Chapter 3 – Simulating network coding in OPNET®

3.1 OPNET® network simulation software

3.1.1 Introduction

OPNET® is a high level event based network level simulation tool. This means it is a computer simulation tool that is easy to use, and for most applications thereof no programming skills are necessary. It uses a graphic interface to construct networks and works on an event basis, meaning it creates, handles and sends each packet individually. The user interface is constructed from C and C++ source code blocks together with a library of OPNET® functions. OPNET® can be used as either a network design simulator or a network management tool.

Most of the attributes of projects, nodes, models, processes etc. in OPNET® can easily be edited by means of an editor which contains all the basic parameters of the relevant object. An example thereof can be seen in the following figure:

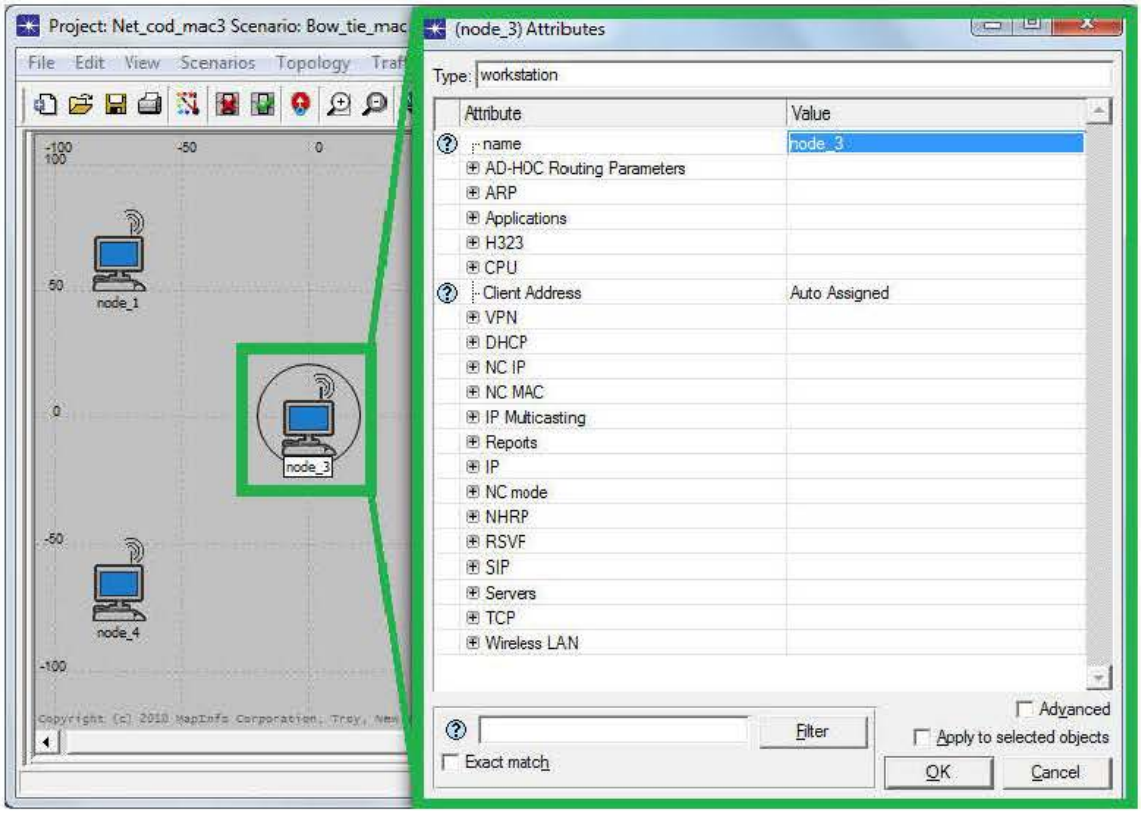

Figure 3-1: Node attributes

3.1.2 The OPNET® environment

OPNET® has an extensive model library, containing models for many protocols and standards. This enables an OPNET® user to build, simulate and analyze 802.11 networks. Brief overviews of a few OPNET® functions which are key to this study are now listed, and explanations on the use of each of them to construct a network follow:

3.1.2.1 Project model editor

The project window defines the network topology and link connections. One can set up and manage various levels of a network from this window. For example, the core or backbone of a network can stretch over multiple continents, while the access and service layers may fit into a room. The project window allows one to handle one's network in a layered approach, which is much more organized and definable than if everything was on one layer. This project uses a simple wireless office network which serves as the testbed.

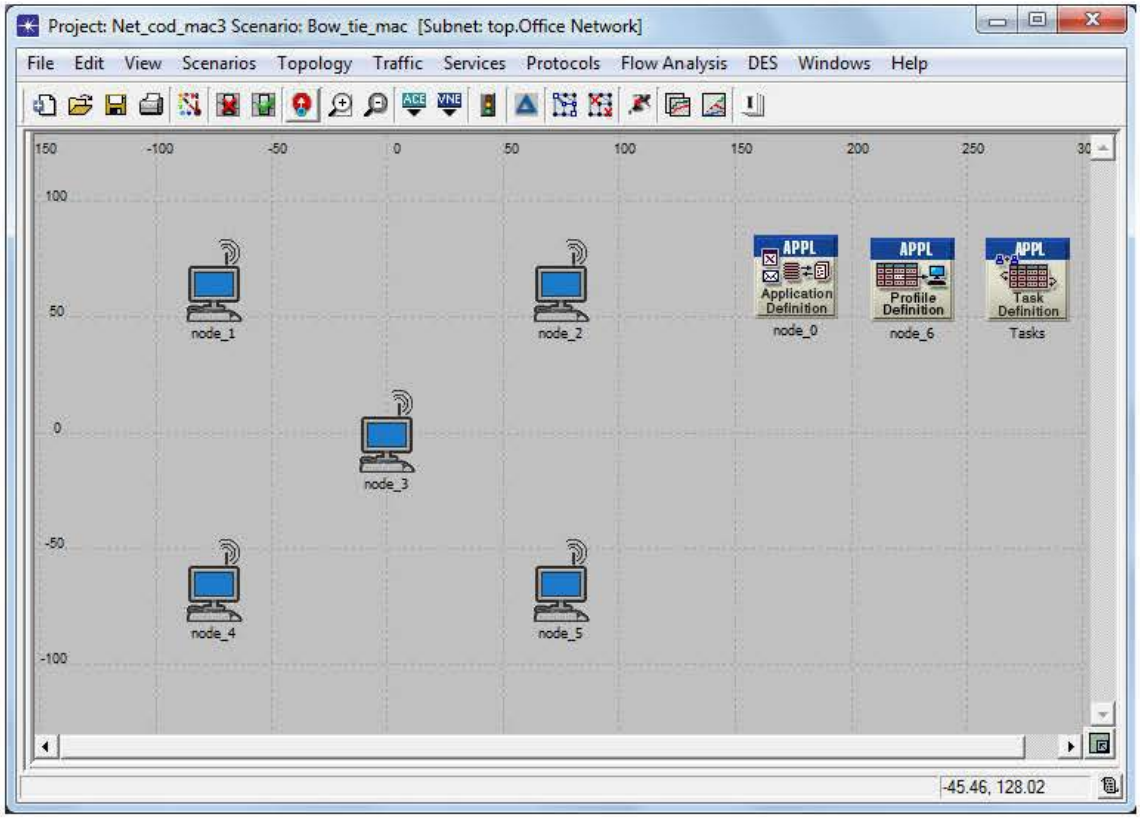

Figure 3-2: The project window

One can edit the attributes of the profile of each "user" behind a wireless workstation by configuring a profile in the profile definition tablet. This will establish how often the workstation uses which application. In the application definition tablet one can configure what kind of network traffic an application generates. The task definition tablet defines the basic unit of user activity within the context of the application.

3.1.2.2 Node model

The node model specifies the internal structure of a network component. For the network we used, wireless workstations were used for the structure of the network, as they depict a standard computer with a wireless interface. With these wireless workstations one can build an ad hoc network, which will form the basis for the experiments to be performed for this thesis. OPNET® has two node models for wireless workstations: a basic model with limited functionality, and an extended model, which has the full functionality of a normal wireless node. Since the extended model works similar to a real physical computer, using the extended node model for simulations would give results closer to that of a physical network. The extended model is better suited for our study since it incorporates more realistic parameters, and thus we may expect more realistic results from the simulations.

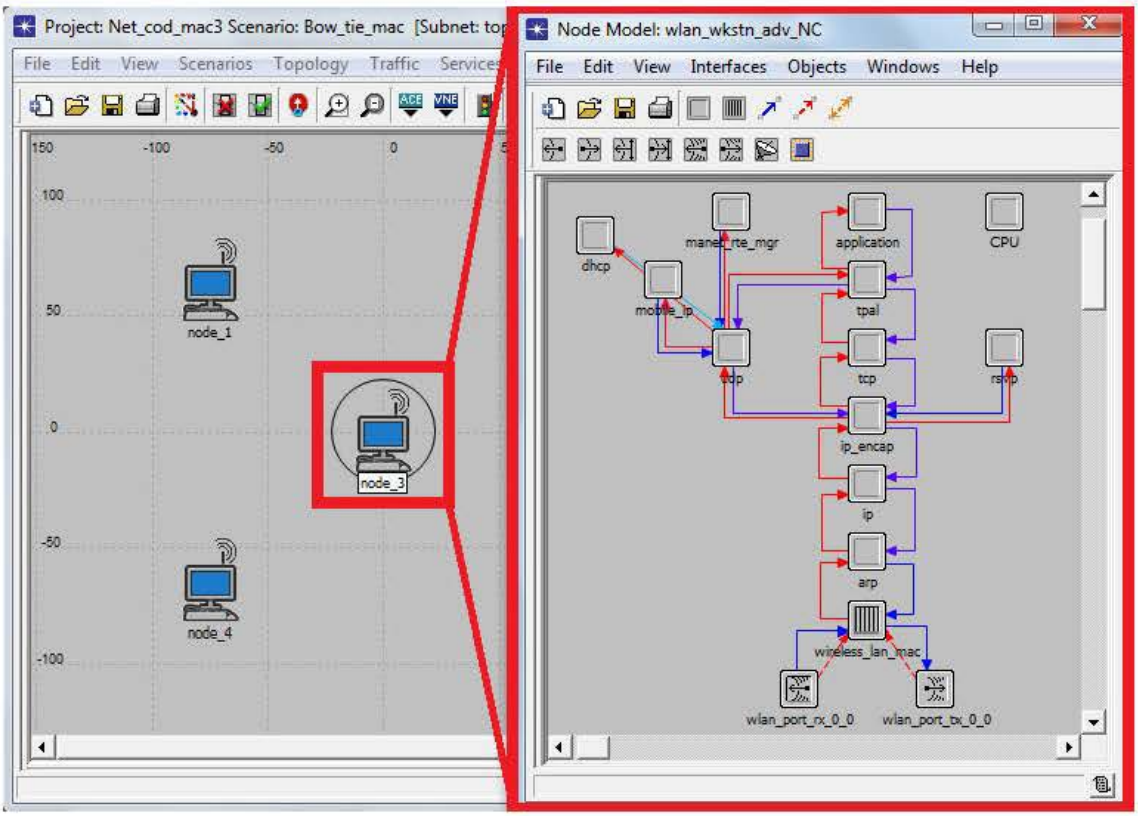

Figure 3-3: The extended node model

Notice in Figure 3.3 the internal working of the extended wireless workstation model is laid out in the layered structure of the OSI stack. This makes the logical flow of the workstations easier to understand, use and change, if necessary.

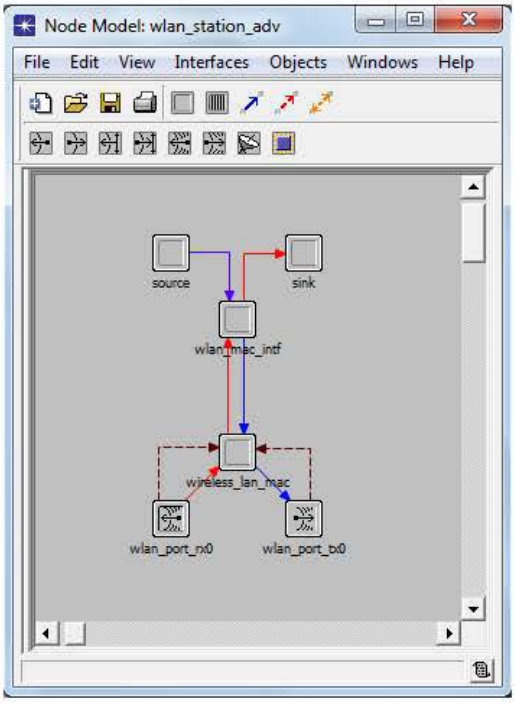

Figure 3-4: The basic node model

How the basic node model works, is easy to understand when compared to the extended model. It is ideally suited for development of lower level protocols, and a good learning platform if one wants to understand how OPNET® works internally.

Note also that OPNET® has no standard node model with network coding functionality. The OPNET® node model window makes it possible to change any OPNET® node model. This feature can be used if one wants to incorporate extra functionality into a standard OPNET® node model such as network coding.

3.1.2.3 Process model

The processes in a node model abstract the behaviour of the applicable network component. The node model consists of different discreet processes (the grey squares) which can communicate with each other by means of communication channels (The blue (output) and red (input) arrows). Each process is further described by its model. A typical process model is shown in the following figure. It is the TCP process model from the wireless workstation.

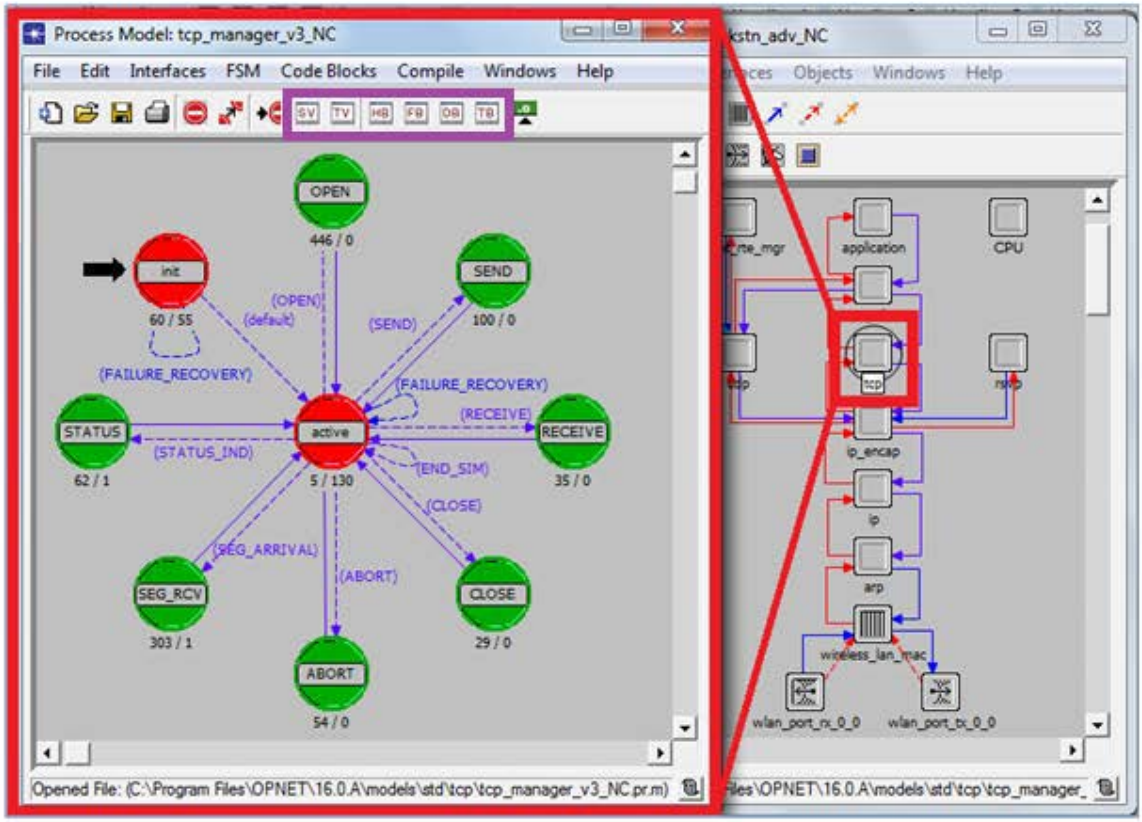

Figure 3-5: The process model

Each process consists of different states (depicted by the red (forced) and green (unforced) circles). There is an initialisation state (pointed out by the black arrow) in each process, and it is executed first when the applicable process is invoked. The states control the activity of the process, and the states pass control to and from each other by means of conditions (depicted by the blue arrows). In the process model editor one can access the process's variables and other blocks of code. The buttons are indicated by the purple square in the above figure. These give access to the state variables, the temporary variables, the header code block, the function code block, the diagnostic code block and the termination code block. The states themselves are just the graphical representation of pieces of source code, and in the source code one can then use the variables and functions which were previously defined. One can access each state's code by means of the source code editing environment. The following figure illustrates this.

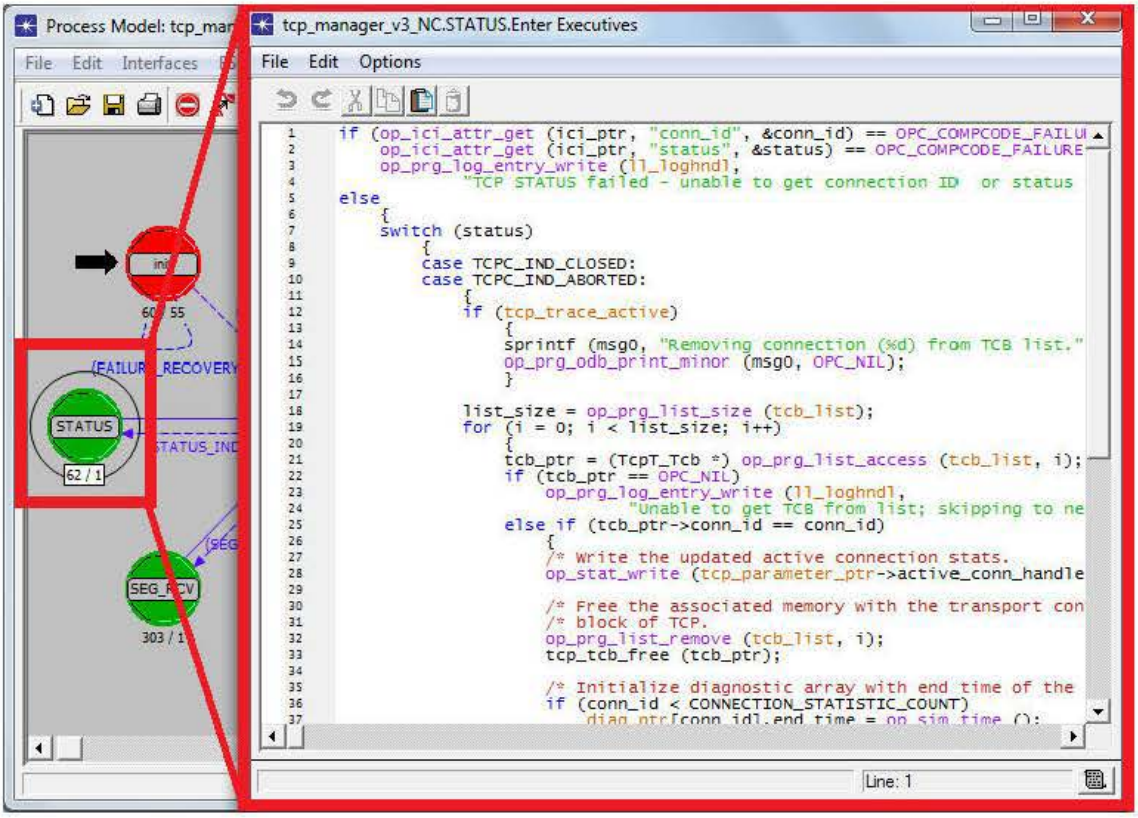

Figure 3-6: Source code editor

3.1.2.4 Simulation window

Each simulation can be set up and managed by the configure/run DES (Discreet Event Simulation) window that captures and displays simulation information and results. One can specify simulation variables such as the values per statistic collected and how often the GUI is updated by the statistics, as well as the seed values which determine the random number generation parameters. OPNET® also has a simulation sequence editor which can be used to iterate simulations with different set parameters. The simulation results can be viewed in graph form, and graphs can be compared. For further analysis, the data can be exported to Microsoft Excel®.

3.1.2.5 Packet format

Packet formats are defined in protocols, and can be seen and edited in the packet format editor. One can edit, add and delete fields of an existing protocol, or create your own packet format. The following shows the packet format for the 802.11 MAC:

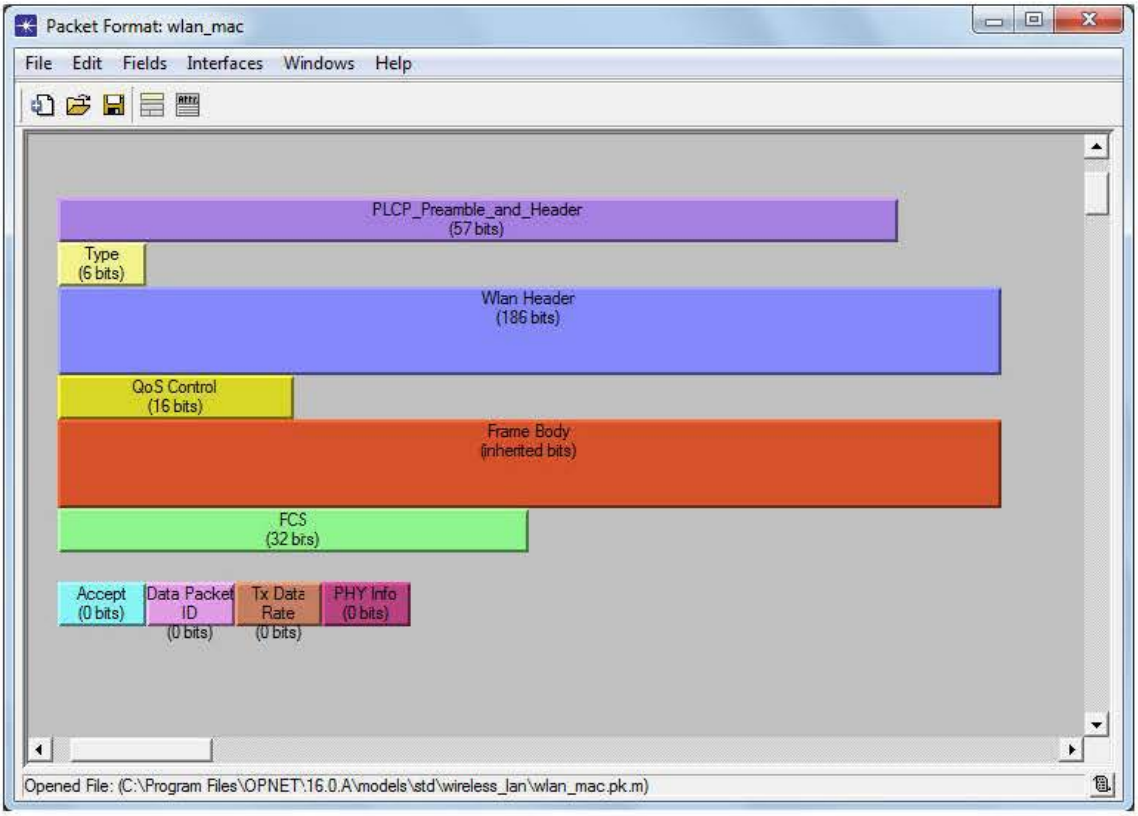

Figure 3-7: Packet format

3.1.2.6 Antenna pattern

In OPNET® one can also create a specific antenna for wireless devices. The antenna pattern editor can be used for this:

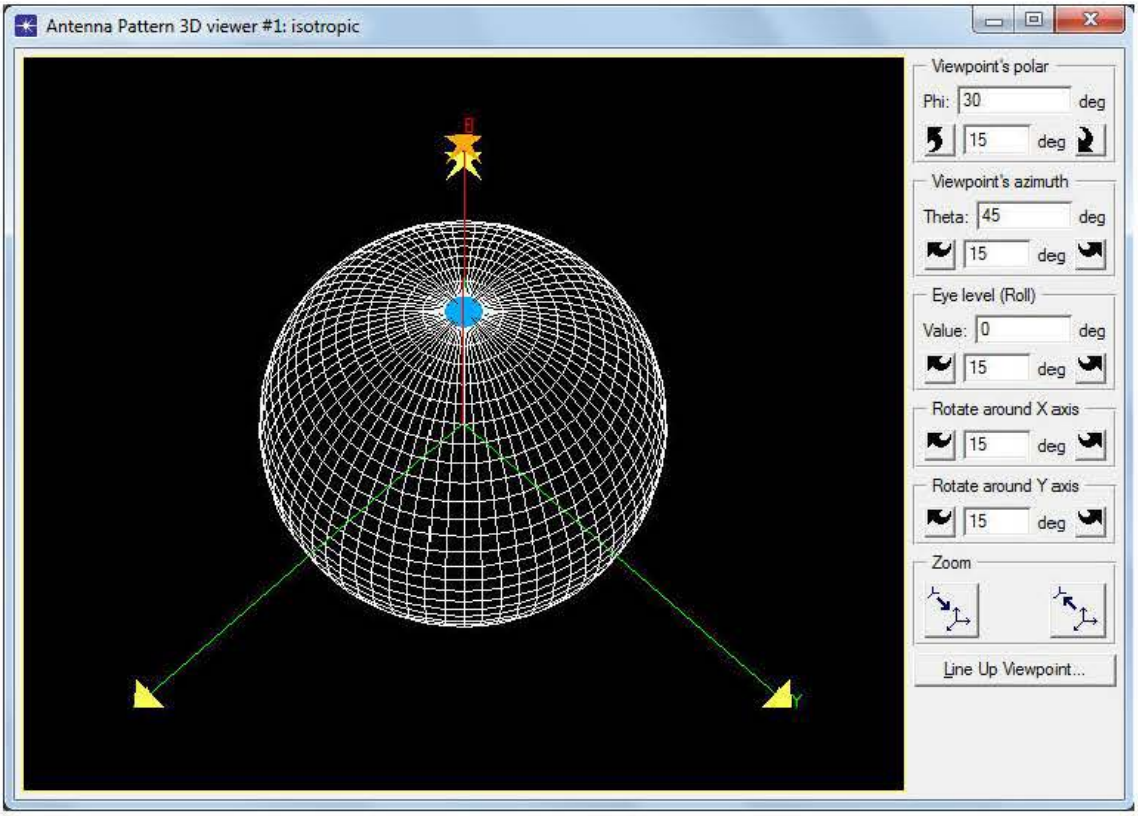

Figure 3-8: Antenna pattern editor

3.1.3 The OPNET® models

A very important factor to consider when using any simulator is the validation of the models it uses, so that one knows whether the results obtained are realistic, and if it can be compared with other results from physical systems. The following presents work which validates OPNET®'s models [58,59], with the following pertaining specifically to OPNET®'s wireless networks [60,61,62].

The fact that OPNET® uses validated models makes it an ideal simulation environment for conducting research, especially when dealing with a subject that requires validated results so that it can be compared to the results from other systems.

This introduction to OPNET® was provided so that the results from the work that we did in OPNET® can be better understood. The following section presents the methodology we followed for experimenting in OPNET®.

3.2 Methodology

3.2.1 Empirical investigation

A network using network coding was simulated in OPNET® to evaluate the implementation of network coding in the data link layer of the OSI stack. A benchmark network was first created, so that changes in the network could be compared to the benchmark, and from the difference in network performance the effects of the changes could be observed. The bench network, further on referred to as our standard test network, is discussed in section 3.2.3. The network architecture, 802.11 standard, data type and speed were varied to cause changes in the network. How these changes were implemented, and their effects are presented and discussed in chapter 4.

There is no standard OPNET® model capable of implementing network coding. If one wants to implement network coding in OPNET®, one first has to create a new OPNET® node model capable of network coding. To prove that this could be done in OPNET®, a proof of concept first had to be done to see if experimentation regarding network coding in OPNET® would be possible.

3.2.2 Proof of concept

A simple prototype simulation was needed to prove that network coding could be done in OPNET®. Firstly the data movement through the simulated network for the proof of concept is described in section 3.2.2.1. The newly created node model is depicted in section 3.2.2.2. Finally, the results and their implication from the proof of concept simulation are presented in section 3.2.2.3.

3.2.2.1 Data flow

The network used for the proof of concept is shown in Figure 3-9. The trafficking of frames in the network is explained thereafter.

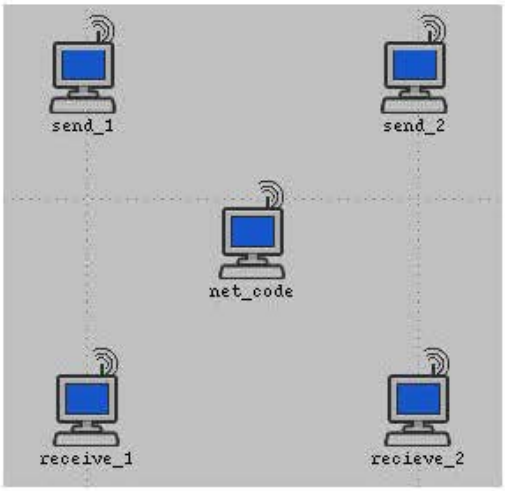

Figure 3-9: Proof of concept network

Figure 3-9 shows the network built in OPNET® for the proof of concept. In this network the nodes *send_1* and *send_2* are responsible for generating frames on the network. Each of these two nodes multicasts all the frames it generates to both nodes *receive_1* and *receive_2*. Node *send_1* has connectivity to *receive_1*, and can send data directly to it, but it does not have connectivity to *receive_2*, so it has to forward its frames to the node *net_code*, which can then send it to *receive_2*. Likewise, node *send_2* has connectivity with *receive_2*, but not with *receive_1*, so it has to forward its frames through *net_code* to *receive_1*. The nodes *send_1, send_2, receive_1* and *receive_2* all use the normal OPNET® basic node model for a wireless workstation. The node *net_code* makes use of our new node model so that it can perform network coding. Nodes *send_1* and *send_2* generate frames, and send them at a data rate of about 10kbps. This data rate was experimentally chosen because it creates enough traffic in the network to observe how the network is affected, without causing network performance to suffer from an overload of traffic. Since both sender nodes multicast all their frames to the receiver nodes, nodes *receive_1* and *receive_2* theoretically have to receive 20kbps. The node *net_code* performs network coding opportunistically. When its buffer is empty, it puts the first frame it receives into it, and waits for a second frame with which to network code the frame in the buffer. When it receives a second frame, the destination addresses of the two frames are compared, and if they are the same, the frame in the buffer is simply forwarded, the newly received frame is moved to the buffer, and the node waits once more to receive a new frame. When the frame's addresses differ, it network codes the frames and forwards the network coded frame to the receiver nodes. After this the node's buffer is empty and the process repeats itself.

Since nodes *send_1* and *send_2* generate the same number of frames at the same rate, *net_code* should receive frames from *send_1* and *send_2* interchangeably. This would result in all the frames being network coded.

3.2.2.2 Network coding node model

The basic node model for a wireless workstation in OPNET® was used as a platform for developing a model which could implement network coding in a network.

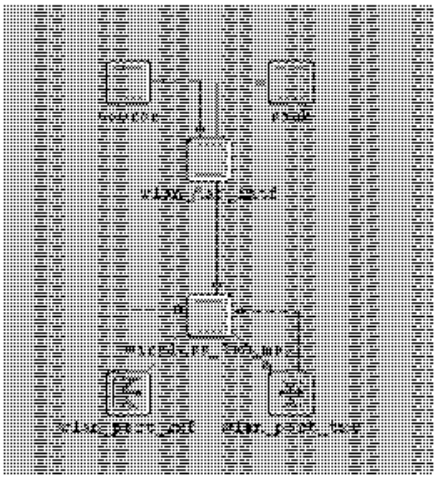

Figure 3-10: Standard OPNET® basic node model

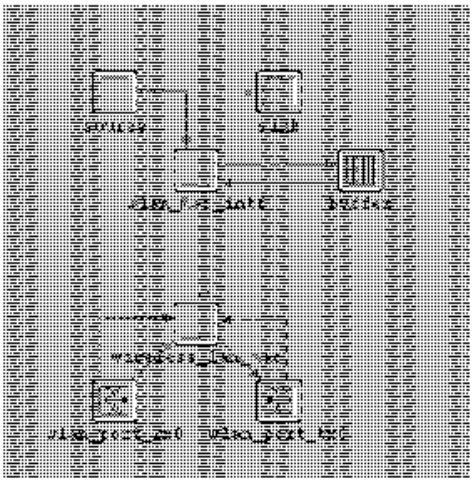

Figure 3-11: Modified network coding node model

Figures 3-10 and 3-11 show how OPNET®'s node model was changed so that it can perform network coding. An additional buffer was added so that a frame could be stored when the node was waiting for another frame with which to network code the first frame it received. The source code of the basic model was changed to accommodate new network coding functionality and so that the data flow previously described, could be realized.

3.2.2.3 Verification

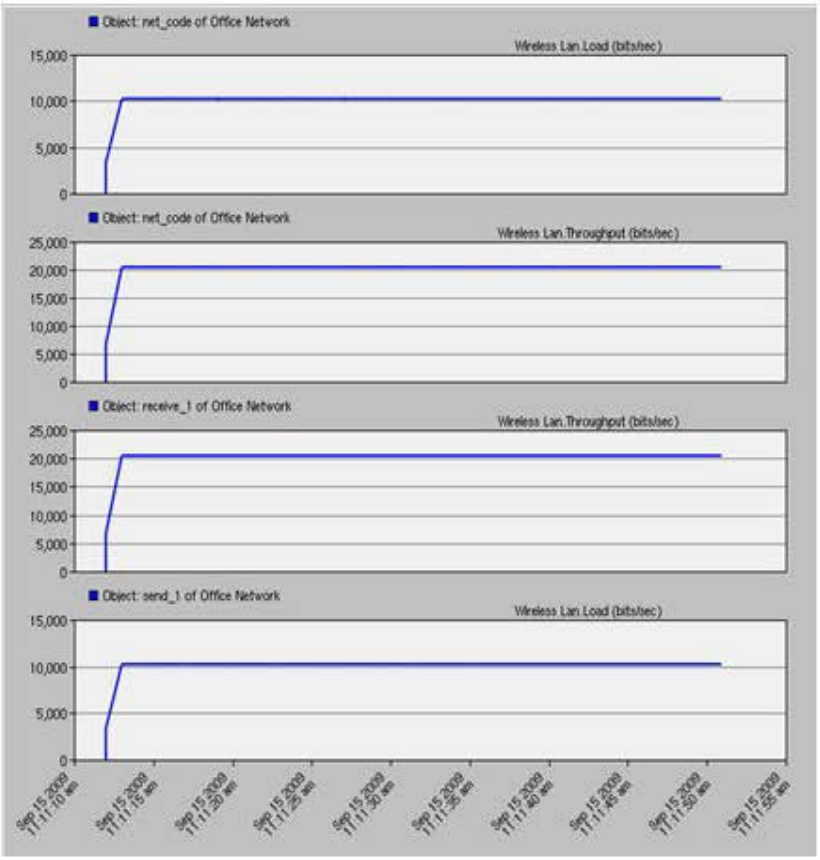

Figure 3-12: Proof of concept simulation results

Figure 3-12 shows the results for the proof of concept network simulation. From Figure 3-12, the top graph shows how many frames per second the *net_code* node sent, and the second graph how much it received. One can see that it sent half the number of frames that it received. This means that about 100% of the frames forwarded through *net_code* are network coded. This agrees to what should happen theoretically in the network. The last two graphs are for confirming that the receiving nodes received 20kbps and the sending nodes sent 10kbps. So finally, the total rate of frames sent in the network from the two sending nodes was 20kbps. The *net_code* node sent an additional 10kbps, and the receiving nodes received a total of 40kbps. The ratio of frames received versus frames sent, was 40k/30k, or 1.3. This clearly shows the potential throughput gain network coding could add to a network, as well as that network coding can be simulated in OPNET®.

3.2.3 Creating a standard test network

The proof of concept shown in the previous section proved that one can implement network coding in OPNET®. This section continues with the task of a more complete evaluation of the implementation of network coding in the data link layer of the OSI stack.

Most practical implementations and simulations of network coding use nodes which are alike to common PC's [4,10,23,36,37,45,49,50]. The software however varies from one implementation to another along with the protocols used.

When one wants to observe or compare the effects a certain component has on the performance of a network, one needs a standard to which one can refer the effect. This section deals with the details of creating a standard benchmark network in which certain components can be changed to see their effect on the network performance. We tried to choose all parameters and protocols so that our network would best simulate a typical real network.

3.2.3.1 Creating a standard node on which to experiment

3.2.3.1.1 Extended network coding node model

To implement network coding in OPNET®, an extended OPNET® wireless workstation node model was used as a base, and modified so that network coding could be implemented in the MAC layer, using the new node model. The greatest change was made to the model's MAC layer, which is represented by the grey blocks pointed out by the green arrows in figures 3-13 and 3-14. The MAC layer was chosen for the implementation of network coding because of the advantages related to the implementation of network coding in the MAC layer. These advantages are discussed in section 2.2.2.1.

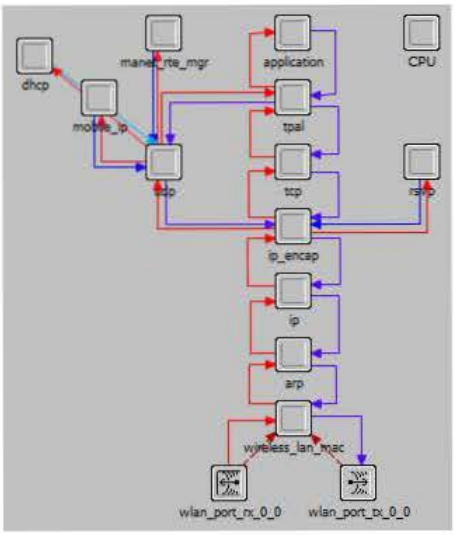

Figure 3-13: OPNET® node model

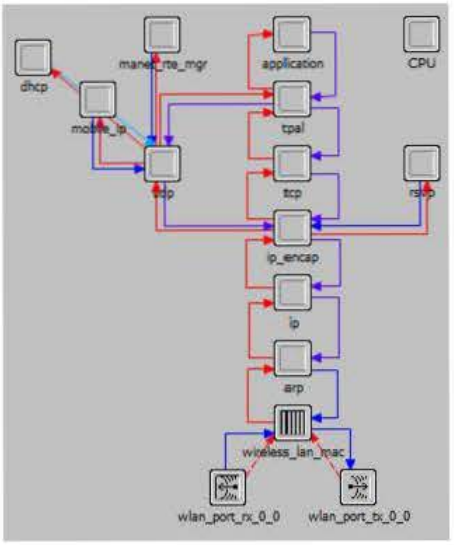

Figure 3-14: New node model

The new node model has added functionality and a buffer in the MAC process so that frames can be stored, and network coding implemented. This system forms a combination of deterministic and opportunistic network coding:

- The model is, in a sense, deterministic, as the system makes use of static routing, which means that the frames are routed in the network according to parameters which are specified before the simulation starts. The frames will always follow the same path to its destination. This was done as the simple network architecture which was used, had no need for a complicated routing algorithm. Using a network coding routing protocol would result in the same network performance, but the development thereof would have been exceptionally demanding. The routing is therefore considered deterministic, and the frames are sent with the route that will supply maximum network coding opportunities; all frames are forwarded through a node which is capable of network coding. This node will then look for network coding opportunities, and perform network coding accordingly.
- The model is also opportunistic because the node responsible for doing the network coding, codes frames when opportunities for network coding arise. The node does not request data in any specific way to accommodate network coding. It rather looks for coding opportunities with the frames that are being forwarded through it. The network coding itself is considered opportunistic because of this.

The new model's functional data flow can be seen in the following flowchart. Only the flow was changed from OPNET®'s model for the reception of a frame from the physical layer. The model stays the same when a frame is sent down the stack from the IP layer and has to be sent into the network through the physical layer.

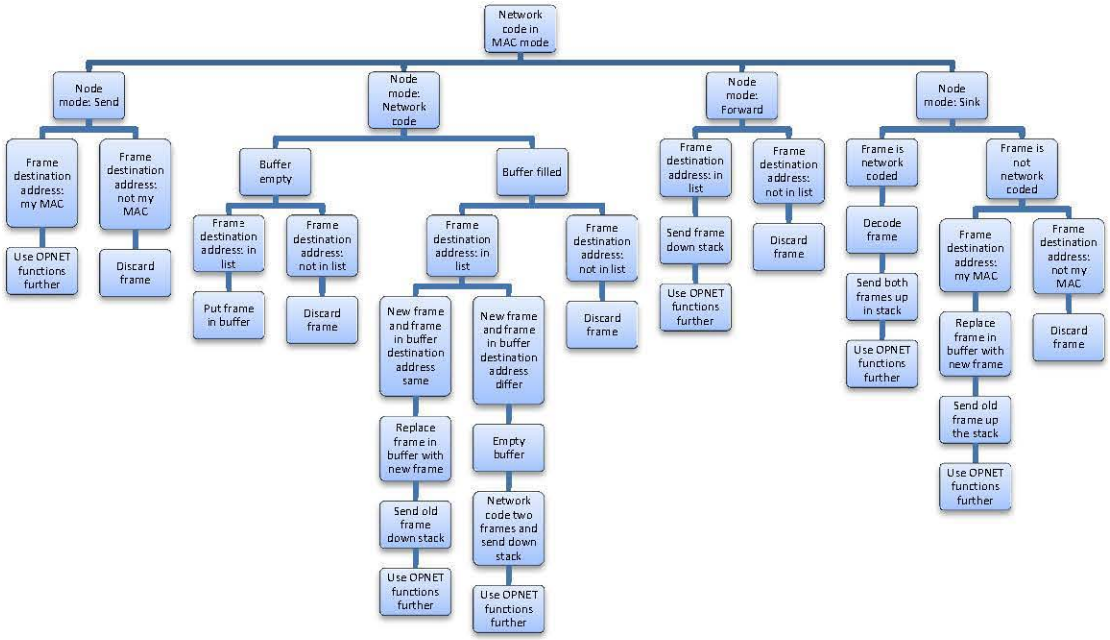

Figure 3-15: New node model's data flow chart for the reception of a frame from the physical layer

The key difference between the original OPNET® wireless workstation node model and our new node model lies in how a frame is handled when the MAC receives it from the physical layer. The functional flow of the new model in Figure 3-15 is now explained for when the MAC receives a frame from the physical layer. There are four different modes in which the new node model can be set to function. These functions are to be chosen to correspond with the function of the applicable node in the network:

1. Send mode. The destination address of the frame is compared to the node's own address, and if it agrees, it is sent further up the stack. When the addresses do not match, the frame is discarded.
2. Network code mode. If the node has no frame stored in its buffer, it stores the frame it has received in its buffer and waits for the next frame. When a second frame is received, the destination address is compared to that of the frame in the buffer. If they match, the frame in the buffer is forwarded, and the newly received frame is stored in the buffer. When the addresses differ, the two frames are network coded and forwarded with a special new address (101) that lets the recipient know that the frame is a network coded frame. The fact that the address 101 was used as a dedicated network coding address caused the restriction that no other node in the network is allowed to use 101 as its MAC address. When no MAC addresses are assigned in OPNET®, OPNET® assigns addresses chronologically (1, 2, 3, 4...) to the nodes in the network. There is therefore room for 100 nodes in a network using our new node model and without the simulation operator personally assigning MAC addresses, before one runs the risk of using address 101. Using address 101 for a node MAC address would result in the new node model performing incorrectly. The Wi-Fi MAC does not

support multicasting, and when a collision happens during a broadcast, there is no back-off by the nodes. These reasons added to the decision to use a unicast with a special network coding address to perform multicasts for Wi-Fi.

3. Forward mode. This mode sets the node to simply forward all frames it has received.
4. Sink mode. If the node has no frame in its buffer, it stores the first frame it receives with the frame's destination address matching the node's MAC address. When there is a frame in the node's buffer when a new frame is received, the destination address of a frame is checked, and when it is the same as the node's address, the newly received frame is put into the buffer, and the old frame from the buffer is sent up the stack to the IP layer. When it receives a frame with a network coded address, it uses the frame in the buffer to decode the network coded frame, and sends both frames from the decoding process up the stack to the IP layer. This process inherently guarantees synchronization between pairs of coded and uncoded frames. When a coded frame is received, and there is no uncoded frame in its buffer, there is a breach in the synchronisation, and the coded frame is discarded. The loss of one uncoded frame will result in the loss of one coded frame. Since the test network was set up in a noiseless environment, the added loss of coded frames which resulted from losing uncoded frames amounted to only about 0.5% of the total throughput. The effect was considered not to have any significant effect on the network performance, and this method of synchronisation was considered to be adequate for our simulations.

The inputs that are required in order for network coding to take place, using the new node model, can be specified in the attributes menu of a node (see Figure 3-16). The new node model has been designed so that no inputs into the source code of the node model are further more necessary. This makes the use of the new node model much more user friendly, and nodes capable of doing network coding can easily be placed in a network and be made to work without complicated configurations. The following Figure illustrates the network coding options for the new node model (the expanded options are used to configure the node's network coding characteristics):

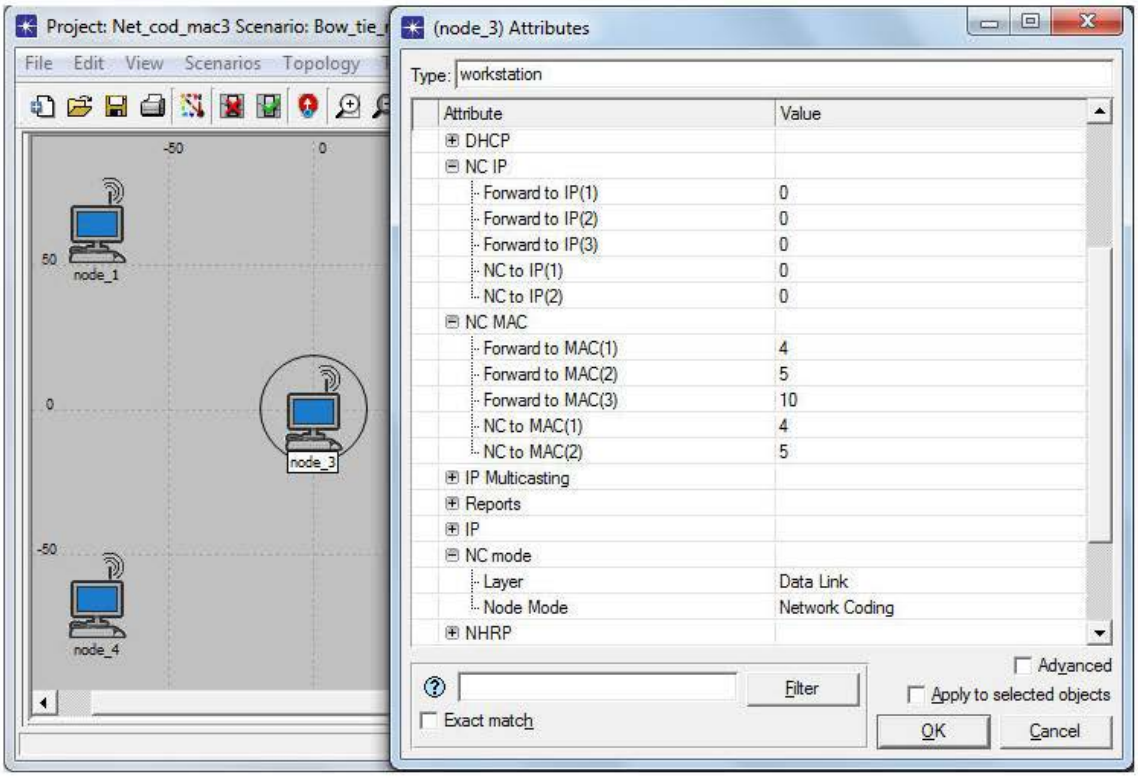

Figure 3-16: Network coding options

The NC Mode options specify in which layer of the node network coding is done, and also the role of the specific node in the network (Node Mode). The NC MAC options are inputs which the node needs so that it knows the addresses of the other types nodes in the network. For instance, if the node is configured to be a network coding node in the network, it needs to know the addresses of the two sink nodes so that it can know which frames to forward and which frames to network code.

The new node model is not designed for use in any network coding environment. It is designed for the specific use of a simple form of network coding, with only two senders and receivers. This simple form of network coding was adequate for the test network we used. The reason for the choice of network topology is explained in section 3.2.3.2.1.

3.2.3.1.2 Validation of new node model built in OPNET®

It has previously been discussed that OPNET® uses validated models in section 3.1.3. Their models can therefore be used to validate other models by means of a comparison. If a model behaves exactly the same as another validated model under the same conditions in all situations, and with the same parameters, then this model too has to be valid. We used this approach to validate our

new model by making a comparison between our model and the original OPNET® model on which it is based.

Two networks were built which were configured exactly the same in all aspects, except the node models used in the workstations. Each network consisted of two workstations, one set up as a server, and the other as a client. The server and client were set up to communicate with each other according to the specifications of the application used. Several simulations were run in both networks using different applications and data speeds. The seed values for creating random data in OPNET® were set to the same value for each simulation, meaning that random variables for the simulation should be generated exactly the same. The results from the two networks were exactly the same for each simulation, and the following figure shows one pair of results using a FTP application in the networks. The amount of data sent and received from each node in the network is shown.

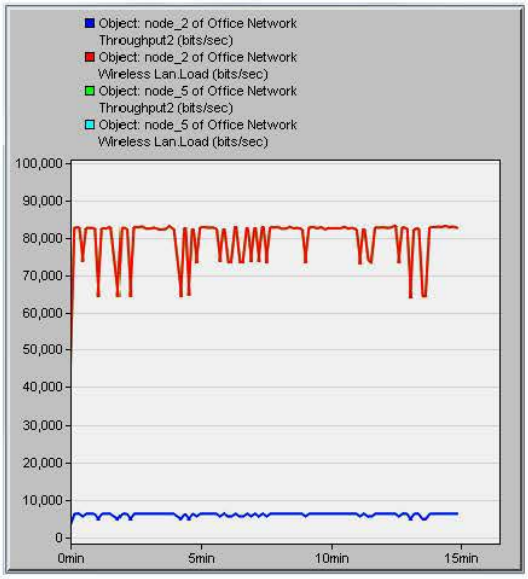

Figure 3-17: Results from network using new node model

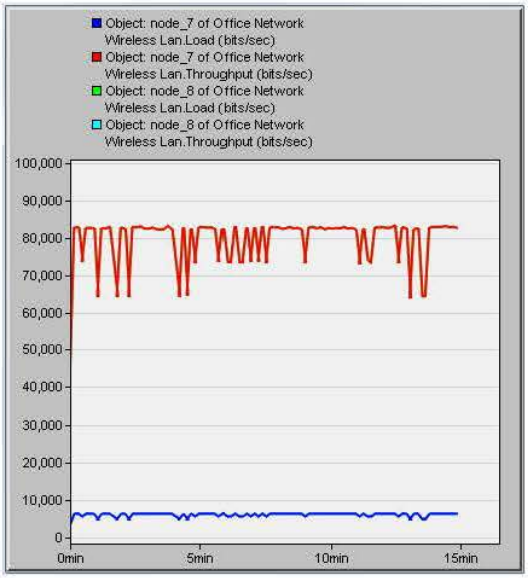

Figure 3-18: Results from network using OPNET®'s original node model

In Figures 3-17 and 3-18 it appears that only 2 of the indicated 4 graphs are shown each time, but this is not the case: the data sent and received by the nodes are exactly the same and lie on top of each other. So each line in the above figures is actually two lines.

Since our new model yields the same results as another validated model, our model must also be valid.

Our new model's data flow was monitored by making each node issue the details of frames received and sent through OPNET's command line. In this way we verified that our node model behaves as described in section 3.2.2.1. Using this tool we also confirmed that the model indeed implements network coding.

3.2.3.2 Creating a standard network on which to experiment

3.2.3.2.1 Network setup

Network topologies like the butterfly or bow-tie are often used to explain or test network coding. The box and butterfly networks are special cases of the normal bow-tie, and the capacity and data flow for these three networks are the same. When network coding is implemented in them, they all have two source nodes, two receiving nodes, and one network coding node. Because the box or high connectivity bow-tie network is simple to implement, and should perform the same as the other two networks, it was chosen to form the topology for our standard test network. Later these 3 network topologies are compared to each other to see if they really do perform the same.

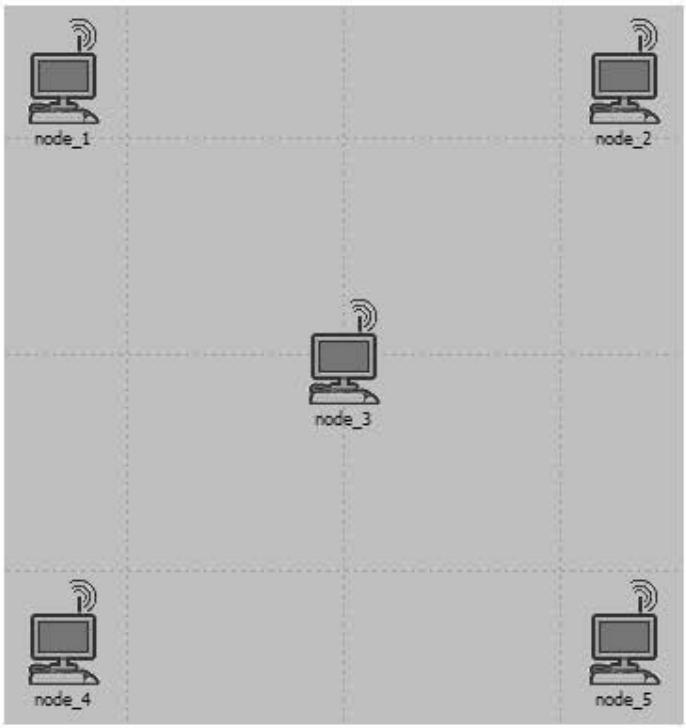

Figure 3-19: Network layout in OPNET©

The network setup of our standard test network is shown in Figure 3-19. Each node has an antenna with no attenuation, transmits with 0.005W and has a receiving threshold of -76dB. These nodes are placed in a theoretically perfect environment with no external interfering signals or obstacles, and with constant signal loss over distance. These factors cause the nodes to have connectivity at a constant distance of 139m. These nodes were found to always have connectivity for distances shorter than 139m, and never have connectivity for longer than 139m. The network's nodes are arranged in a symmetrical bow-tie network, with a distance of 130m from nodes 1 to 2, 91.5m from nodes 1 to 3 and 184m from nodes 1 to 5. Since this is a symmetrical network, the unmentioned distances correspond with the above mentioned distances. This layout causes all nodes to have direct connectivity except nodes 1 and 5, and nodes 2 and 4. This layout is referred to as the high connectivity bow-tie network forthwith, and is used to compare sending FTP and VOIP data. Section 3.2.3.3.1 explains why those two traffic types were chosen. The following figure shows a graphical representation of a high connectivity bow-tie network (also known as a five node box network), with the arrows indicating connectivity:

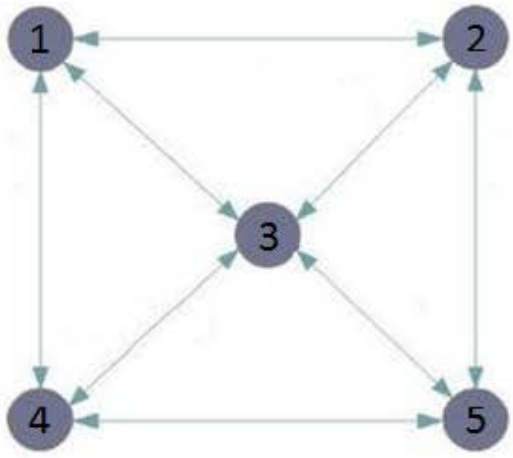

Figure 3-20: High connectivity bow-tie network

The layout of the nodes in Figure 3-20 shows the topology of our standard test network, and wherever reference is made to the nodes by their numbers forthwith, it will relate to the numbers and positions of the nodes in this figure. Also nodes 1 and 2 are forthwith called the sending nodes, node 3 the network coding or relay node, and nodes 4 and 5 the receiver or sink nodes.

All the nodes in the network were set up to use 802.11g with a data rate of 54Mbps. The range over which the nodes in the network can establish connectivity is not influenced by the data rate which it is set to.

3.2.3.2.2 Data flow

In this network the goal is to create opportunities for network coding, so data must be sent via a relay node (node 3) to two or more receiving nodes (nodes 4 and 5). To realize this, multicasts are sent from node 1 to nodes 4 and 5, and from node 2 to nodes 4 and 5. Since an OPNET® full node model (which is explained later in this section) is used, applications have to generate the data that is sent down the node model, and ultimately travels over the network to its destination node and application. A suitable application has to be chosen, and OPNET® makes provision for applications generating different data types on a server – client basis. Two applications were chosen: one requesting FTP (File Transfer Protocol) data, and the other VOIP (Voice Over Internet Protocol) data. Nodes 1 and 2 were chosen to be servers supporting the applicable application in the experiment, while nodes 4 and 5 ran the application that requested the data from the server. The data requested by the server was multicast to the nodes running the application. Node 3 served as a forwarding or network coding node, because nodes 1 and 2 individually send data to both nodes 4 and 5, and nodes 1 and 5 and 2 and 4 did not have connectivity, the data had to be forwarded through node 3.

3.2.3.2.3 Interference due to the hidden node problem

Because only node 3 has connectivity with all the other nodes in the network, the hidden node problem could occur because nodes cannot always see other nodes occupying the channel it wants to use. This causes interference and loss of data. Example: If node 1 is busy transmitting data across the network, node 5 will not realize that node 1 is occupying the channel on all the other nodes because it does not have connectivity with node 1. Node 5 could request a transfer session with node 2 because it does not realize node 2's channel is being occupied by node 1. This request will cause interference with the data being sent from node 1. If requests for data could be minimized, interference in the network due to the hidden node problem will also be minimized. The ratio of requests for data, and replied data varies from one application to another. Some applications could therefore cause more interference than others.

3.2.3.3 Creating a standard application on which to experiment

3.2.3.3.1 Applications in OPNET®

The purpose of the experiment is to see the effect of network coding on the data traffic in the constructed network. Therefore, it would be desirable to find a single application which shows the best visibility of the effects of network coding.

For data to flow in the network, it has to be generated by applications when one is working with the full node model from OPNET®. It works on a client-server basis, meaning one can set up some nodes

to be servers supporting certain applications, and then other nodes can be set up to use specified applications, thus requesting sessions with the applicable servers which support the applications they run. Once the sessions are established, the data will be sent to and fro from the client to the server over the network. In the simulations for the evaluation of the applications used, we used a network with client applications on nodes 4 and 5, with servers supporting these applications on nodes 1 and 2. All data sent from the servers, are always multicast to both client nodes.

Ideally one would want to send two constant streams of data from node 1 to 4 and 5, and from node 2 to 4 and 5. This would create an ideal environment for node 3 to apply network coding to both streams, because the streams would intersect at node 3. Two applications which could potentially simulate this perfect scenario for network coding were chosen, and their results were compared. The data rates for these applications were chosen by experimentally establishing that they put as much load on the network possible without the network being overloaded.

OPNET® applications can generate most stochastic and stationary data types. All data generation parameters, as well as the data structure itself can be determined by the applications to meet the user's need. How these features can be used to generate user-specific data is illustrated in section 4.1.3.

FTP:

The purpose of FTP is so send large files or data over a network. It works by a receiver sending a request for data to a server, and then the server replies with the data requested. The ratio of the amount of data sent and received by the receiver is dependent on the size of the data requested, but usually the request for data is less than 5% of the data itself for our applications. FTP uses TCP normally because a file transfer requires data integrity, otherwise the file one downloads, could end up being corrupt.

The network is set up so that node 4 attempts to send an average of about 6.1k bits per second as a request to the node 1, resulting in 79k bits per second sent back down from node 1 to both the nodes 4 and 5. Likewise node 5 attempts to also send 6.1k bits per second to node 2. The data that node 2 replies with is also sent at a rate of 79k bits per second to both nodes 4 and 5. The data from nodes 1 and 2 are thus multicast to both nodes 4 and 5.

VOIP:

The VOIP protocol is used for voice calls over a data network. With VOIP two clients establish a call session and simultaneously send a continuous stream of data to each other of the same size. Since OPNET® works on a client server basis a session will then be established between the server and

client and data will be exchanged at a one to one ratio. VOIP mostly uses UDP in normal data networks.

The network is set up so that node 4 attempts to send an average of about 16k bits per second as a request to the node 1, resulting in the same amount of data sent back down from node 1 to both the nodes 4 and 5. Likewise node 5 attempts to also send 16k bits per second to node 2. The data that node 2 replies with is also sent at a rate of 16k bits per second to both nodes 4 and 5. The data from nodes 1 and 2 are thus multicast to both nodes 4 and 5.

3.2.3.3.2 Choosing a suitable data type:

A suitable application and data type must be chosen for further simulations so that the effect of network coding can clearly be seen. This application will then be used to see the effect of other changes to the network on network coding performance. Our standard test network was used for a comparison between VOIP and FTP, to find out which of the two illustrates the effect of network coding the best.

3.2.3.3.2.1 VOIP:

When the VOIP applications were set up at first, they were set up to send data continuously, as most VOIP applications would normally do. However, this caused one server to occupy the wireless band, and not allowing the other any time to transmit at all. This is because it started transmitting a fraction of a second before the other, and once it started transmitting data, it never stopped to give the other server a chance to transmit data. The one server occupied the channel all the time, and the other could not start transmitting because the channel was occupied. Because of this, the applications were set to send data for ten seconds at a time with a break of about five seconds in between data transmissions to give the other server a chance to send. The nature of voice traffic is such that practically a transmission is seldom shorter than ten seconds, but this small value was chosen so that the servers could send data interchangeably as much as possible, because this created more network coding opportunities. The break was set up as five seconds, but further experiments showed that any break of less than ten seconds yielded the same results because it caused the servers to transmit interchangeably. When one server stopped transmission for its break, the other immediately started. Through the course of the simulations either one of the two servers occupied the channel. The time variables of ten and five seconds were set to a poisson distribution so that the average is ten and fifteen seconds are delivered with a poisson deviation for each repetition. The applications are not synchronised, and the applications would sometimes attempt to transmit simultaneously. This was done so that collisions would occur, as in a normal network.

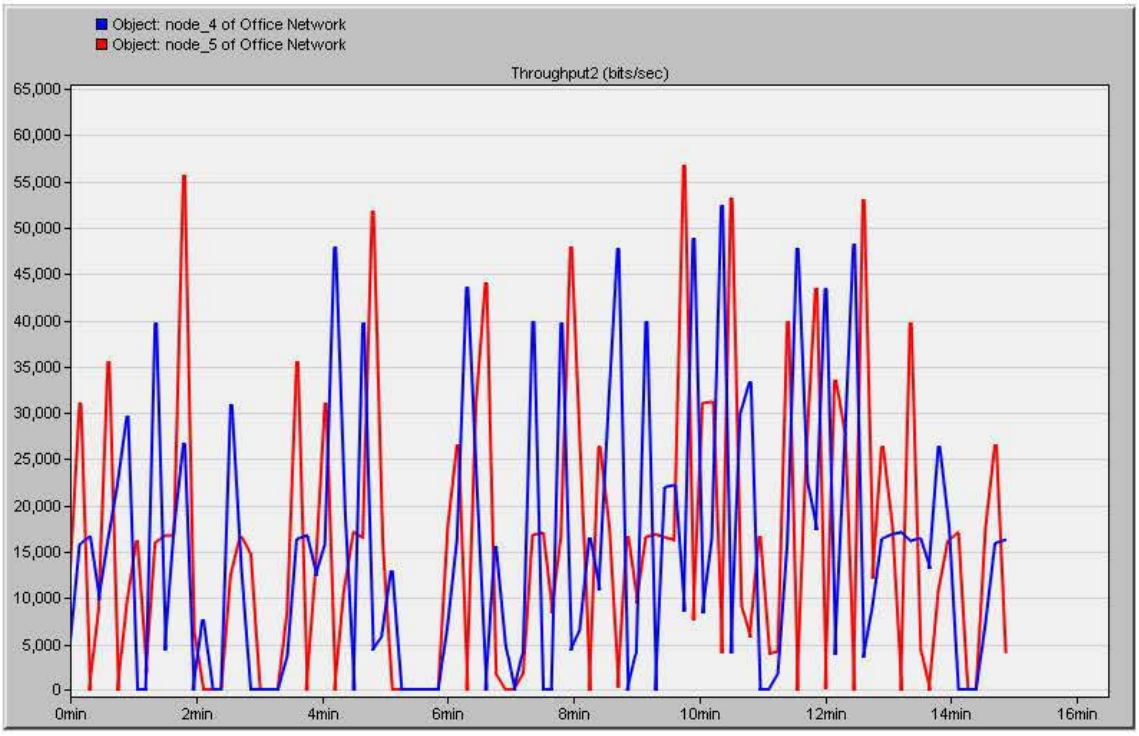

Figure 3-21: Throughput of two receiving nodes

Figure 3-21 shows the total number of frames received by each receiving node. Since all data sent in the network are multicast to both the receiving nodes, both receiving nodes should receive exactly the same frames. Because of interference due to the hidden node problem, the nodes do not receive the same frames. When many frames are lost a clear evaluation of the effect of network coding would be difficult.

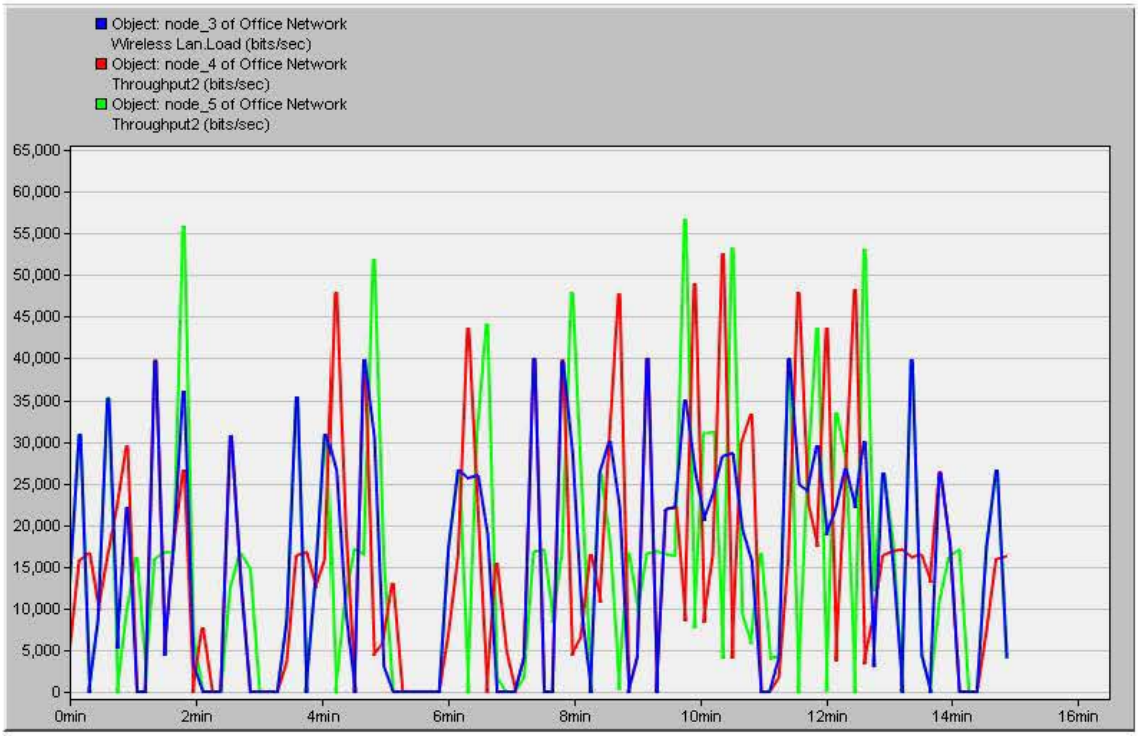

Figure 3-22: The rate by which nodes 4 and 5 received frames, together with the rate at which node 3 sent frames

All data sent from the servers (node 1 and 2) are multicast to both receiver nodes (nodes 4 and 5). The total number of frames that nodes 4 and 5 receive, should thus be exactly the same because all data that the servers send are multicast to both nodes. This also means that all frames sent from the servers should travel through node 3. If all frames sent to node 3 are network coded, the total number of frames that node 3 sends, should be equal to the total amount node 4 or 5 receives. If no network coding happens at node 3, the total number of frames that node 3 sends should be equal to the sum of what nodes 4 and 5 receive together, because all frames are forwarded through node 3. In other words, if all frames are network coded at node 3, the total number of frames node 3 sends, should be equal to half of what it should send when no network coding takes place. In the simulations only 3% of the total number of frames that node 3 sent, were network coded. This means that the amount of data that node 3 sent, should be very close to the sum of the number of frames nodes 4 and 5 receive. In Figure 3-22 one can clearly see that this is not the case – in fact the figure suggests that this happens infrequently. This is because too many frames are lost in the network due to interference.

Because of the symmetrical send and receive nature of VOIP, interference due to the hidden node problem often happens in the network because the VOIP applications send the same amount of traffic that they receive. The VOIP applications also create greatly varying amounts of traffic in the network. These two effects make the analysis of data from the simulations in the data link layer difficult, and the characteristics of network coding are difficult to observe, using VOIP applications.

3.2.3.3.2.2 FTP:

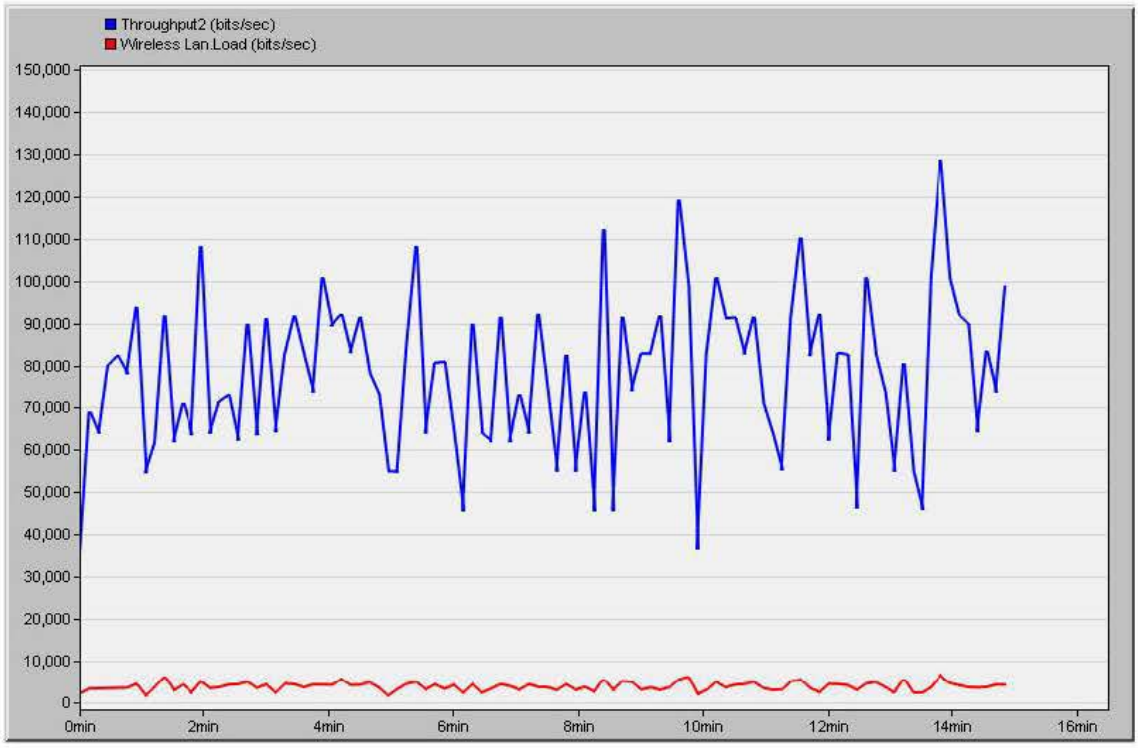

Figure 3-23: The amount of data sent and received by node 4

When the applications on the nodes are set up to use FTP, the requested data sent to the servers is small in comparison to the amount of data sent back by the servers, and thus the hidden node problem is minimized. The servers wait for an open channel to send their data, and thus compete evenly for the channel. This causes them to send data interchangeably. Network coding can happen when data from one server is still in the buffer, and new data from the other server is received. So these interchanging transmissions create an opportunity for network coding to take place. If fewer frames are sent before the other server starts to transmit, more network coding will take place.

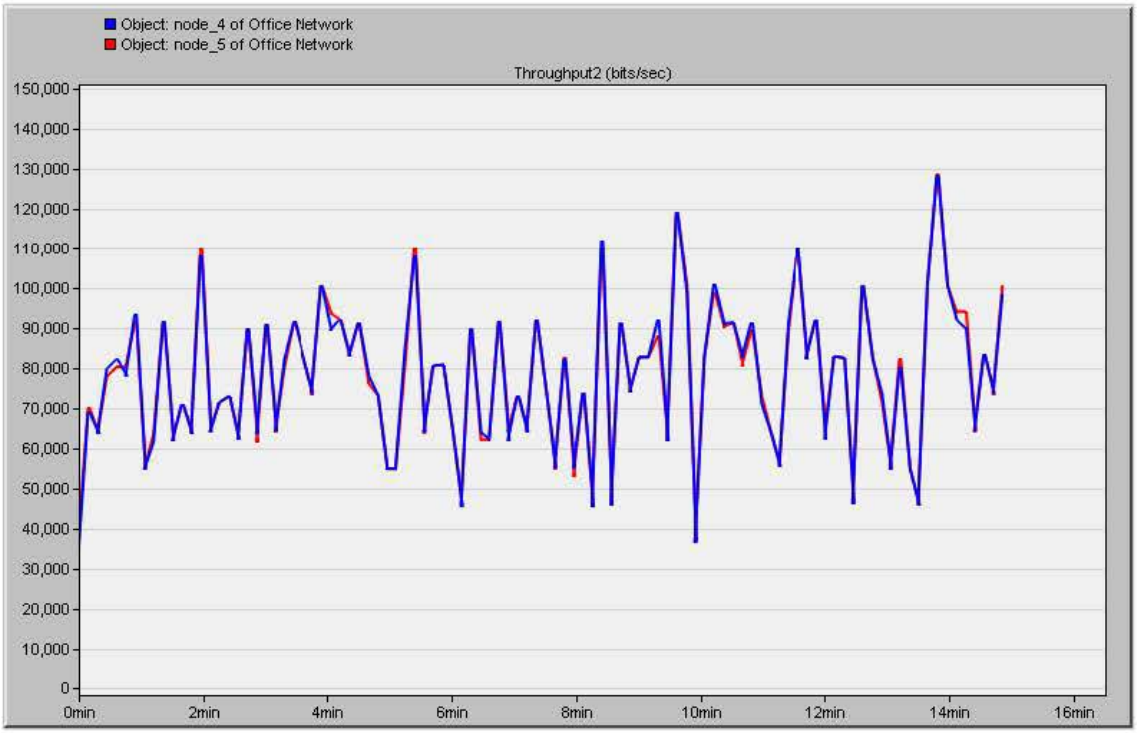

Figure 3-24: Throughput of two receiving nodes

In Figure 3-24 it can be seen that interference plays a much smaller role while FTP is in use than with VOIP (Figure 3-21). The two receiving nodes should receive the same data because all data sent from the servers are multicast to both receiving nodes, and the two graphs in Figure 3-24 are the two receiving nodes' received data, and these two graphs are almost identical. This shows how few frames are lost due to the interference caused by nodes 4 and 5 requesting new data (the hidden node problem).

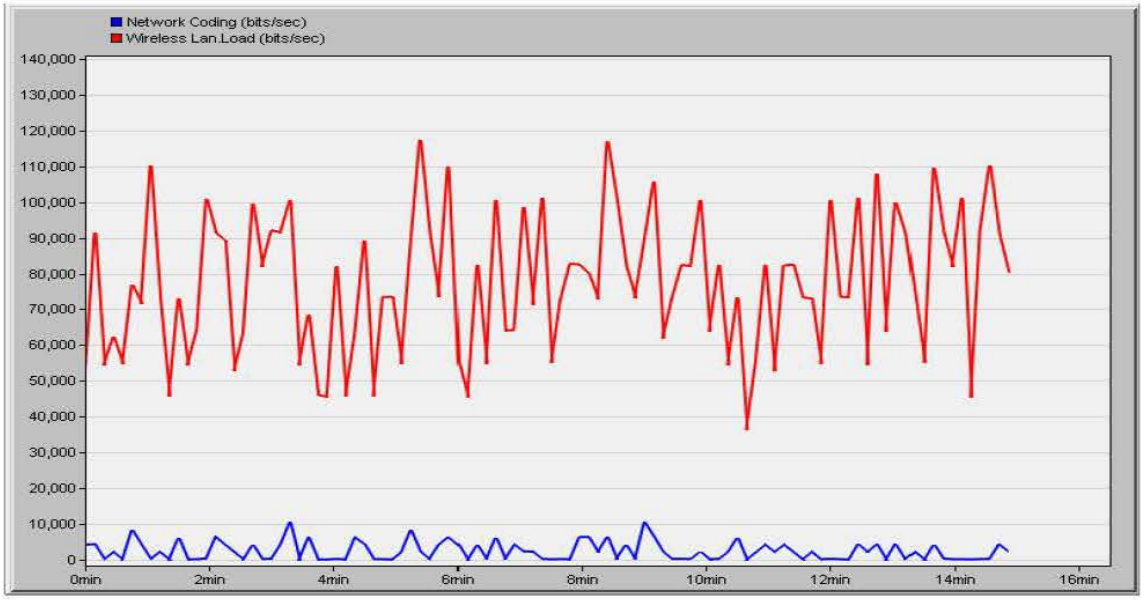

Figure 3-25: The total number of frames sent by node 3 (load), and the number of those frames which are network coded

2.95% of the total number of frames sent by node 3 was network coded when FTP was used, which is almost the same as VOIP (0.05% less). VOIP however uses UDP, which is later shown to be more favourable for network coding than TCP, which FTP uses. In section 4.1.5 we conduct a similar exercise with FTP over UDP and it shows significant improvements when compared to VOIP over UDP. The ratio of frames which were network coded to the total amount sent can clearly be observed when one uses FTP, as can be seen in Figure 3-25.

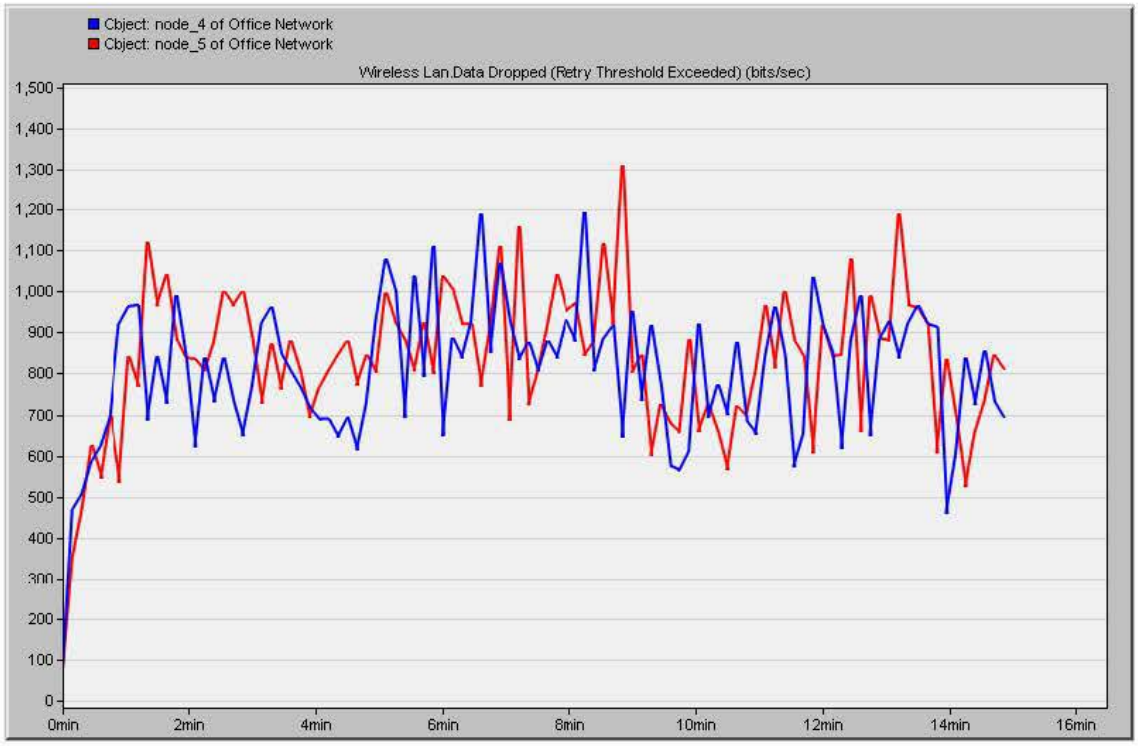

Figure 3-26: Data dropped by the receiver nodes

Some frames are dropped by the receiving nodes. This will cause the receiving nodes to receive less than they should. The average data dropped, is 803bps for node 4 and 830bps for node 5 (this is about 1% of the data it should receive).

Much less data is dropped because of interference with FTP in comparison to VOIP. Also both servers can be set to send data simultaneously with FTP because there are breaks between fragments of data being transmitted, and both servers get an equal chance to transmit. With VOIP however the servers must be set up to send interchangeably. Also the traffic that FTP causes in the network is more evenly spread than that of VOIP. These characteristics make FTP the better choice for analyzing the effect of network coding, and all the following simulations for the analysis of network coding was done using the FTP application.

3.3 Conclusion

This chapter gives an introduction on the network simulator OPNET®, and explains some of its key features and functions. Our methodology for the experimentation is explained, and the model we used for creating a network coding node in OPNET® is discussed. The proof of concept process we followed to determine if network coding could be done in OPNET® is also explained and presented.

We illustrate how we created a benchmark network which can be used to find the characteristics of network coding in the MAC. The node model, network topology and applications that were used in the benchmark network are also presented and discussed in this chapter. Our benchmark network is discussed with regard to the following: the node model used for the nodes, the network aspect, and the application used for generating network traffic.