, \alpha_2][Y_{W,X}]$, where $Y_{W,X} = \alpha_1 \times TW + \alpha_2 \times UW$. If the generated coefficient for α_1 is 0 and for α_2 is 1, the data to send to X is $0 \times TW + 1 \times b_2 = UW$ and the format of the message is $[0,1][UW]$.

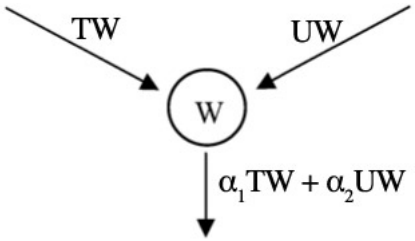


Figure 2.6: Encoding under random network coding of node W

2.2.2.2 Decoding

When a sink nodes receives, it stores the received packets, encoded or not, row by row, in what from now on will be known as a *decoding matrix*. On the first rows of such matrix there will be the non-encoded packets received by the sink node, with the corresponding encoding vector, if the said vector exists. Every time a encoded packet is received it is inserted in the last row of the matrix, and it can be considered non-innovative or innovative, depending on whether or not it produces a row of zeros by *Gaussian elimination*, respectively.

Continuing the previous example, when the sink node Y receives the first message from the channel TY, it gets an non-encoded packet $\begin{pmatrix}1\\0\end{pmatrix}$ and adds it to the decoding matrix. When it receives the encoded packet from channel XY $\begin{pmatrix}1\\1\end{pmatrix}$ it adds it as the last row of the decoding matrix, leaving it like

$$\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}.$$

By applying Gaussian elimination to the matrix, the first row, already in echelon form, is subtracted from the second (i.e. the encoded packet), leaving it also in row echelon form. As it

can be seen from the result

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

the matrix is in triangular form, and the decoded message is received, that is $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

2.2.3 Benefits

As can be perceived throughout this chapter, network coding can potentially bring several benefits when applied to a network, which makes this technique so interesting and so worth looking into. The major benefits found so far are:

- **Transmission rate**: the main idea that lead to the research of network coding was to achieve the max-flow bound on the data transmission rate in a multicast scenario. When the problem is looked upon as a graph problem, it resembles the *max-flow min-cut* theorem of graph theory. So by analysing the network and defining that the transmission rate is equal to the minimum transmission rate registered between the source and the sink nodes, we can guarantee that there will not be fluctuations on the transmission rate;

- **Load balancing**: when applying network coding one is not only defining the shortest path and the maximum flow between nodes, but also ensuring that all the paths between the source and each sink are used to achieve said max-flow and by doing this the burden of the data transmission is equally distributed through all the paths, diminishing the burden put on some paths;

- **Bandwidth savings**: by combining and compressing data packets received at a node into one encoded packet to send to the output nodes, network coding is able to save bandwidth in situations where sometimes traditional multicast would not, since it only uses a store-and-forward technique, it only replicates the data received, so if it receives more than one data packet, it will take more units of time to achieve what network coding can achieve in a single one;

- **Network management and robustness**: the ability to recover from long term link failures, like a link cut or a permanent removal of an edge, is also present in the network coding technique seeing that this technique can operate under failure scenario, since these are previously designed, adding no further overhead in terms of management when compared to the need to do rerouting or path protection in traditional network management, making it both more robust and easier to manage;

- **Security**: since non-source nodes do not necessarily receive every symbol on a message, and can introduce incremental or different encodings to the same symbols, it makes the job to decipher these packet harder (i.e. linear combination are invertible all by themselves) and so it renders attacks such as *man in the middle* mute. If we consider random network

coding, security is taken to another level since the topology of the network does not need to be known, each node produces random coefficients that are merged into the data itself, creating a level of information security, almost as a free cipher as proven in [LMB07];

- **Latency and energy consumption minimisation**: by saving bandwidth and balancing the load imposed on the network, network coding may reduce latency and energy consumption as a result, for the reason that since it may not need so many time slots to deliver the same amount of data, the data reaches the receivers faster and also the energy that would be needed during those extra time-slots is saved, making the network more efficient.

2.2.4 Detriments

Even though network coding can potentially bring a wide range of benefits in several areas, it is not perfect and contains some detriments that should be taken into account when considering its application to a network. The detriments so far are:

- **Delay**: in traditional multicast intermediate nodes just store the received data and forward it to their outgoing nodes, but in network coding the received data needs to be processed before it can be sent, and like everything this process takes some time adding some delay to the connection;

- **Synchronisation**: since intermediate nodes need to encode all their incoming channels into one data packet to send to their outgoing channels, it is necessary to achieve synchronisation between the times of arrival of all the incoming channels to prevent process hanging and possible end of communication due to lack of answer, however in real time networks, the information packets travel asynchronously and such packets may experience random delays or losses during the transmission through an edge;

- **Packet loss**: if one packet is lost during the communication, the decoding and understanding of more than one packet could be in jeopardy and as a consequence put all the communication at risk;

- **Failures**: if some kind of loss or failure happens, the decoding nodes must be aware of the failure patterns in order to apply the most suitable linear decoding function, which proves to be a problem since this pattern needs to be communicated in a reliable way;

- **Knowledge**: some centralised knowledge of the topology applied must be known in order to be able to determine the maximum transmission rate inside a network or even the coding agreement between nodes, which can be difficult to obtain in a real implementation of the network and also the ability to broadcast this kind of information to all the network, in a reliable way, may pose some concerns.

Still, if it is possible to overcome and prevent these problems, this technique is a very powerful one.

2.3 Related Work

This thesis work appears as the follow-up to the work done in [LBS10] where some research work was done within the area applied to online multiplayer gaming. As common knowledge dictates, millions of people play online multiplayer games everyday, and even though some of these games do not really need many updates per second (e.g. MMORPG like World of Warcraft or Starcraft) others do, especially action games like first person shooters (e.g. Medal of Honor, Call of Duty). If such games suffer some delay, the whole user experience can be ruined.

Studies that have been carried out prove that this proposed network coding technique can boost network capacity when compared to the traditional store-and-forward mechanism in a variety of scenarios, outperforming traditional unicast in terms of average network latency. The opportunity for the present thesis work came from the need to further research the impact of this method on more complex topologies, not only in terms of packet loss but also computational overhead, in order to be applied into, for example, real-time video calling between mobiles, to fully understand if this technique is robust and simple enough to be handled by the low computational power of such devices.

2.4 Possible Applications

Network coding may offer potential benefits in several areas and this potential was acknowledged and is being researched by some of the biggest technological companies in the world like Microsoft, Hewlett-Packard, Intel, Cisco and Ericsson, to name a few. The most important scenarios to take into consideration are: content distribution and multihop wireless networks.

When talking about content distribution, whether it is file distribution, streaming or distributed storage, network coding may bring advantages during the scheduling protocol which in turn can lead to shorter downloading times and better robustness to a possible departure of an entity in the network. It is not difficult to deduce that if network coding is applied to P2P networks of communication streaming, it would provide a faster obtaining of files and a steadier and more fluid connection when streaming multimedia (i.e. video and audio for example) in a communication.

Multihop wireless networks comprises wireless mesh networks, wireless sensor networks, mobile ad-hoc networks and cellular relay networks. The application of network coding to these networks potentially provides a gain in network transmission rate, with more nodes becoming source nodes (i.e. transmitting information). When referring to cellular and mobile networks, it must be taken into account that these kind of devices nowadays, still cannot handle the communicational power of other devices (i.e. uplink and downlink channels are not powerful enough when compared to other devices), but still they are able to create, store and exchange multimedia content to cloud services (e.g. Facebook, Twitter) and by applying network coding in these situations, the transmission delay may be surpassed, avoiding the congestion of these channels.

2.5 Summary

Throughout this chapter, the technique that will serve as the core to this thesis work has been discussed. Network coding is a technique that encourages data processing between every node belonging to a network to achieve the maximum throughput according to the max-flow min-cut theorem of graph theory. This data process between nodes is done using algebraic linear combinations, which are fast to compute and easily decoded with the right information. Network coding offers several appealing benefits when compared to the traditional multicast techniques but still has some flaws that need to be overcome in order to achieve a big penetration in today's network systems. However, seeing that the benefits are so significant, network coding is already attracting some research interest and may have some impact on the future network design and management. From a more theoretical side, network coding proves to be an interdisciplinary research area which creates mind-puzzling questions across diverse areas such as information theory, algorithms, algebra, coding and graph theory.

Chapter 3

Network Coding Setup

3.1 Baseline Scenario

In a situation as a videoconference a predefined topology is something that cannot be established, seeing that the number of participants may vary and so may their bandwidths. With this in mind, fixed linear network coding is not an option for the implementation of such environment due to its need for the previously mentioned points. So for the development of this project, the flavour of network coding to be used is **random network coding**.

Needless to say, when trying to understand the usability and feasibility of a new network paradigm, there is a need to compare this new paradigm to the existing standard in the field. In this dissertation, the use of random network coding will always be compared to a multicast equivalent network. This decision prompted itself as the obvious one since, in a videoconference environment, users can connect to all the existing users on the service, creating a fully connected network and thus a broadcast scenario, but it can happen that some users are just interested in other specific users, so it stands to reason to use a multicast paradigm, users subscribe the stream of the desired users and these only transmit to the subscribed ones.

This dissertation project, if successful, is to be followed by the implementation of a videoconference application for heterogenous devices, such as smartphones, tablets or laptops, with this in mind, one point in consideration is the display "real estate" of said devices, and truncate the application to the maximum of the smallest device. Logically the smallest device able to operate this type of application is a smartphone, where screens have small dimensions. To maintain an acceptable level of *QoE*, there was a need to truncate the maximum number of incoming streams for each user. This upper bound was set to nine, meaning that, at most, ten users can be fully connected in a videoconference environment. The upper bound was chosen by trivially dividing a smartphone screen in a matrix and conclude what would be the maximum admissible for such an application. Although a 3×3 matrix scenario is the upper bound of the testing cases, it is believed that the optimal scenario in terms of *QoE* would be a 2×3 matrix, meaning six incoming streams.

3.2 Topologies

As was mentioned in the previous section, by using random network coding there is no need to have a predefined static network, and so the testing scenarios can be developed trying to achieve near real-life existing topologies.

In what concerns the physical layer, since all users will connect to the internet, even they are using the application connecting to *3G* instead of *Wi-Fi*, the physical layer will resemble a fully connected graph, where users can successfully reach one another without the need to relay the message through a intermediate node.

Concerning the logical network layer, the overlay network, at first, *ELTE University* was supposed to provide the author a set of topologies that were used in real world scenarios, and the author, with this in mind developed a way of parsing *graphML* files into the format that would be processed by the simulator. Unfortunately, this partnership did not go through as expected, with the Hungarian supervisor not being able to retrieve said files from *ELTE*. Facing this problem, the next option was to find a way to recreate this behaviour as much as possible, and in [SLL08], where the core is the study of topologies applied to network coding, some topologies are suggested, for certain scenarios, with the intuit of maximising efficiency.

Small-world topology, also known as *six degrees of separation topology*, has become famous over the years for explaining social interaction (i.e. the theory behind the topology states that everyone in the world is connected, on average, by six friendships/acquaintances). The key is organising all peers in a ring, like in figure 3.1, and connect each peer to a certain number of local neighbours, that is, the peers next to him in that ring. After that, rewire all the links created, with a certain probability, to a random peer in the ring. By inserting this randomness, it will become more likely to create shorter paths between peers on the opposite side of the ring, achieving a low clustering, which is the aim of the project, the less clusters there are, the faster information travels.

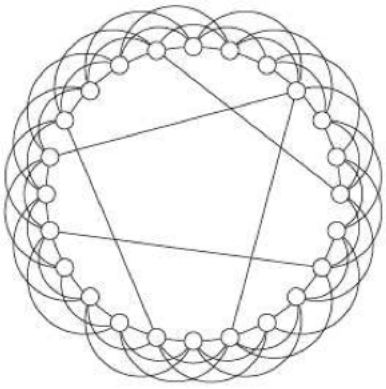


Figure 3.1: Small-world topology

3.3 Used Setup

After some research, the author came across the desired values for certain key-points in the implementation of random network coding.

First of all, in terms of finite field to be used, according to [SLL08], [Lea11] and [LWLZ10], the best finite field to use in these types of implementation is 2^8, since the probability of creating independent coefficients for each component of the symbol to send is very high. This probability is calculated by the inverse of the finite like so $1 - \frac{1}{2^8} = 0.9961 \rightarrow 99.61\%$. Besides this high-probability, by having coefficients from the 2^8 finite field, means that each one will occupy 1 byte, which depending on the number of blocks on a segment to be sent, can produce a small overhead.

To ensure that this overhead remained small, the number of blocks in a segment was defined as 5. Each of these blocks have the size of 1 kilobyte, which in turn corresponds to a packet sent under the $userdatagramprotocol(UDP)$. As stated in the previous section, the physical layer of the network will rely on the Internet, and therefore there is the need to think about which protocol to use. The segment being broken into blocks makes it easy to just transmit datagrams instead of an actual stream (as provided by *TCP*). Even though it is possible to ensure the transmission of messages through *TCP*, the separation needed to be done at the peer, contributing to more processing, burden and finally undesired delay. In the end, *UDP* is a great lightweight weapon to ally to network coding that supports multicast and does not need connection, message ordering nor traffic control, in random network coding encoded blocks are always being generated, there is only the need to wait for, in this case, five linearly independent blocks to arrive fully to decode the segment. In terms of scalability, it also performs well, seeing that it is stateless, meaning that it does not need to make a connection, and only receives or transmits.

In what concerns the small-world topology, there are two arguments to define, the mean degree of the topology and the rewiring probability, studied to detail in [SLL08]. Here it has been stated and proven that having a mean degree of six proved to be the best fit, even when the network is scaled, seeing that having too few neighbours may hinder the process of introducing new information and consequently the transmission of redundant blocks between peers; having too many peers can also be a problem in the case of peers with common upstream nodes receiving replicated information. As for the rewiring probability, it was also proven that a high probability will lead to an almost totally random topology with very low clustering and with a very low probability, it would form a near perfect network ring and subsequently high clustering, which would introduce redundancy, with peers sharing the same information as its neighbours. With this in mind, and also the degree of the topology, it was proven that a rewiring probability of 0.1 would be a best fit. Summing up, in average, each peer will transmit to six other peers in the network having a rewiring probability of 0.1.

3.4 Adopted Heuristics

To ensure fairness and to reduce the completion times of all peers a set of heuristics were created. These heuristics focus mainly on the selection of what (and what not) message to send, but also on who to send it to. The heuristics are as follows:

- **Permutation of the players to send**: each peer has a list containing the peers he is supposed to send messages to. When the time for a peer to transmit comes, he checks this list for any peer, if no peer is present, he will rebuild this list, but never in the same order, that is randomly deciding the order of the new list. By doing this the simulation is achieving fairness between peers and abstracts itself of the need of a ordering system, that would imply some previous evaluation of the network to determine an order;

- **Number of personal transmissions**: since each peer is inserted in the environment of a videoconference, it is only natural that it produces its own messages (i.e. his video and audio), so in order to assure that all the peers that desire his contents do not receive more block than needed for that segment during that segment step, the peer keeps a map of how many times it has transmitted its own contents to each peer. When he reaches the sufficient number of linearly independent blocks to decode the segment, which in this case is 5, it stops transmitting its own stream and retransmits contents he received. This way it is assured that no redundant blocks are sent, culminating in bigger completion times for the peers;

- **Retransmission weight**: not all the peers have the same bandwidth capacity, so in order to ensure that even the slowest peers are able to transmit their messages at a competitive time, a global mapping of peers and their bandwidth is kept over the network, and the mode of the present bandwidths is calculated (i.e. the most common bandwidth in the network). When deciding which message to retransmit, a peer checks all the messages he already has and according to the owner bandwidth, gives it a weight. Weights are given exponentially according to the inverse of the bandwidth size, meaning peers under 75% of the mode will have the biggest weight and peers with bandwidth of more than 125% of the mode have the smallest. After this, a weighted probability is calculated and the result will be which peer to retransmit;

- **Useless retransmissions**: retransmissions are useful, to disseminate messages from peers that do not have a big bandwidth size through peers with faster connection. Even so, there is a point that the number blocks of a certain peer is enough for retransmissions and the reception of more blocks from said peer will only contribute to the creation of more redundancy. To prevent this, there is also a threshold on the number of received messages on each step. When a peer receives more than the double of needed blocks per segment of some peer, it announces to the network that from that moment on, the latter is considered useless to him and no more transmissions/retransmissions of him should be send to him, leaving the opportunity of the reception of important blocks in its place.

3.5 Summary

This chapter focused on some important key aspects regarding the use of network coding on this specific scenario, that are compiled in table 3.1.

It established what kind of flavour of network coding to use, which in this case is random network coding and also set the baseline scenario of comparison of the project, the standard multicast version said project.

Topologically speaking, small-world topologies play a big part on the overlay topology of the network, achieving the desired level of abstraction and randomness desired in such a system, with an average degree of six and a rewiring probability of 10% . As for the physical layer, a fully connected topology will be considered, using the Internet for this means. As for key values in defining network coding, the chosen finite field was 2^8 which ensures a high probability of linearly independence of coefficients and also fits right into the UDP protocol, necessary for transfer over the Internet.

In order to maintain fairness and reduce the process times and redundancy a set of heuristics were defined, so a more realistic simulation is achieved. Peers never send messages to others in the same order, this order is always randomly selected. Also each peer never transmits his blocks more than he needs to (i.e. the number of transmissions to each receiving peer is equal to the number of blocks) and if he needs to retransmit, he decides which blocks to retransmit with a weighted probability (i.e. blocks belonging to slower peers are more probable to be retransmitted). Lastly, if a peer already has the double of the needed blocks to decode a segment, he announces that it is useless for other peers to send him more of those blocks, creating the opportunity of receiving more important/needed blocks.

Network Coding Type	Random
Physical Topology	Fully Connected
Overlay Topology	Small-World
Node Average Overlay Degree	6
Node Overlay Rewiring Probability	10%
Finite Field	2^8
Internet Protocol	UDP
Block Size	1024B
Number of Blocks in a Segment	5

Table 3.1: Compilation of key values for network coding setup

Chapter 4

Simulator

In order to test the proposed implementation at hand, there is the need for a simulation platform with recording capabilities, so that the runtime logs can be later analysed and compared. Previous work on this field made in [LBS10], has provided this tool, nonetheless some modifications were needed, although the environment of the simulation is totally different, the foundations are the same. Due to these changes, this chapter will start with a description of the initial architecture of the simulator, followed by the requisites of the new environment, the the additions/modifications to meet these requisites.

4.1 Architecture

The Java simulator has itself five pretty distinguishable modules: *control, monitor, network, overlay* and *scheduler*. These models are all interconnected achieving a level of abstraction needed for future transformations of it. This loose coupling provided the author of this report the necessary foundations for his goal, without having to let him worry about the implementation of the modules shared by both projects. The modularity of the simulator, through this abstraction and separation of concerns, made it a suitable option for the task at hand.

The scheduler module, root of all the simulator, is responsible for the sequence of events over the runtime. The simulator runs in discrete time and it is event driven, so for this purpose it possesses a *Clock* class that manages the passing of time and schedules the beginning tasks. These tasks are supposed, at the end of their runtimes, to schedule the following tasks and so the clock runs until no more tasks exist. When this happens, it means the simulation is over.

The monitor module is the part of the system that oversees the scheduler module and serves as an interface between it and the business logic of the system (i.e. the overlay module). It also serves as the bridge for the logging part of the simulator, writing all the simulation happenings in a set of organised files, keeping track of the completion times and the packets used by each peer in each step; individual records for each peer, keeping track of the received and sent packages, their times of happening and coefficients involved.

The control module, is in charge of dealing with the configuration of the business logic of the simulation, importing/exporting configuration files and presenting the information contained within these files in a pre-processed manner. To achieve this, descriptors for nodes and edges are created in order to act as placeholders for the information to be used at a later time. The purpose behind this is the abstraction and separation of concerns mentioned earlier, because by doing this it is possible to apply different types of business logics without having to re-implement the import/export process of the configuration files, that are business logic independent.

The network module, defines the physical layer of the simulation and is responsible for message delivery. Peers can join and leave the network created in this layer, and receive and send messages through it. This module emulates all the parameters of a network (packet loss, bit fault, latency, bit error and bandwidth), in order to make it more realistic during the simulation process (i.e. the transmission of messages between peers).

The overlay module is where the business logic is defined and it comprises all the possible variants of the process. Firstly, a package containing the implementation of linear coding, common to all the variants, is implemented. Here is where the definition of the finite field is set and all the mathematical procedures for encoding/decoding are present. Lastly, all the different testing cases (i.e. the variants of network coding, as well as the control group, broadcast in this case) are implemented. This way, during the simulation trials, there is only the need to specify which type of variant is being run, without having to resort to code modification.

4.2 Requirements

Although the existing simulator is a huge leap in the development of this project, it was meant for a different purpose, where the load to be transferred is not that high, and where all peers share everything with each other. Some requisites to the solution to be studied, require some modifications and in some cases addition of features to the simulator to make it suitable for testing.

One of the biggest differences, is the amount of data needed to be transferred. For this, a bigger finite field is needed to support such a constraint. As it was stated in the previous chapter, the adopted finite field is 2^8, which at heart is a binary finite field. When dealing with binary polynomials, the arithmetics to be used in the network coding have to be altered.

The baseline scenario to the present project is also a new variant of the testing cases, and in that order it needs its own implementation. As for the implementation of the network coding, essentially it is the same, with some workarounds to make it compatible with the new finite field and the insertion of the heuristics previously declared in the previous chapter.

The original iteration of the simulator lacked means of visually analysing the data gathered throughout the simulation, so there was a need to include a new feature that at the end of each runtime is able to depict it through the form of graphs was essential. Another feature included is the possibility of analysing and comparing to different simulation runtimes through the average of both.

As previously stated, one of the main requisites was the implementation of a parser of graphs, to a suitable format, readable by the simulator. Even though the requisite was rendered useless after the lack of a proper database of graph, this feature was already implemented at the time, so the solution passed by changing the requisite and create program that would create random configuration files according to desired values.

4.3 Configuration

In order to make the import/export process faster, a decision to transform the configuration files into *XML* format was made. *XML* makes the configuration files structured, more readable and prone future usage on other projects, when comparing to a simple text file.

The settled *XML* format in listing 4.1. Here the parameters that define the network are included in the *config* tag. Inside it, can be found the *peers* tag that contains all the peers that will be a part of the network to run. Each peer has as its values the parameters that define him and contains two sets of children, *edges* and *contentPeers*. While edges contain the physical connections to other peers, contentPeers contains the list of peers that the peers need to get information from.

Methods to interpret and create this new type of configuration file were added to the *Configuration* class, as well as the need for interpreters for the placeholder methods for peers and edges.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <config bitError="0.0" blockSize="1024" gameStepLength="10000" jitter="5"
       maxBandwidth="0" maxLinkDelay="50" minBandwidth="0" minLinkDelay="5"
       numberOfPeers="10" packetLoss="0.0" packetSize="300" segmentSize="5"
       totalGameDuration="200000.0">
3    <peers>
4      <peer bandwidth="15600.0" id="0" messageSendingDelay="15.482626741531053"
           sourcePeer="false" startDelay="0.0" startGamestep="0">
5        <edges>
6          <edge bitError="0.0" jitter="5" latency="145" packetLoss="0.0" to="3"/>
7          ...
8        </edges>
9        <contentPeers>
10         <peer from="9"/>
11         ...
12       </contentPeers>
13     </peer>
14     ...
15   </peers>
16 </config>
```

Listing 4.1: Example of *XML* configuration file

4.4 Binary Field Element

As a new class of the *linear_coding* package, *BinaryFieldElement* defines the new finite field to be used. Although a previous class for finite field already existed, due to the big difference in arithmetic, a new class had to be implemented, mainly due to the fact that for this new project, network coding will be dealing with binary polynomials, and their arithmetics.

In order to make the application runtime faster, it was decided that all the possible arithmetic values would be calculated beforehand and kept in maps, originating in three tables: addition, multiplication and division. This tables are then just accessed and the value retrieved, resulting in a much faster process time. To decrease the overhead of calculation, all the tables at the beginning of the simulation runtime, the tables are exported to external files, that then are imported at the beginning of the simulation, meaning that at most, these tables are only calculated once and then stored and kept for as long as the project exists.

4.5 Multicast Set

This package was created to implement the class that would serve as the baseline scenario for the testing. The *MulticastPeer* class, that extends from the more general abstract *NetworkCodingPeer* previously implemented, differs from the network coding counterpart, as it only decides what user to transmit to, based on his transmission list, and if the transmission number is lower than the number of segments needed, as can be seen through listing 4.2. If he finds a suitable peer he can then proceed to create the message to send, if not (i.e. if all the peers already received the exact number of messages as the number of needed blocks) the peer does not send anything.

The multicast peer also supports the eventuality of the existence of a collecting server, that is, a peer that receives information from others, but none of the others require his stream. With this kind of peer, scenarios where these kind of servers help the communication can be tested. In these scenarios, the collective peer receives information from all the peers and instead of transmitting messages from himself, it transfers messages important for the destination peer. With this kind of implementation it is considered that the server is aware of the multicast mappings (i.e. the server knows which peers are subscribed to every other peer). It is worth mentioning that on a network coding scenario, there is no need of implement such a server, since if one peer is not interested in the messages of the source, the source will retransmit, messages received by him, automatically.

```
1 do{
2   if (peersToSendMessageTo.size() == 0) {
3     determineMessageSendingOrder();
4     boolean stop = true;
5     for(Integer peer : peersToSendMessageTo){
6       if(messagesSent.get(peer) < segmentSize){
7         stop = false;
8       }
9     }
```

```
10      if(stop){
11        return null;
12      }
13    }
14    destination = peersToSendMessageTo.firstElement();
15    peersToSendMessageTo.remove(0);
16  }while((messagesSent.get(destination) >= segmentSize
17      || !overlayTopology.get(destination).contains(creator))  && !receiverPeer);
```

Listing 4.2: Method of choosing to which peer to send on a Multicast Scenario

4.6 Network Coding Set

Although this package already existed, it needed some modifications in order to support the new finite field class. After the implementation of a simple network coding peer, it was decided to go further and create a better version, and heuristics started to be implemented. To verify if these heuristics were beneficial for the simulation, the previously existing class was made abstract and two others were created, one called *NetworkCodingNormalPeer* with the purpose of serving as a comparison to the other one, the *NetworkCodingWeightedPeer*.

To make it easier to change between implementation without having to resort to code alteration, Java reflection capabilities were used, and depending on the arguments provided by running the simulation, the correct instance would be created.

On the *NetworkCodingWeightedPeer* class, all the heuristics were applied. These heuristics can all be seen during the decision of which peer and what to transmit. Once the destination is known, by getting it from the permutation vector, it is checked whether there is a peer that is on the transmission route of the destination which needs the creator of this message (listing 4.3), this way the probability of creating useful retransmissions is bigger. If it happens a flag is set to on.

```
1  boolean needed = false;
2  for (Vector<Integer> peer : overlayTopology.values()) {
3    if (peer.contains(creator) && peer.contains(destination)) {
4      needed = true;
5    }
6  }
```

Listing 4.3: Loop to check the usefulness of the message for retransmission purposes

Afterwards there is the necessity to check if the creators of any of the received messages are not considered useless by the destination peer (listing 4.4).

```
1  int retransmissionable = 0;
2  for(Integer p: receivedMessages.keySet()){
3    if(p != destination && !uselessPeerRetransmission.get(destination).contains(p)){
```

```
4       retransmissionable++;
5     }
6   }
```

Listing 4.4: Loop to check if any of the received messages are useful for retransmission

When these verifications are done it is time to check whether the peer will send a message of its own or retransmit something he already received. First he checks if there is no need for transmitting something of its own, if there is not and a retransmission is possible, he will do it, otherwise he just stops. After that, it is checked whether or not there is a change for retransmission of a packet from itself or if with it reached the number of blocks needed to linearly decode the segment. If this is not the case, the peer's bandwidth is tested, meaning that if his bandwidth is higher than 125% of the network bandwidth's mode then there is a probability it will retransmit 80%. In all the mentioned cases of retransmission, the decision of choosing the messages to retransmit follow the weighted probability mentioned in the previous chapter.

```
1   if(notWorthTransmit(creator, destination)) {
2     if (worthRetransmit(destination)) {
3       creator = getNewCreator(creator,destination);
4     } else {
5       return null;
6     }
7   } else {
8     if (messagesSent.get(destination) >= segmentSize && worthRetransmit(destination))
          {
9       creator = getNewCreator(creator,destination);
10    }
11
12    else if (bandwidth > (bandwidthMode * 0.75) && worthRetransmit(destination)) {
13      double rand = Math.random();
14      if (bandwidth > (bandwidthMode * 1.25)) {
15        if (rand > 0.19) {
16          creator = getNewCreator(creator,destination);
17        }
18      }
19    }
20  }
```

Listing 4.5: Method of choosing to which peer content to send on a Network Coding Weighted Scenario

4.7 Graph Drawer

The lack of some visual aid to analyse all the information gathered, resulted in a search for a free graphic library that could provide different types of graph, specially embedded in the same

window. This search lead to the discovery of *JFreeChart*.

JFreeChart is a free software, licensed and distributed under the terms of the *GNU Lesser General Public Licence (LGPL)*, that provides all the features required without the need of re-implementing the modules already produced. In the following chapters, its flexibility and graphic capacity will be in display, through the graphs of the simulation tests.

With the help of this library, the *Graph Drawer* package was created with two distinct classes: the *ComparisonDrawer* and the *PeersGraphDrawer*. The former, as its name suggests compares simulations through the averages calculated during each step of the simulation. The latter, which normally runs at the end of every simulation, shows the performance and behaviour of each peer, as well as the average, during the simulation.

Both classes provide the user the same type of graph, a window divided into two types of graphs: a bar chart, containing the number of messages received by each peer, or the average, or both throughout the steps of the simulation; a line graph, stating analogously the times, in milliseconds.

4.8 Graph Parser

The parser, built at the beginning of the project's development, is an independent application that can take different types of files. It can create a configuration file from a *GraphML* file, another configuration file or just by indicating the number of peers we want in the network.

In order to achieve the construction of the graphs needed for the simulation process (i.e. the fully connected graph for the physical layer and the small-world graph for the overlay business logic), the *Builder* class uses two java libraries, *JUNG (Java Universal Network/Graph)* and *Agape*. They both provide great graph algorithms and graph theory heuristics that are able to provide the needed types of graphs for the simulation. They also provide a system to visualise the graphs created, in order to cross-check what was built.

The parser package is in charge of parsing from and to all the supported types of the application. When parsing from, the application is capable of parsing *GraphML* files, which are a structured *XML* file, created for supporting the storage of graphs. The information contained in the file, is then translated to object of the form of *nodes* and *edges* and put into a *JUNG* graph. If the case is parsing from an already existing configuration file, the process is analogous, but it is useful to build new small-world topologies for the already existing networks.

When the task is to parse to the configuration files readable by the simulator, there are two possibilities, the new version, *XML* configuration file, but also it has support for the old text configuration file, used by the old simulator.

Simulator

Chapter 5

Simulation

5.1 Modus Operandi

To start the runtime, there is the need to provide the simulator with three arguments: the type of simulation to run (i.e. multicast, normal network coding or weighted network coding), the directory to host the resulting log files from the simulation, and the location and name of the configuration file. The simulator then creates a folder to host all the files regarding the simulation with a unique name, so there is no risk of duplicating simulations.

To start the simulation, the simulator first needs to import the information contained in the configuration file, and for this purpose it uses the configuration class of the control class mentioned in the previous chapter. Once all the information is loaded, the simulation controller creates all the entities needed for the simulation to run (i.e. the peers, its sockets, connections and first events) and as soon as everything is all set the simulation starts to run. It is relevant to say that while the controller is setting up, the binary field tables are also loaded, without them no messages can be built and with this the simulation will not run.

During the actual simulation, monitors control all the transmission process and have the duty of writing the information in the log files and also in the console so the users have a certain perception of what is happening. These transmissions contain the the source, the destination, the creator of the message and the coefficients of the message, not containing actual data. For simulation purposes there is no need for exchanging actual data, but just the coefficients to build and decode the matrix, through gaussian elimination.

When one event is occurring, it schedules the next event in time, this discreet method of event handling simulates a possible timeline as depicted in figure 5.1 as the best approach to a real-world scenario, where the simulation would follow a continuous timeline. At the end of each step in the simulation (i.e. transmission of a segment), the times needed for transmission and the number of received packets are written in both outputs.

The moment the simulation is over, a graph pops up containing the number of packets received and the time needed for completion per peer per step. This is also accompanied by the average of the simulation.

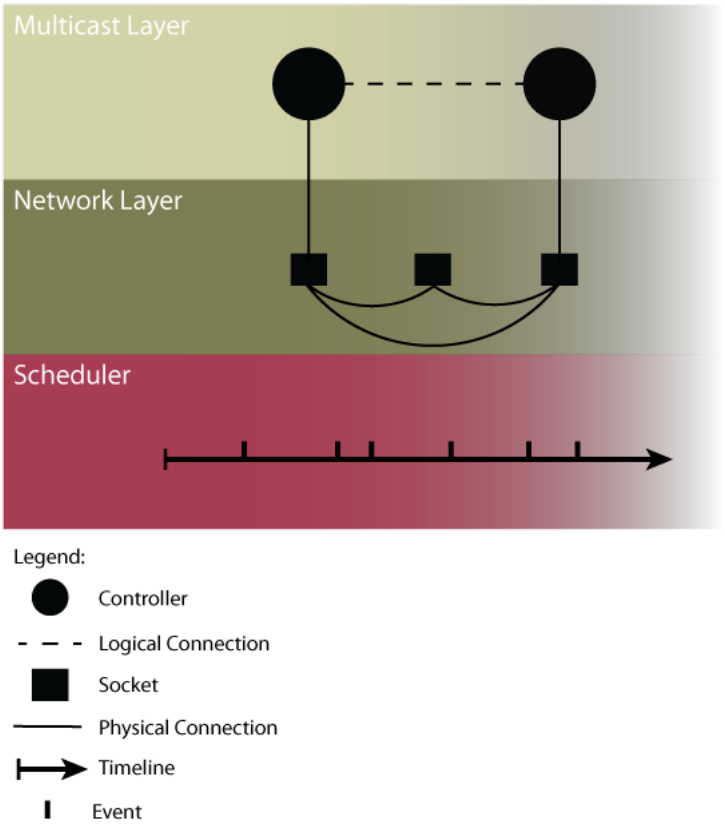


Figure 5.1: Depiction of the simulator during the simulation

5.2 Testing Scenarios

In this section, the testing scenarios to be described are two: a ten peer scenario and a twenty peer scenario. Both scenarios are composed of different types of peers (i.e. peers with different bandwidths and delays).

The scenarios are tested in different variants, being the following:

- Simulation over Wi-Fi;
- Simulation over 3G;
- Simulation over 3G with a server collecting all information and broadcasting retransmissions to all peers.

On the ten peer scenario there are two additional tests: a fully connected overlay (i.e. each peer receives messages from the other nine peers, the defined upper bound for number of incoming peers) and a mix scenario, where in both clusters there are Wi-Fi and 3G peers at the same time. It is worth mentioning again that these simulations are run twice, one over a traditional multicast environment and the second one using the developed network coding scenario.

5.2.1 Ten Peer Scenario

The ten peer scenario is composed by two clusters (i.e. two groups of peers), one situated in *Porto*, *Portugal* and the other one in *Budapest*, *Hungary*, as it can be seen in figure 5.2. The reason for creating a pair of clusters separated by a considerable distance, is to test not only the performance of peer communication in short distance (i.e. inside the cluster) but also test the performance of the communication when additional delay is added, in this case the transmission delay from two countries in Europe. Also present in figure 5.2 are the delays, in milliseconds, between peers inside the clusters and when communicating between clusters. Two delay values are displayed, the delay over Wi-Fi and delay over a 3G connection.

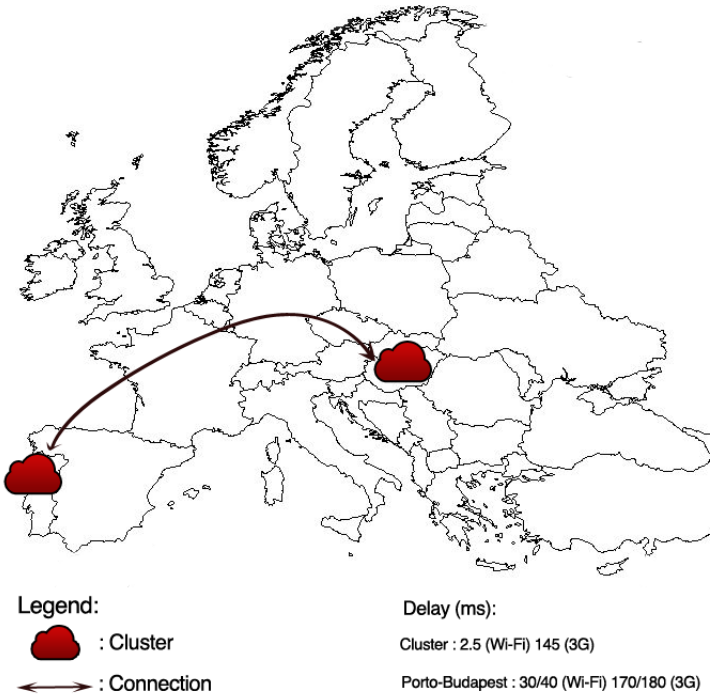


Figure 5.2: Depiction of the ten peer test scenario

The overlay small-world topology used for test is present in figure 5.3 and provides a clearer idea of how the network is connected. Peers belonging to Porto's cluster are numbered from zero to four, the rest belonging to Budapest's cluster. Within this testing scenario, as it can be seen in the figure, on average, each peer requires the transmission of three other peers and analogously transmits, on average, to three other peers, regardless of its bandwidth size.

5.2.2 Twenty Peer Scenario

On the twenty peer scenario, a cluster is added to the previously described ten peer scenario, situated in *San Francisco*, *United States of America* (Figure 5.4). The location was chosen in order to test a transatlantic connection, to one of the furthest places of the other cluster. The delay added by the distance is far greater, so it makes it a key scenario to understand if the application

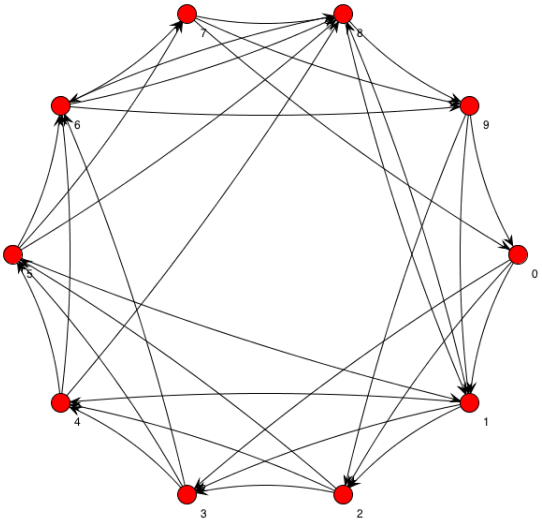


Figure 5.3: Ten peer scenario overlay small-world topology

will be able to withstand long distance videoconferences. Once again, akin to the ten peer example, figure 5.4 displays the communication delays of the scenario.

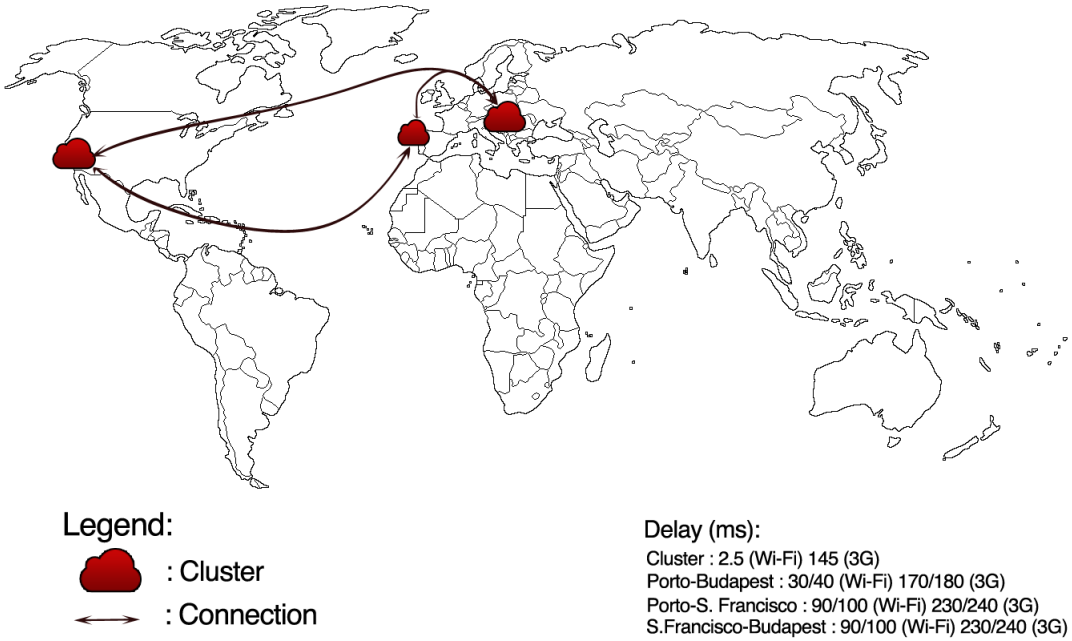


Figure 5.4: Depiction of the twenty peer test scenario

The overlay small-world topology used in the simulation tests can be seen in figure 5.5. In this figure the formation of a ring is more perceptible, characteristic of small-world topologies and some rewiring cases become more noticeable. Peers from zero to six belong to Porto's cluster, from seven to thirteen to Budapest's cluster and the remainder of peer belong to San Francisco's cluster. In this testing scenario, the number of incoming/outgoing transmission is increased to, on

average, six, hereby achieving the optimal number of connections defined beforehand.

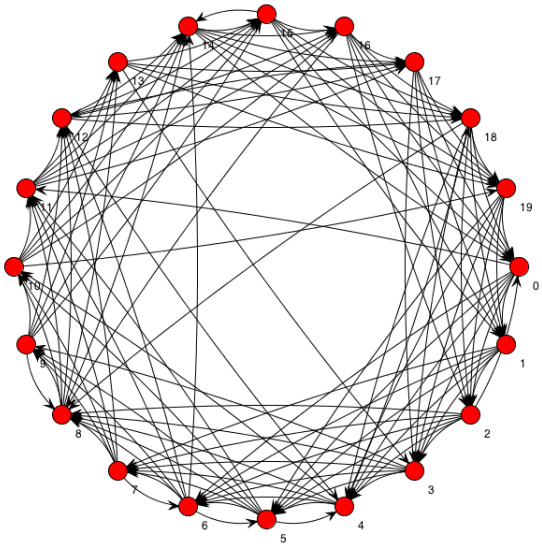


Figure 5.5: Twenty peer scenario overlay small-world topology

Chapter 6

Results and Evaluation

6.1 Evaluation Parameters

In order to understand the way performance varies, there are two key results to study: *block redundancy* and *distribution time*. Block redundancy is the quotient between the number of blocks needed to successfully decode the needed segments and the actual number of received blocks. Distribution time is the time each peer needs to complete a simulation step, from the first initial transmission of information until the last decoded segment needed.

6.2 Simulation Over Wi-Fi

As it was presented on figures 5.2 and 5.4, the biggest hindrance of the system is not the delay caused by the Wi-Fi transmission, but the actual bandwidths of the intervenient peers and their connections. The greater the demand of stream messages from a slow peer, the longer it will take the ones requiring the segment to finish receiving the segment on its whole. By applying network coding, with weighted probability of retransmission, the author expects to render this hindrance to its lowest level, seeing that all peers will be able to transfer it, which will culminate on a performance boost when comparing to the baseline scenario.

6.2.1 Ten Peer Scenario

According to figure 6.1, the network coding approach performs far better than its multicast counterpart, in terms of average distribution time, achieving an average of $105.59ms$ per step, against $226.43ms$ over a normal multicast. This means a decrease of 53.37%. This means that over network coding using solely Wi-Fi, as much as $9segment/second$ can be sent, whereas over traditional multicast only $4segment/second$ are possible. When framing these results on the scenario, a communication between peers from two cities within Europe, both scenarios provide good results, being able to maintain a connection of $160kbps$, and, in the case of the network coding, the quality of the service could be improved by double and still have a segment of margin.

The average block redundancy on this test scenario is approximately 3.68, seeing that the average block reception for network coding is 58.78, whereas over multicast is only 15.99. Even though this redundancy is high, the system performs better, pointing that this redundancy can be beneficial to the system.

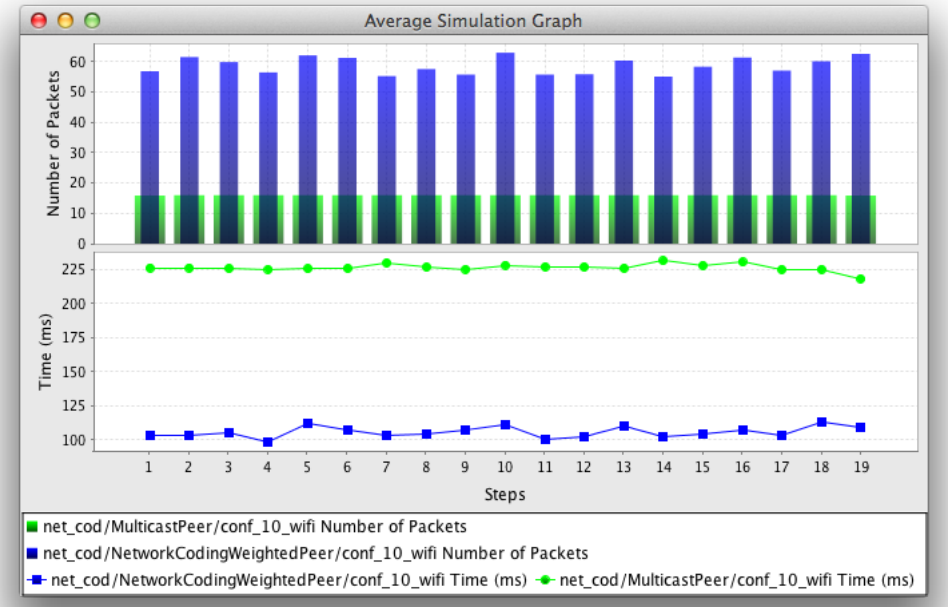


Figure 6.1: Results of the ten peer test simulation over Wi-Fi

6.2.2 Twenty Peer Scenario

On the twenty peer scenario test, presented by figure 6.2, the network coding case performs even better than in the ten peer scenario. It achieves an average of 220.53ms of average distribution time, which comparing to an average of 508.62ms achieved over a normal multicast, translates to a decrease of 56.64%. In this case, over network coding using solely Wi-Fi, as much as 4$segment/second$ can be sent, whereas over traditional multicast only 1$segment/second$ is possible. Under these results, when trying to achieve a transatlantic communication between peers, multicast is not able to provide a suitable throughput, while network coding does maintain the 160$kbps$.

The average block redundancy on the twenty peer test scenario is approximately 3.32, with an average block reception for network coding of 103.57, and of 31.24 for standard multicast. It can be said that redundancy found is similar to the ten peer scenario, suggesting that it is not the distance nor the increasing of number of incoming streams, cause for the redundancy provoked by network coding.

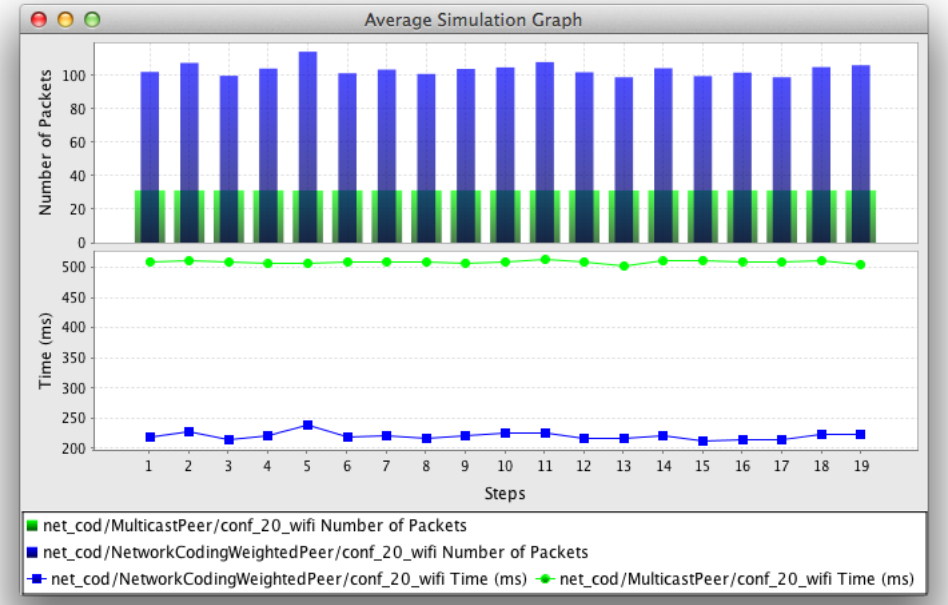


Figure 6.2: Results of the twenty peer test simulation over Wi-Fi

6.3 Simulation Over 3G

When considering a mobile application for smartphones, one has to consider the use of mobile internet, such as 3G. Even though faster standards for mobile Internet exist, with the introduction of 4G in various countries, such as Portugal or Hungary, this standard is not yet widespread worldwide and so the decision of testing under 3G had to be made. In the future, further tests considering 4G networks may be carried, seeing that recent studies show that delay decreases in more than 50% when comparing 4G to its predecessor 3G.

As can be seen by figures 5.2 and 5.4, the biggest challenge when testing over a 3G network is the delay brought by the standard. Even inside a cluster, the delay caused by transmitting a message between peers is around 145ms, more than the delay of a transmission Budapest-San Francisco over Wi-Fi. In order not to increase this value even more, through cell transmission, when communicating outside the cluster, messages go from the reception cell tower, to a modem connected to the internet, are delivered to the provider's modem, sent to the tower and finally to the phone, this way only adding the Wi-Fi delay to the inter-cluster communication. Through all this, the question raised is whether or not it is feasible to have the application running on mobile internet or not and the following tests attempt to shed some light on this question.

6.3.1 Ten Peer Scenario

By analysing figure 6.3, it can be concluded that once again the network coding approach performs better than its multicast counterpart, in terms of average distribution time, the former achieving an average of 306.16*ms* per step, while the latter achieved 339.88*ms*. This translates to a decrease of 9.92%. Although this decrease does not compare to the decreases seen on the Wi-Fi test scenarios, as much as 3$segment/second$ can be sent over a network coding scheme, whereas over traditional multicast only 2$segment/second$ are possible. Nonetheless, both solutions would require a drop on the quality of transmission, in order to accommodate the application to a scenario over 3G, exclusively.

The average block redundancy on this test scenario is approximately 7.79, since the average block reception for network coding is 120.81, whereas over multicast it is only 15.5. This redundancy may be due to the bigger delays found between peers, seeing that since a message takes longer to arrive, the peer stays active longer, which then culminates in more peers transmitting to him. Each peer ends up receiving too many redundant packets that in Wi-Fi conditions would not have been sent.

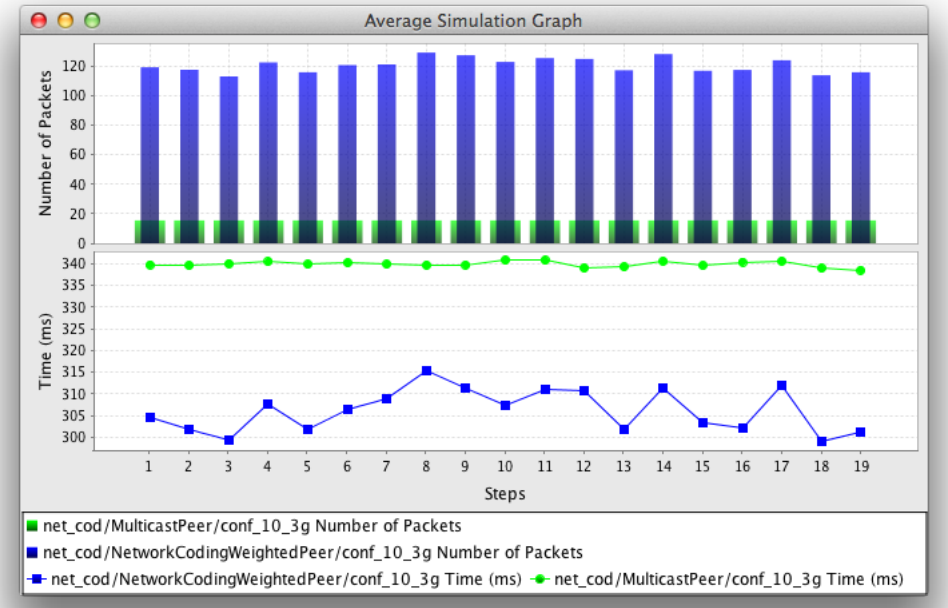


Figure 6.3: Results of the ten peer test simulation over 3G

6.3.2 Twenty Peer Scenario

When introducing ten more peers and a transatlantic cluster, presented by figure 6.4, the network coding case, when compared to the baseline scenario, has a better performance when regarding

the previous subsection. However this improvement is mostly due to the higher rate of degradation of the multicast scenario when compared to the rate of the network coding. The latter achieves an average of 426.60ms of average distribution time, while the former reaches 669.52ms, meaning an improvement of 36.28% over the standard protocol. Still, these improvements do not manifest themselves due to an improvement of the distribution times of the network coding solution, seeing that both solution are degraded with the added burden, network coding using solely 3G, can only transmit as much as 2$segment/second$ and traditional multicast only 1$segment/second$. Like the previous test, it is not possible to maintain the network working on the desired conditions, to do so a degradation of the quality of the service is needed.

The average block redundancy on the twenty peer test scenario is approximately 4.83, with an average block reception for network coding of 147.24, and of 30.49 for standard multicast. Once again, the additional peers and delays did not have a big impact on the average block reception over network coding. In fact the quotient improved, continuing to point out to the speculation created on the previous section that neither the distance nor the increasing number of incoming streams, are a cause for the redundancy present on the network coding solution.

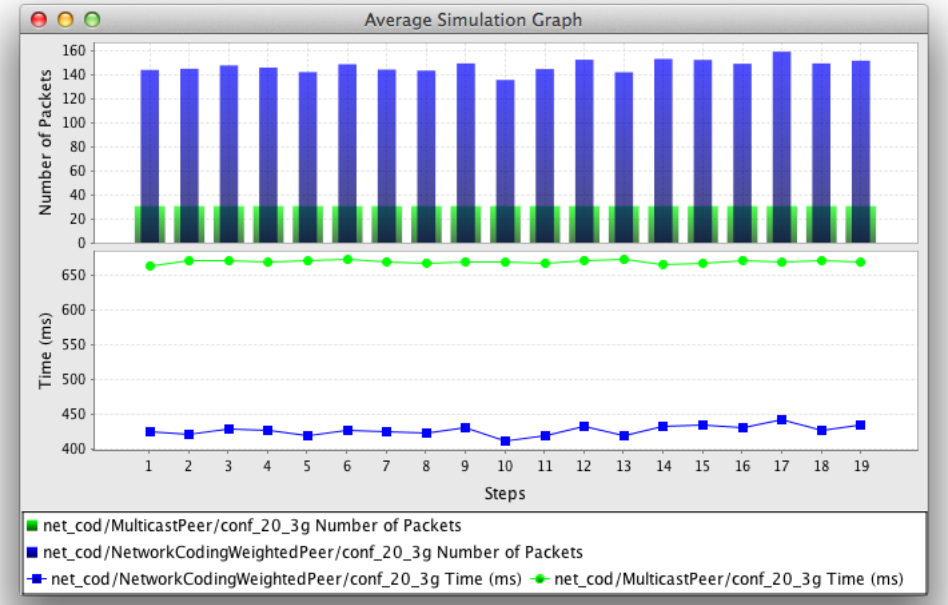


Figure 6.4: Results of the twenty peer test simulation over 3G

6.4 Simulation Over 3G With Collecting Server

The results of the previous section, showed that to support an exclusively 3G network, some adjustments had to be done to the quality of the service, this way decreasing the QoE. In order not

to do so, solutions to bring down the distribution time must be thought of.

By applying a collecting server that gathers messages from all peers and also transmits to all peers, it may be possible to bring this value down, since the delay to transmit to the server is half of the delay to transmitting to another peer, and also because this reduced delay is also present in the retransmissions made by the server itself.

6.4.1 Ten Peer Scenario

When analysing figure 6.5, it is concluded what has been concluded so far, and that is that network coding performs better than a standard multicast, in terms of average distribution time. It achieves 294.05*ms* per step, which comparing to the 321.15*ms* results in an improvement of 8.44%. Both solutions can transmit an upper bound of $3 segment/second$.

On the other hand, the average block redundancy is approximately 5.83, finding an average block reception of 109.63 on the network coding solution, and an 18.81 on the baseline scenario. Although this value is still higher than the one observed during the Wi-Fi tests, it is possible to notice a decrease of 25.26% in redundant blocks.

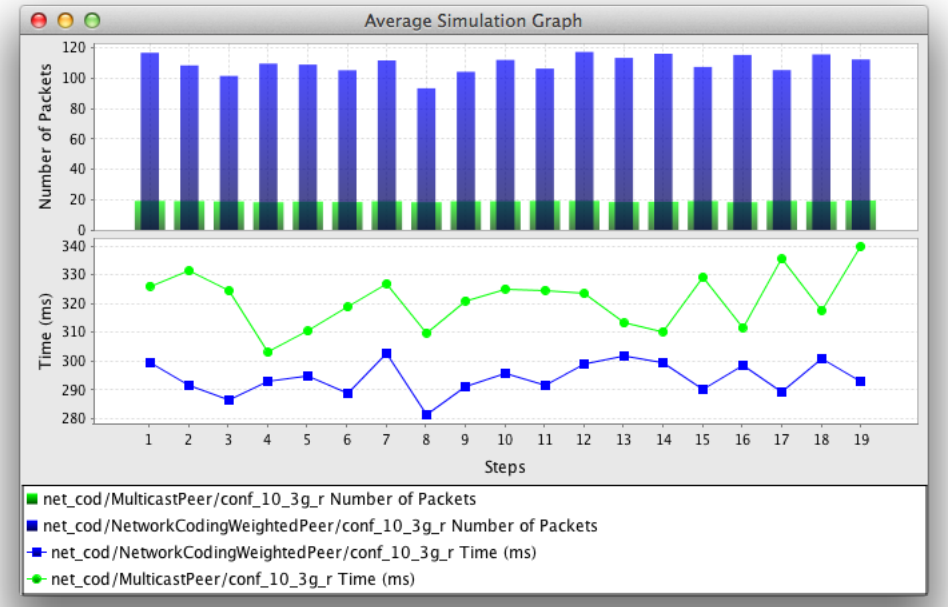


Figure 6.5: Results of the ten peer test simulation over 3G with collecting server

When comparing both network coding solutions from this section and the previous (Figure 6.6), it is undoubtable that the use of a server helps increase the performance of the overall network. This performance boost is more noticeable on the decrease of redundant messages than on the time of distribution of said messages. However, even with the server, it is still only possible to transmit $3 segment/second$ on the network under these circumstances. So the question of

whether or not to implement the dedicated collecting server has to be answered by the question of whether the profit to lower the redundancy is bigger than the implementation of the server.

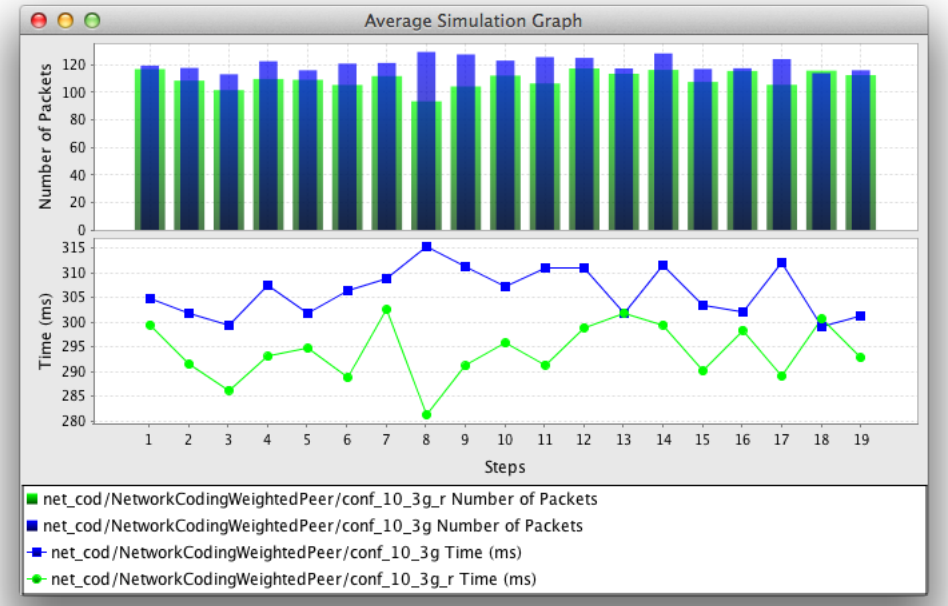


Figure 6.6: Comparison between the use or not of collecting server on ten peer network coding scenarios

6.4.2 Twenty Peer Scenario

As for the analysis of figure 6.5, it is concluded what has been concluded so far, and that is that network coding performs better than a standard multicast, in terms of average distribution time. It achieves 413.44*ms* per step, which comparing to the 646.96*ms* results in an improvement of 36.09%. This decrease in improvement is due to the fact that the use of a collecting server has more impact on the baseline scenario than it does on the network coding scenario being studied.

When it comes to the average block redundancy, its value approximately 3.82, with a network coding average block reception of 142.26, and a 37.27 of multicast average block reception. When compared to its server-less counterpart, it is possible to notice, once again an improvement of 20.91% in redundant blocks and it is also possible to confirm that the redundancy level came down to the levels observed during the the Wi-Fi simulations.

If the comparison between both network coding scenarios (i.e. with or without collecting server) is made (Figure 6.8), the improvements of using the server are also visible. But still the same question remains, of the worthiness of a server setup, since none of the improvements were in such a scale that would make the decision of applying it a trivial one.

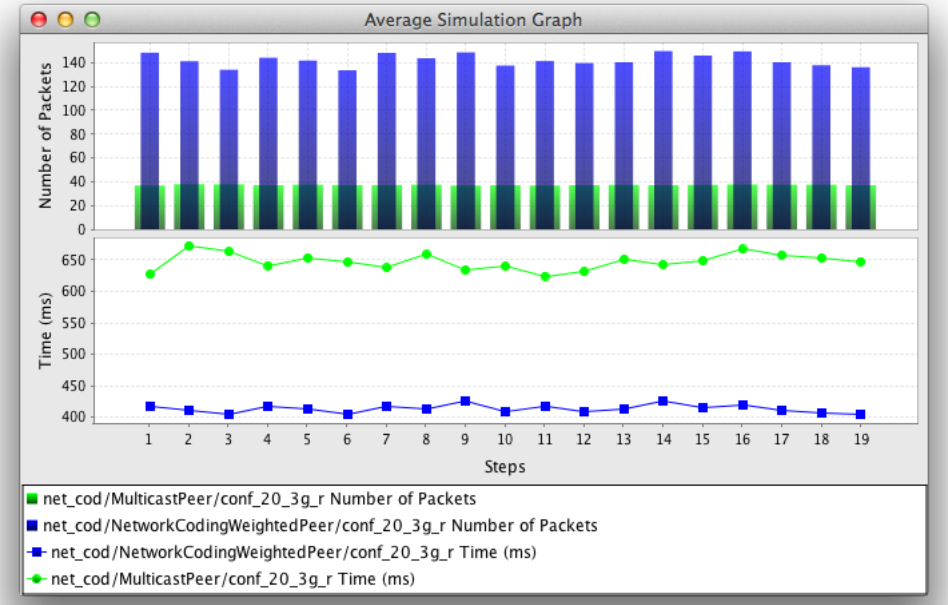


Figure 6.7: Results of the twenty peer test simulation over 3G with collecting server

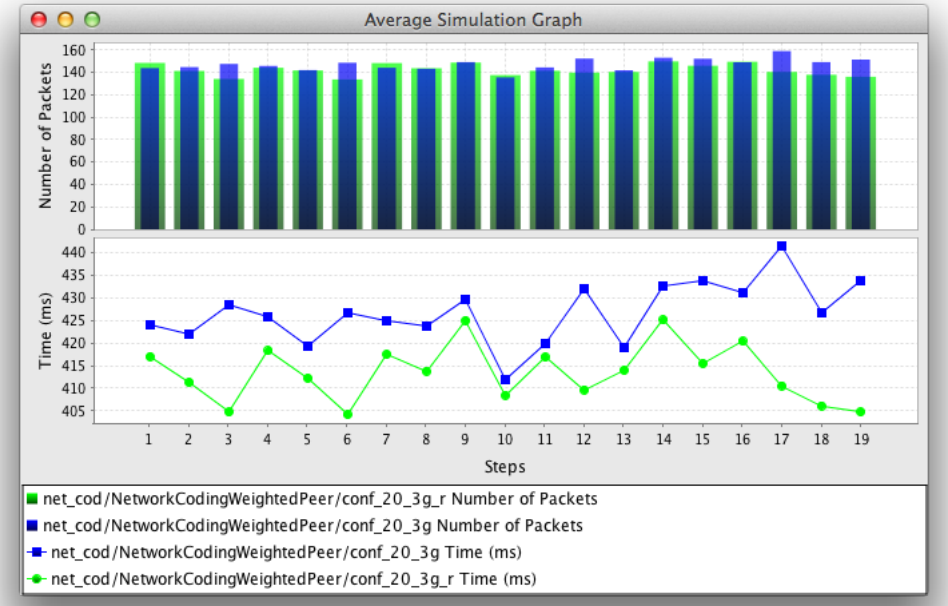


Figure 6.8: Comparison between the use or not of collecting server on ten peer network coding scenarios

6.5 Simulation Over Mix Network

So far, the tested scenarios only involved exclusive network (i.e. just Wi-Fi or just 3G). In a real-world environment, in an optimal scenario users could either be connected via Wi-Fi or 3G. Although these restrictions are not an obstacle that would render this project useless (e.g. *Apple's FaceTime* software for video-calling, up to the day of the writing of this report, requires a Wi-Fi connection to work on their mobile devices - the iPhone), but if it is proven that network coding can withstand the mixture of both internet access, it would prove an improvement over commercial applications.

In this test scenario, users simulate connections over 3G and Wi-Fi, this way there is a way to test 3G to 3G, Wi-Fi to Wi-Fi and 3G to Wi-Fi, inside and between clusters and study if the network is able to support the transmission of data needed for the future application of video-conference software under these conditions.

When analysing figure 6.9, it is clear that network coding performs better than a standard multicast, as a matter of fact a result that has been omnipresent throughout all the tests done, when concerning the average distribution time of each step of the simulation. The network coding solution achieves an average of 182.70ms per step, while the standard multicast only achieves an average of 282.24ms which translates to an improvement of 35.27%. Also on the network coding solution, as much as 5$segment/second$ can be sent, making it possible to support the stream of data thought for the future application, whereas over traditional multicast only 3$segment/second$ are possible, not supporting the desired kbps for stream.

Concerning the average block redundancy, the found value is 5.80, finding an average block reception of 92.72 on the network coding solution, and a 16 on the baseline scenario.

6.6 Fully Connected Simulation

To test the worst case scenario, it is necessary to create a network where all peers are connected to each other, since the defined upper bound of incoming streams was set to nine, this means that the worst case scenario is a ten peer network, all communicating to each other, this way filling all the channels of the network with messages. This test serves to ensure, in case it proves to be worthy of mobile implementation, that the solution achieved in this project is capable of reaching and handling the said upper bound. This test was done under the previously tested scenarios: using Wi-Fi or 3G exclusively.

6.6.1 Over Wi-Fi

Under this test (Figure 6.10), network coding achieved the best performance improvement logged until now relating average distribution time, reaching an improvement of 82.47% over standard network, since it registered an average of 144.07ms while the latter registered 821.69ms. The results registered show that under a network coding scenario, a maximum of 6$segment/second$

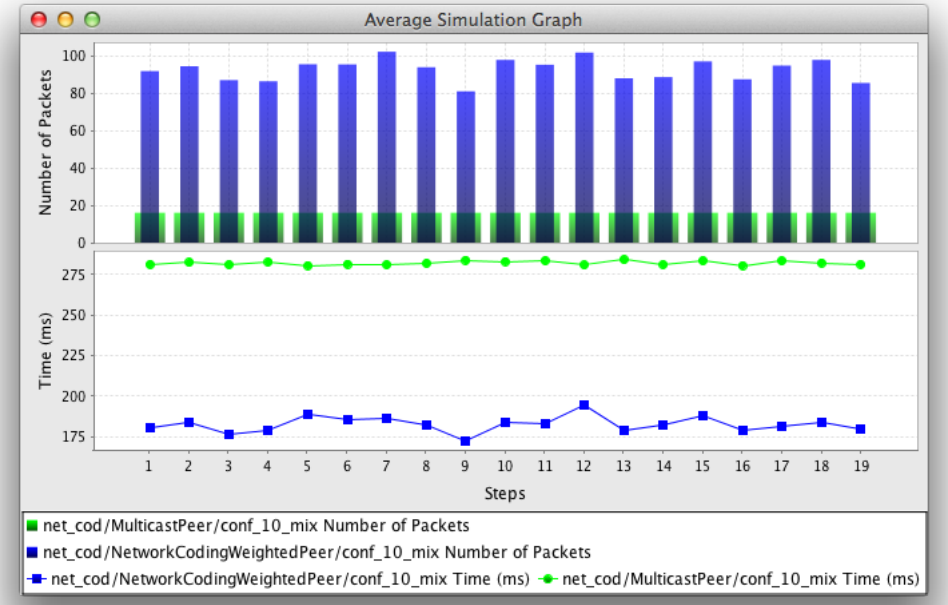


Figure 6.9: Results of the ten peer test simulation over mix network (3G + Wi-Fi)

can be sent, fully capable of withstanding the burden of transmission needed, while the standard multicast can only send $1 segment/second$ at best, far from the needed number of segments.

Also under this test, the average block redundancy is the lowest registered, being approximately 2.12. Network coding, on average, receives 95.13 blocks per step while the baseline counterpart only receives 44.98.

6.6.2 Over 3G

The 3G scenario, did not report such a big improvement as its Wi-Fi counterpart, but still, network coding, as recorded in all the tests run, proved once again to be faster than the baseline scenario. In this specific test (Figure 6.11), network coding reaches an average of $412.99ms$ while multicast reaches a much higher value, reaching $969.26ms$, on average. Even though this means there is a decrease of 57.39%, it is, still, not good enough to support the value of $160kbps$ seeing that at most it can transmit only $2 segment/second$. The multicast scenario performs even worse only being able to send 1 segment each second.

The average block redundancy registered in this test is of 4.30. This is due to the fact that the average block reception per step over network coding is 193.35 while under multicast is only 45. Like in all the previous tests, even though network coding inserts a big redundancy, it still performs better in terms of distribution time.

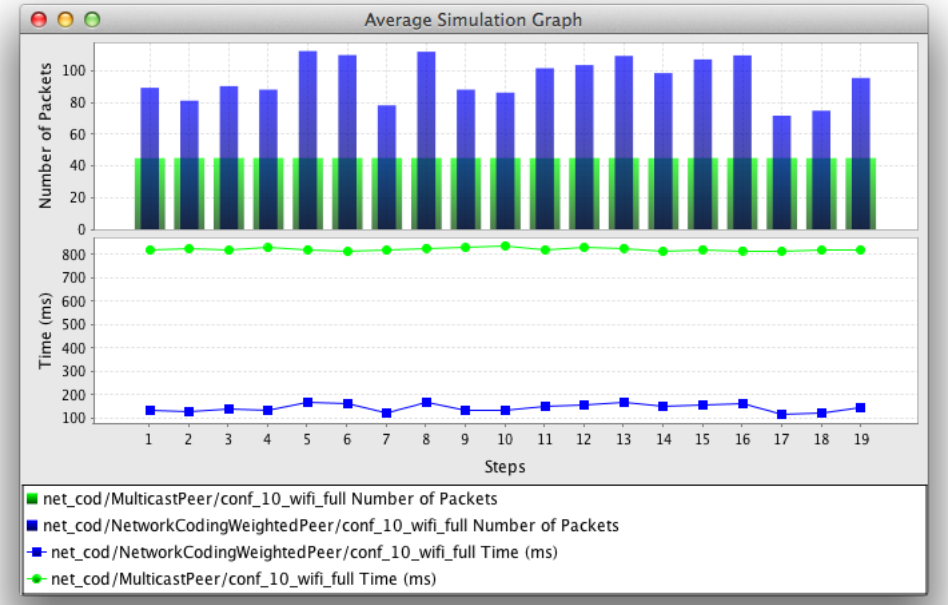


Figure 6.10: Results of the fully connected network test simulation over Wi-Fi

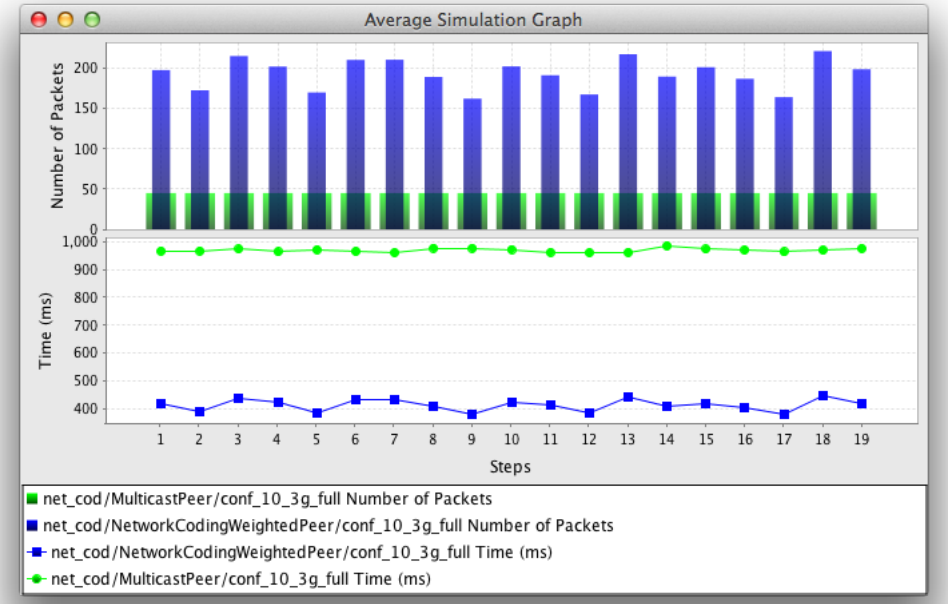


Figure 6.11: Results of the fully connected network test simulation over 3G

6.7 Summary

In all the tests carried throughout the development of this dissertation project, one key aspect was always present: network coding always performed better in terms of distribution time despite the fact it added more redundancy to the simulation environment. After all the tests, it can be stated that some redundancy may be beneficial for the overall distribution of the system, seeing that peers can retransmit messages that do not belong to them, where these messages may be of interest to their downstream peers. On the multicast scenario, the defined baseline, there is no retransmission, and this may be the source of such slow distribution times, since slow peers do not have any help from their receiving peers and this leads, in turn, to a longer waiting for new messages from said peers, rendering the network inefficient.

To sum up, table 6.1 provides an overview of all the tests done. Wi-Fi is clearly, as predicted, the preferred optimal environment to establish such a coding standard. It provides the needed throughput even when mixed with some 3G users. In terms of block redundancy, Wi-Fi scenarios demonstrated the lowest quotient of all tests, however, from these tests, it is possible to say that there is no direct relation between redundancy and distribution time, but from the information gathered by comparing network coding situation to standard multicast ones is that the existence of redundancy helps the system to bring down the distribution time, as previously stated. There was no need to test a scenario over Wi-Fi with a collecting server, since the delay times are not that long, when compared to a 3G environment, and thus being the only reason of implementing a server, to reduce the delay, it is not worth testing this scenario.

3G scenarios, although it proved that network coding performs better the the baseline scenario, still need some improvement to sustain a video-conference involving multiple peers. As has already been mentioned, many commercial application present in mobile phones nowadays still need a connection to Wi-Fi to keep up a stream between users, so this proved to be a good rehearsal in terms of performance comparison, but it is still not advisable to develop an application that would only resort to 3G exclusively. An alternative, as pointed out by the tests, is mixing up between 3G and Wi-Fi, so users that have the chance to use the application under a Wi-Fi network should do so, where this proved to be successful. Another alternative is to test the performance behaviour under the new 4G networks that begin to appear in various countries and mobile carriers.

		Network Coding		Multicast		Distribution Time Improvement	Block Redundancy
		Time (ms)	Packets	Time (ms)	Packets		
Wi-Fi	Ten Peers	105.59	58.78	226.43	15.99	53.37%	3.68
	Twenty Peers	220.53	103.57	508.62	31.24	56.64%	3.32
3G	Ten Peers	306.16	120.81	339.88	15.5	9.92%	7.79
	Twenty Peers	426.60	147.24	669.52	30.49	36.28%	4.83
3G with Collecting Server	Ten Peers	294.05	109.63	321.15	18.81	8.44%	5.83
	Twenty Peers	413.44	142.26	646.96	37.27	36.09%	3.82
Wi-Fi + 3G	Ten Peers	182.70	92.72	282.24	16	35.27%	5.80
Fully Connected	Wi-Fi	144.07	95.13	821.69	44.98	82.47%	2.12
	3G	412.99	193.35	969.26	45	57.39%	4.30

Table 6.1: Compilation of test results

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis work proved to be such a mind opening and fascinating theme, since it brought together so many areas that were subject of study in the past, but also it brought something completely new to the table, the opportunity to do some actual research, which until now the author had not experienced, during his academic path. The grandeur of the task itself, by being affiliated to such big entities, and the possibility of the publishing of a paper and its presentation in a possible conference proved also to be challenging motives for the semester that is ending at the time of writing.

Concerning the technique, it is concluded, after testing and evaluating its potential that network coding is a huge opportunity to do research and develop upon, where right now researchers just started to look through its pinhole and realised the untapped possibilities that lie within it. Since it encompasses many fields of study, it makes it a perfect candidate for academic research, not only from a master's point of view, but beyond it, where the author believes there are chances to better exploit all these new fast and broadband resources such as faster internet connections (standard and mobile) and the ever growing bandwidth capabilities of devices, specially mobile devices.

Since the methodologies used in the thesis work were already well defined (i.e. programming language, experimental setup and the evaluation parameters) the dissertation project took a step more towards an almost purely research and development project. Of course some modification intrinsic to the project's core purpose had to be done to the experimental setup (e.g. the finite field in use), but mainly the time spent by the author was focused on the research of the best key values through the reading of many scientific papers and the subsequent implementation of these values and testing.

The project tried to apply network coding to situations found on standard scenarios nowadays (i.e. using Wi-Fi and 3G networks), and by trading network efficiency for added delay and redundancy, a way to achieve a multi-to-multi point interactive communication suite was found and proof that the delay that was supposed to add did not pan out, contrary to initial belief. It is believed that the defined objective of this dissertation work, of developing a proof of concept, was a

success, being able to understand and prove that this solution is feasible.

In what regards the topology aspect of the environment, although it would have been a great step on the road to achieve a close to reality simulation environment to have real world topologies provided to the project, it is the author's belief that the use of small-world topologies is the second best choice, due to their randomness and scalability. These topologies came also highly recommend throughout the scientific paper read and proved to be a wise choice.

By analysing the previous chapter, the author has no doubt that network coding is a trivial successor of nowadays' multicast scenarios. By proving that distribution times are shorter, in every tested situation, it is safe to say that network coding is a clear choice for future development of networks and their application. The question about the overhead caused by the redundancy of messages can be considered, but it is the author's belief that since devices are prone to evolve at such a fast pace, and so do the Internet solutions available to the average user, the hindrance put on by this redundancy and the ensuing process of this redundancy can be discarded. Still this is merely speculative and lacks actual proof.

However, the author would like to express a concern, shared also by the PhD who aided him on his work, that has to do with the feasibility of the implementation of the suggested group conference-chat application for mobile devices running on the Android environment. Even though the overhead redundancy and process inflicted by the network coding technique may not cause a significant obstacle to present mobile devices, the processing of various video streams may constitute an overbearing burden to said devices, seeing that to have a group video call would mean to have as many video *codecs* running as the users in the conference. An alternative would be to develop a codec that would treat the screen as a matrix that would change according to the number of participants.

7.2 Future Work

This project, has many ways on how it could be continued. Throughout this report many suggestions have been made, but concerning the simulation testing, a new series of testing scenarios contemplating 4G networks could be developed, to see how they behave when compared to its predecessor, also concerning the simulation, an attempt to make a more realistic environment, more towards continuous simulation, in contrast to a current one, which is discrete, and with it make the simulation more dynamic, with peers entering and leaving the network as the simulation runs. This entails a change of the overlay topology every time an event as such occurs, and a system of peers announcing their arrival or departure. Also, a more profound study on the redundancy can be made, to understand if diminishing this quotient affects the distribution time in any way. So far there is no correlation between the two, but this study could shed some light to the topic, because it is in the interest of the network to lower the redundancy, as long as it does not affect the performance, when looking at the problem through a scalability point of view.

On a more practical note, the simulation built so far could be refactored to work on actual mobile devices, to understand how it behaves when it comes to process time. This could be

done before the development of the Android application, to be carried out by the University of Stockholm and the Swedish Ericsson, as a proof of concept to the network coding technique itself as a suitable method for low processing power devices.

References

[ACLY00] Rudolf Ahlswede, Ning Cai, Shuo-yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 46(4):1204–1216, 2000.

[Aga] Agape. https://traclifo.univ-orleans.fr/Agape/. Time of Last Access: 06/06/2012.

[CJHG08] Kaikai Chi, Xiaohong Jiang, Susumu Horiguchi, and Minyi Guo. Topology design of network-coding-based multicast networks. *IEEE Trans. Parallel Distrib. Syst.*, 19(5):627–640, May 2008.

[CWJ03] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding, 2003.

[FLBW06] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36:63–68, January 2006.

[FS07a] Christina Fragouli and Emina Soljanin. Network coding applications. *Foundations and Trends in Networking*, 2(2):135–269, 2007.

[FS07b] Christina Fragouli and Emina Soljanin. Network coding fundamentals. *Foundations and Trends in Networking*, 2(1), 2007.

[GMR06] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Anatomy of a p2p content distribution system with network coding. In *IPTPS*, 2006.

[GR05] Christos Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. 2005.

[Gra12] Afonso Graça. P2P Multicast Overlay For Smart Content Delivery. Master's thesis, Faculdade de Engenharia da Universidade do Porto and Budapesti Műszaki és Gazdaságtudományi Egyetem, Portugal, 2012.

[GRM09] Borislava Gajic, Janne Riihijärvi, and Petri Mähönen. Performance evaluation of network coding: Effects of topology and network traffic for linear and xor coding. *JCM*, 4(11):885–893, 2009.

[HLM+03] T. Ho, B. Leong, M. Medard, R. Koetter, Y. Chang, and M. Effros. The benefits of coding over routing in a randomized setting. jun 2003.

[JFr] Jfreechart. http://www.jfree.org/jfreechart/. Time of Last Access: 05/06/2012.

[JUN] Jung. http://jung.sourceforge.net/. Time of Last Access: 06/06/2012.

REFERENCES

[KMM03] Ralf Koetter, Muriel Médard, and Senior Member. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782–795, 2003.

[KRH+06] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. In *In Proc. ACM SIGCOMM*, pages 243–254, 2006.

[LBS10] Balázs Lajtha, Gergely Biczók, and Róbert Szabó. Enabling p2p gaming with network coding. pages 76–86, 2010.

[Lea11] Baochun Li and et al. Random network coding in peer-to-peer networks: From theory to practice, 2011.

[LMB07] Luísa Lima, Muriel Médard, and João Barros. Random linear network coding: A free cipher? *CoRR*, abs/0705.1789, 2007.

[LWLZ10] Zimu Liu, Chuan Wu, Baochun Li, and Shuqiao Zhao. Uusee: Large-scale operational on-demand streaming with random network coding, 2010.

[Mar04] Chip Martel. Analyzing kleinberg's (and other) small-world models. In *in Proc. of ACM Symp. on Princ. of Dist. Comp. (PODC*, pages 179–188. ACM Press, 2004.

[SLL08] Tara Small, Baochun Li, and Ben Liang. Topology affects the efficiency of network coding in peer-to-peer networks, 2008.

[VAHP05] Vasos Vassiliou, Josephine Antoniou, George Hadjipollas, and Andreas Pitsillides. A simulation tool to evaluate radio resource management algorithms for enhanced umts. In *Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WIOPT '05, pages 396–403, Washington, DC, USA, 2005. IEEE Computer Society.

[VLS10] Ingo Viering, Andreas Lobinger, and Szymon Stefanski. Efficient uplink modeling for dynamic system-level simulations of cellular and mobile networks. *EURASIP J. Wirel. Commun. Netw.*, 2010:73:1–73:15, April 2010.

[YLCZ05] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang. Network coding theory. *Foundation and Trends in Communications and Information Theory*, 2(4 and 5):241–381, 2005.

[yRLMYC03] Shuo yen Robert Li, Senior Member, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49:371–381, 2003.

[ZcLL06] Shengli Zhang, Soung chang Liew, and Patrick P. Lam. Physical-layer network coding. In *in ACM Mobicom '06*, 2006.