

**SEVENTH FRAMEWORK PROGRAMME**  
**THEME 3**  
**Information & Communication Technologies (ICT)**



**ICT-213311**  
**Omega**



**Deliverable 5.4**

**Inter-MAC Protocols Performance Report**

|                                      |   |
|--------------------------------------|---|
| <b>Contractual Date of Delivery:</b> | <i>March 31<sup>st</sup>, 2010</i>                          |
| <b>Actual Date of Delivery:</b>      | <i>June 16th, 2010</i>                                      |
| <b>Editor(s):</b>                    | <i>Cristina Peña Alcega, Torsten Meyer, Vincenzo Suraci</i> |
| <b>Author(s):</b>                    | <i>See list of authors</i>                                  |
| <b>Work package:</b>                 | <i>WP5</i>  |
| <b>Estimated person months:</b>      | <i>132</i>  |
| <b>Security:</b>                     | <i>PU</i>   |
| <b>Nature:</b>                       | <i>Report</i>   |
| <b>Version:</b>                      | <i>1.4</i>  |
| <b>Total number of pages:</b>        | <i>44</i>   |

**Abstract**

*This document describes the configuration, the operation and the results of the simulations and measurements on real demonstrators that have been carried out in order to evaluate the performance of Inter-MAC protocols in data plane and control plane.*

**Keyword list**

*Inter-MAC, simulation, demonstration, performance, protocol, data plane, control plane, path selection, HWMP*

## Executive Summary

### *WP5 objectives in Omega*

The Inter-MAC work package aims at the definition of a technology independent convergence layer in charge of interfacing the network protocol with a number of heterogeneous physical transmission technologies in a seamless way. This Inter-MAC layer controls the different MAC layers by means of flexible and extensible technology-dependent Inter-MAC adaptors and provides functionalities like path selection, guaranteed quality of service, security, energy efficiency and is backward compatible and has better performance than state-of-the-art approaches.

### *Deliverable D5.4*

D5.4 “Inter-MAC Protocols Performance Report” describes the configuration, operation and results of the simulations and measurements that have been carried out in order to evaluate the performance of Inter-MAC protocols. The specification of such protocols and the interfaces among Inter-MAC engines was provided in D5.3 “Inter-MAC Protocol Entities Interfaces Specification” [OMD53] whereas the functionalities provided by them were described in D5.2 “Inter-MAC architecture design” [OMD52]. D5.4 is also related to D1.1 “Final Usages Scenarios Report” [OMD11] since two of these scenarios have been taken as a reference to evaluate handover (Scenario My media follows me) and performance when a link in usage goes down (Scenario Working from home).

### *Simulations and measurements*

Inter-MAC protocols performances have been tested in a simulated world as well as in a real world. The Opnet simulations cover the entire Omega network, from the single node equipped with the Inter-MAC layer to the whole Omega network scenarios. The demonstrators cover different parts of Inter-MAC architecture: the path selection algorithm to be run in the control plane, the data plane RAPTOR board, the timing analysis of the data plane, and the Linux kernel implementation of data plane.

In terms of activities, following tasks have been carried out in order to evaluate the performance of Inter-MAC protocols:

- Evaluation of control plane, through simulation with Opnet, and data plane performance with a particular focus on two possible path selection algorithms: reactive and hybrid (proactive + reactive);
- Evaluation, through direct measurement on demonstrator testbeds, of data plane running on different platforms with and without optimization:
  - o Data plane running on Power PC on FPGA.
  - o Linux PC kernel implementation of data plane.
  - o Linux PC userspace application implementing data plane.
  - o Data plane optimized for the System-on-Chip on a Virtex-4 FPGA of the RAPTOR-X64.
- Evaluation, through direct measurement on ad-hoc test-bed, of convergence time for control plane reactive path selection;

The simulations helped to identify the feasible concepts, while the demonstrators are evidences that what has been done in a simulated world can be realized in real prototypal setup. Moreover, the demonstrators showed that the prototypal performances are encouraging and further optimization of C-code will lead to even better performances.

### *Document structure and content*

Chapter 2 focuses on the detailed description of the simulation environment used to test the whole Inter-MAC and Omega network protocols performances and concepts.

Chapter 3 analyses, by means of different focused demonstrator, different protocol performances dealing both with data plane and control plane functionalities.

Chapter 4 highlights the main achievements, results and lessons learnt by the analysis of the simulation and demonstrator results in order to assess the real theoretical and practical performances of the Inter-MAC protocols and of the whole Omega network in different and challenging scenarios in an objective way.

### *Impact on the other Workpackages*

The present document impacts mainly on WP7, because all the considerations derived from the simulation and demonstration evidences provide clear indications on which protocol and architectural solutions are more suitable to achieve the desired results for the different Omega scenarios to be addressed in WP7.

## List of Authors

| First name      | Last name      | Beneficiary | Email address  |
|-----------------|----------------|-------------|--|
| Cristina        | Peña           | TID         | <a href="mailto:alcega@tid.es">alcega@tid.es</a>   |
| Michael         | Bahr           | Siemens AG  | <a href="mailto:bahr@siemens.com">bahr@siemens.com</a>   |
| Torsten         | Meyer          | Siemens AG  | <a href="mailto:torsten.meyer@siemens.com">torsten.meyer@siemens.com</a>                       |
| Francesco       | Delli Priscoli | UoR-CRAT    | <a href="mailto:dellipriscoli@dis.uniroma1.it">dellipriscoli@dis.uniroma1.it</a>               |
| Alberto         | Isidori        | UoR-CRAT    | <a href="mailto:isidori@dis.uniroma1.it">isidori@dis.uniroma1.it</a>                           |
| Roberto         | Cusani         | UoR-CRAT    | <a href="mailto:roberto.cusani@uniroma1.it">roberto.cusani@uniroma1.it</a>                     |
| Maria Gabriella | Di Benedetto   | UoR-CRAT    | <a href="mailto:gaby@acts.ing.uniroma1.it">gaby@acts.ing.uniroma1.it</a>                       |
| Vincenzo        | Suraci         | UoR-CRAT    | <a href="mailto:suraci@dis.uniroma1.it">suraci@dis.uniroma1.it</a>                             |
| Marco           | Castrucci      | UoR-CRAT    | <a href="mailto:castrucci@dis.uniroma1.it">castrucci@dis.uniroma1.it</a>                       |
| Antonio         | Pietrabissa    | UoR-CRAT    | <a href="mailto:pietrabissa@dis.uniroma1.it">pietrabissa@dis.uniroma1.it</a>                   |
| Manlio          | Proia          | UoR-CRAT    | <a href="mailto:manlio.proia@elsagdatamat.com">manlio.proia@elsagdatamat.com</a>               |
| Gabriele        | Tamea          | UoR-CRAT    | <a href="mailto:tamea@infocom.uniroma1.it">tamea@infocom.uniroma1.it</a>                       |
| Andi            | Palo           | UoR-CRAT    | <a href="mailto:andi@dis.uniroma1.it">andi@dis.uniroma1.it</a>                                 |
| Guido           | Oddi           | UoR-CRAT    | <a href="mailto:oddi@dis.uniroma1.it">oddi@dis.uniroma1.it</a>                                 |
| Philippe        | Christin       | FT          | <a href="mailto:philippe.christin@orange-ftgroup.com">philippe.christin@orange-ftgroup.com</a> |
| Marcin          | Brzozowski     | IHP         | <a href="mailto:brzozowski@ihp-microelectronics.com">brzozowski@ihp-microelectronics.com</a>   |
| Christian       | Liss           | UPB         | <a href="mailto:liss@hni.uni-paderborn.de">liss@hni.uni-paderborn.de</a>                       |
| Stefan          | Nowak          | UniDo       | <a href="mailto:stefan.nowak@uni-dortmund.de">stefan.nowak@uni-dortmund.de</a>                 |

## Document History

| First name | Last name | Version | Comments  |
|------------|-----------|---------|---|
| Cristina   | Peña      | 0.0     | Creation of document.   |
| Torsten    | Meyer     | 0.2     | Siemens: Included Chapter 5 "Convergence time measurements for reactive path selection".  |
| Cristina   | Peña      | 0.3     | TID: Included list of tables, list of figures, draft of Chapter 1 "Introduction", and reviewed Chapter 2 "Simulation Results" (by UniRoma). |
| Cristina   | Peña      | 0.4     | TID: Included combined description about data plane measurements set-up in demonstrator v1 coming from FT and IHP                           |
| Vincenzo   | Suraci    | 0.5     | Document structure changed; introduction, executive summary and conclusion added.   |
| Cristina   | Peña      | 0.6     | Improvement of Introduction.  |
| Christian  | Liss      | 0.7     | Included Data Plane results in Chapter 3.   |
| Torsten    | Meyer     | 0.8     | Siemens: Updated section 3.2 "Control Plane measurements".  |
| Cristina   | Peña      | 1.0     | TID: Updated chapter 3 with new contribution about Data plane measurements from IHP. Added conclusions.                                     |
| Vincenzo   | Suraci    | 1.1     | Executive summary, overall review.  |
| Stefan     | Nowak     | 1.2     | Overall review.   |
| Cristina   | Peña      | 1.3     | Final version to be sent to internal quality review process.  |
| All        |           | 1.4     | Final version to be submitted to EC.  |

## List of Acronyms

| Acronym | Meaning   |
|---------|---|
| AODV    | Ad-hoc On-demand Distance Vector                |
| ARP     | Address Resolution Protocol                     |
| BER     | Bit Error Rate                                  |
| DES     | Discrete Event Simulation                       |
| DP      | Data Plane                                      |
| FE      | Fast Ethernet                                   |
| FPGA    | Field Programmable Gate Array                   |
| FSM     | Finite State Machine                            |
| FTP     | File Transfer Protocol                          |
| HWMP    | Hybrid Wireless Mesh Protocol                   |
| ICI     | Interface Control Information                   |
| ICMP    | Internet Control Message Protocol               |
| ICT     | Information and Communication Technologies      |
| ID      | IDentifier                                      |
| IEEE    | Institute of Electrical & Electronics Engineers |
| IMAC    | Inter-MAC                                       |
| IP      | Internet Protocol                               |
| ISO     | International Organization of Standardization   |
| MAC     | Medium Access Control                           |
| MIT     | Massachusetts Institute of Technology           |
| MTU     | Maximum Transmission                            |
| NAS     | Network Attached Storage                        |
| OMEGA   | hOME Gigabit Access                             |
| OSI     | Open System Interconnection                     |
| PC      | Personal Computer                               |
| PDA     | Portable Digital Assistant                      |
| PDF     | Probability Density Function                    |
| PERR    | Path Error Message (HWMP Message)               |
| PLC     | Power Line Communication                        |
| PREP    | Path Response (HWMP Message)                    |
| PREQ    | Path Request (HWMP Message)                     |
| PS      | Path Selection                                  |
| QoS     | Quality of Service                              |
| QP      | Queue or Processor                              |
| RANN    | Route ANNouncement (HWMP Message)               |

---

|              |  |
|--------------|--|
| <b>RF</b>    | <b>Reply and Forward (HWMP Flag)</b>       |
| <b>RX</b>    | <b>Reception</b>                           |
| <b>STD</b>   | <b>State Transition Diagram</b>            |
| <b>TCP</b>   | <b>Transfer Control Protocol</b>           |
| <b>TO</b>    | <b>Target Only (HWMP Flag)</b>             |
| <b>TSPEC</b> | <b>Traffic SPECification</b>               |
| <b>TX</b>    | <b>Transmission</b>                        |
| <b>UDP</b>   | <b>User Datagram Protocol</b>              |
| <b>VHDL</b>  | <b>VHSIC Hardware Description Language</b> |
| <b>WLAN</b>  | <b>Wireless Local Area Network</b>         |

## Table of contents

|            |   |           |
|------------|---|-----------|
| <b>1</b>   | <b>Simulation Results.....</b>                              | <b>9</b>  |
| <b>1.1</b> | <b>Description of Tools and Methods used.....</b>           | <b>9</b>  |
| 1.1.1      | Opnet Modeler .....   | 9         |
| 1.1.1.1    | Opnet Modeler general description.....                      | 9         |
| 1.1.1.2    | Modeler architecture .....                                  | 9         |
| 1.1.1.3    | Opnet Editors and tools .....                               | 10        |
| 1.1.1.4    | Node Domain .....   | 11        |
| 1.1.1.5    | Process Domain.....   | 12        |
| 1.1.2      | Inter-MAC implementation on Opnet.....                      | 15        |
| 1.1.2.1    | The Omega device .....                                      | 15        |
| 1.1.2.2    | The Inter-MAC root process.....                             | 18        |
| 1.1.2.3    | Path Selection Engine Process .....                         | 20        |
| 1.1.2.4    | Monitoring Engine Process .....                             | 23        |
| 1.1.2.5    | QoS Engine Process.....                                     | 24        |
| 1.1.2.6    | Forwarding Engine Process.....                              | 24        |
| <b>1.2</b> | <b>Simulation Setup.....</b>                                | <b>25</b> |
| 1.2.1      | Simulation configuration .....                              | 25        |
| 1.2.1.1    | My media follows me – Scenario.....                         | 27        |
| <b>1.3</b> | <b>Comparison between reactive and hybrid mode.....</b>     | <b>32</b> |
| 1.3.1      | Path setup and convergence time test.....                   | 32        |
| 1.3.1.1    | Reactive Mode Path Set-Up Time .....                        | 32        |
| 1.3.1.2    | Proactive Mode Protocol Convergence Time.....               | 33        |
| 1.3.2      | Protocol Overhead of HWMP .....                             | 33        |
| <b>2</b>   | <b>Demonstrator results.....</b>                            | <b>36</b> |
| <b>2.1</b> | <b>Data Plane .....</b>                                     | <b>36</b> |
| 2.1.1      | Performance of Data Plane on FPGA and PC platforms.....     | 36        |
| 2.1.1.1    | Description of demonstrator .....                           | 36        |
| 2.1.1.1    | Configuration of demonstrator and measurements .....        | 36        |
| 2.1.1.2    | Results of PowerPC on FPGA demonstration .....              | 37        |
| 2.1.1.3    | Results of Linux kernel demonstration .....                 | 37        |
| 2.1.1.4    | Results of Linux user space demonstration.....              | 37        |
| 2.1.2      | Performance of the Data Plane optimized on RAPTOR-X64 ..... | 37        |
| 2.1.2.1    | Description of demonstrator.....                            | 37        |
| 2.1.2.2    | Configuration of demonstrator and measurements .....        | 37        |
| 2.1.2.3    | Results .....   | 37        |
| <b>2.2</b> | <b>Control Plane .....</b>                                  | <b>39</b> |
| 2.2.1      | Protocol Procedures.....                                    | 39        |
| 2.2.2      | Measurement Setup .....                                     | 39        |
| 2.2.3      | Measurement Results .....                                   | 40        |
| 2.2.4      | Summary.....  | 40        |
| <b>3</b>   | <b>Conclusion .....</b>                                     | <b>42</b> |
| <b>3.1</b> | <b>Conclusions on control plane performance .....</b>       | <b>42</b> |
| <b>3.2</b> | <b>Conclusions on data plane performance .....</b>          | <b>42</b> |
| <b>4</b>   | <b>References.....</b>                                      | <b>44</b> |

## List of Tables

|   |    |
|---|----|
| Table 1: Variables.....                               | 14 |
| Table 2: Node Technologies in the Omega Network ..... | 28 |
| Table 3: Flows characteristics .....                  | 28 |

## List of Figures

|  |    |
|--|----|
| Figure 1: Modeling steps .....   | 9  |
| Figure 2: Opnet graphical user interfaces .....  | 10 |
| Figure 3: Node Model .....   | 12 |
| Figure 4: Possible Representation of TCP/IP Protocol Stack .....                             | 12 |
| Figure 5: Process Editor .....   | 13 |
| Figure 6: Forced and Unforced States .....   | 13 |
| Figure 7: WLAN/Ethernet Workstations .....   | 15 |
| Figure 8: Inter-MAC Interfacing .....  | 16 |
| Figure 9: Single technology Omega Devices: Ethernet, PLC and Wi-Fi.....                      | 17 |
| Figure 10: Multiple Technologies Omega Device .....  | 18 |
| Figure 11: Root Process Model.....   | 19 |
| Figure 12: Inter-Mac attributes .....  | 20 |
| Figure 13: The problem of high initial latency in the on-demand routing mode .....           | 21 |
| Figure 14: The problem of non optimum routing paths in the proactive routing.....            | 22 |
| Figure 15: Path Selection Engine process .....   | 22 |
| Figure 16: Monitoring Engine process.....  | 23 |
| Figure 17: QoS Engine Process.....   | 24 |
| Figure 18: Forwarding Engine process .....   | 25 |
| Figure 19: Placement of nodes and links .....  | 25 |
| Figure 20: Inter-MAC attributes .....  | 26 |
| Figure 21: Defining a new application .....  | 26 |
| Figure 22: Deploying Applications.....   | 27 |
| Figure 23: My media follows me.....  | 28 |
| Figure 24: Data traffic received/sent .....  | 29 |
| Figure 25: Smartphone end-to-end delay.....  | 30 |
| Figure 26: Data packet dropped.....  | 30 |
| Figure 27: Smartphone bit error rate .....   | 30 |
| Figure 28: PDA data traffic received/sent.....   | 30 |
| Figure 29: Packet End-To-End delay .....   | 31 |
| Figure 30: Big Screen traffic received .....   | 31 |
| Figure 31: Download response time.....   | 32 |
| Figure 32: Bus PLC throughput.....   | 32 |
| Figure 33: Path Set-Up Time using the Reactive mode of HWMP. ....                            | 33 |
| Figure 34: Convergence Time using the Proactive mode of HWMP .....                           | 33 |
| Figure 35: Control overhead in kbit for the Reactive and Hybrid modes of HWMP .....          | 34 |
| Figure 36: Control overhead in kbit/s for the Reactive and Hybrid modes of HWMP .....        | 35 |
| Figure 37: Average control traffic for attempt.....  | 35 |
| Figure 38: Testbed configuration.....  | 36 |
| Figure 39: TCP throughput measurement with iperf (output of jperf).....                      | 38 |
| Figure 40: Profiling output for the Data Plane (RAPTOR-X64 optimized version).....           | 38 |
| Figure 41: Experimental wireless mesh network for measurement of HWMP convergence time ..... | 39 |
| Figure 42: Failure of node B causing link break on path A-B-D .....                          | 39 |
| Figure 43: New path between nodes A and D through node C.....                                | 40 |
| Figure 44: Convergence Time of HWMP after link break.....                                    | 40 |

# 1 Simulation Results

## 1.1 Description of Tools and Methods used

### 1.1.1 Opnet Modeler

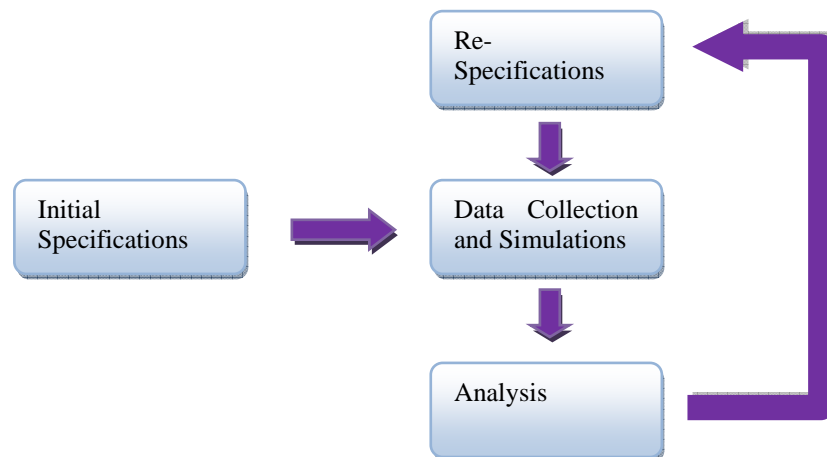
#### 1.1.1.1 Opnet Modeler general description

Opnet [OPNET] is a powerful network simulation tool that can be used to model communication systems and to predict network performance. It was originally developed at MIT (Massachusetts Institute of Technology) and was introduced in 1987 as the first commercial network simulation tool. Actually, it provides a comprehensive development environment supporting the modeling of communication networks and distributed systems. Both, the behavior and the performance of modeled systems can be analyzed by performing discrete event simulations. The Opnet environment incorporates tools for all the phases of a project, including modeling design, simulations, data collection and data analysis.

#### 1.1.1.2 Modeler architecture

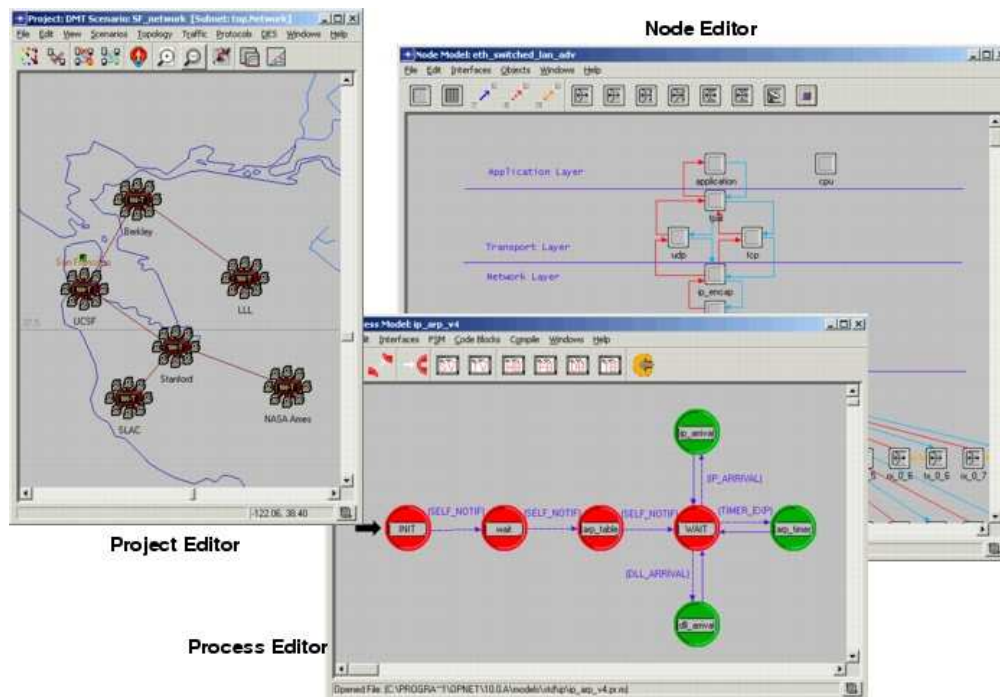
As previously stated, Opnet is provided with a number of editors and tools, each one focusing on particular aspects of modeling task. These tools fall into three major categories that correspond to the three phases of modeling and simulation projects: *specification*, *data collection and simulation* and *analysis of the results*. These three phases are necessarily performed in sequence and generally form a cycle, due to a return to the specification phase at the end of the analysis phase. Moreover, the specification phase is actually divided into two parts: initial specification phase and re-specification phase, with only the latter belonging to the cycle.

The simulation project cycle is depicted in the following figure:



**Figure 1: Modeling steps**

The workflow for Opnet (i.e. the steps required to build a specific network model) is based on three fundamental levels, the Project Editor, the Node Editor and the Process Editor. Opnet network models define the position and interconnection of communicating entities, or nodes. Each node is described by a block structured data flow diagram, or Opnet node model, which typically depicts the interrelation of processes, protocols and subsystems. Moreover, each programmable block in a node model has its functionality defined by an Opnet process model, which combines the graphical power of a state-transition diagram (FSM) with the flexibility of a standard programming language (C/C++) and a broad library of pre-defined modeling functions. Opnet makes use of graphical specification of models wherever appropriate. Thus, the model-specification editors all present a graphical interface in which the user manipulates objects representing the model components and structure.



**Figure 2: Opnet graphical user interfaces**

Each editor has its specific set of objects and operations that are appropriate for the modeling task on which it is focused. For instance, the Project Editor makes use of node and link objects; the Node Editor provides processors, queues, transmitters and receivers; the Process Editor is based on states and transitions. As a result, since no single paradigm of visual representation is ideally suited for all three of the above mentioned model types, the diagrams developed in each editor have a distinct appearance and Opnet models fit together in a hierarchical fashion.

The objective of most modeling efforts is to obtain measures of a system's performance or to make observations concerning a system's behavior. Opnet Modeler supports these activities by creating an executable model of the system. Provided that the model is sufficiently representative of the actual system, Opnet Modeler allows realistic estimates of performance and behavior to be obtained by executing simulations. Several mechanisms are provided to collect the desired data from one or more simulations of a system; for example, Opnet supports both local (related to an object) and global (related to the overall system) statistics and modelers can take advantage of the programmability of Opnet models to create proprietary forms of simulation output. Moreover, Opnet simulations can generate animations that can be viewed during the run, or played back afterwards. Several forms of predefined or automatic animations are provided (packet flows, node movement, state transitions, and statistics). In addition, detailed, customized animations can be programmed if desired.

The third phase of the simulation project involves examining the results collected during the simulation phase. Opnet provides basic access to this data in the Project Editor and more advanced capabilities in the Analysis Tool, which provides a graphical environment that allows users to view and manipulate data collected during simulation runs. In particular, standard and user-specified probes can be inserted at any point in a model to collect statistics. Simulation output collected by probes can be displayed graphically, viewed numerically, or exported to other software packages. First and second order statistics on each trace as well as confidence intervals can be automatically calculated. Opnet supports the display of data traces as time-series plots, histograms, probability density and cumulative distribution functions. Graphs (as with models at any level in the Opnet modeling hierarchy) may be output to a printer or saved as bitmap files to be included in reports or proposals.

### 1.1.1.3 Opnet Editors and tools

Opnet supports model specification with a number of tools or editors that capture the characteristics of a modeled system behavior. Because it is based on a suite of editors that address different aspects of a model, Opnet is able to offer specific capabilities to address the diverse issues encountered in networks and distributed systems. To present the model developer with an intuitive interface, these editors break down the required modeling information in a manner that parallels the structure of actual network systems. Thus, the model-specification editors are organized in an essentially hierarchical fashion. Model specifications performed in the Project Editor rely on elements specified in the Node Editor; in turn, when working in the Node Editor, the developer makes use of models defined in the Process Editor. The remaining editors are used to define various

data models, typically tables of values, which are later referenced by process or node level models. This organization is depicted in the following list:

- **Project Editor:** the Project Editor is used to construct and edit the topology of a communication network model. A network model contains only three fundamental types of objects: sub-networks, nodes, and links. There are several varieties of nodes and links, each offering different basic capabilities. In addition, each node or link is further specialized by its “model”, which determines its behavior and functionality. The Project editor also provides basic simulation and analysis capabilities. Finally, it is worth highlighting that the entire system to be simulated is specified by the corresponding network model.
- **Node Editor:** the Node Editor is used to specify the structure of device models. These device models can be instantiated as node objects in the Network Domain (such as computers, packet switches, and bridges). In addition to the structure, the node model developer defines the interface of a node model, which determines what aspects of the node model are visible to its user. This includes the attributes and statistics of the node model. Nodes are composed of several different types of objects called modules. At the node level, modules are “black boxes” with attributes that can be configured to control their behavior. Each one represents particular functions of the node operation and they can be active concurrently. Several types of connections (packet streams, statistical wires and logical associations) support flow of data and control information between the modules within a node.
- **Process Editor:** The Process Editor is used to specify the behavior of process models. Process models are instantiated as processes in the Node Domain and exist within processor and queue modules. Processes can be independently executing threads of control that perform general communications and data processing functions. They can represent functionalities that would be implemented both in hardware and in software. In addition to the behavior of a process, the process model developer defines the model interfaces, which determines what aspects of the process model are visible to its user. This includes the attributes and statistics of the process model. Process models use a finite state machine (FSM) paradigm to express behavior that depends on current state and new stimuli. FSMs are represented using a state transition diagram (STD) notation. The states of the process and the transitions between them are depicted as graphical objects.
- **Link Model Editor:** the Link Model Editor is used to create, edit, and view link models.
- **Packet Format Editor:** the Packet Format Editor is used to develop packet formats models. A packet format contains one or more fields, represented in the editor workspace as coloured rectangular boxes. The size of the box is proportional to the number of bits specified as the field size. Fields can assume fixed length or inherited length for simulation specific needs (e. g. payload encapsulation) and may be one of several types: integer, double, structure, information and packet.
- **ICI Editor:** the ICI Editor is used to create, edit, and view interface control information (ICI) formats. ICIs are used to communicate control information between processes.
- **Antenna Pattern Editor:** the Antenna Pattern Editor is used to create, edit, and view antenna patterns for transmitters and receivers (Modeler/Radio only).
- **Modulation Curve Editor:** the Modulation Curve Editor is used to create, edit, and view modulation curves for transmitters (Modeler/Radio only).
- **PDF Editor:** the PDF Editor is used to create, edit, and view probability density functions (PDFs). PDFs can be used to control certain events, such as the frequency of packet generation in a source module (Modeler only).

The next two sections are focused on the two principal domains and their editors: the Node Domain and the Process Domain because the implementation of Inter-MAC mainly relies on these two domains.

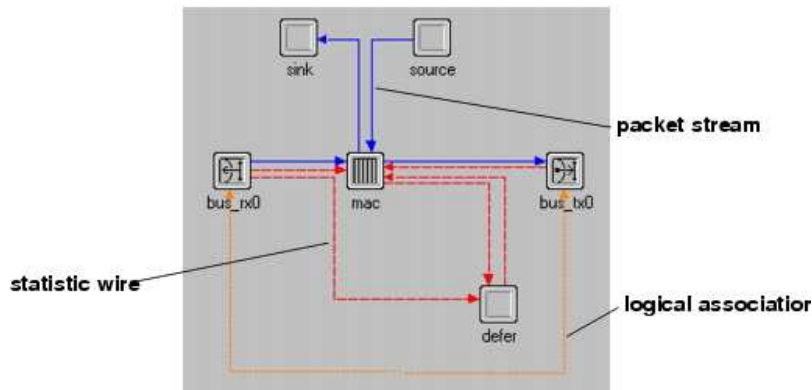
#### 1.1.1.4 Node Domain

The Node Domain provides for the modeling of communication, devices that can be deployed and interconnected at the network level. In Opnet Modeler terms, these devices are called nodes, and in the real world they may correspond to various types of computing and communicating equipment such as routers, bridges, workstations, terminals, mainframe computers, file servers, fast packet switches, satellites, and so on. Nodes are created as instances of node models, meaning that a node model is the blueprint for all of the individual nodes of a particular type.

Node models are developed in the Node Editor and are expressed in terms of smaller building blocks called *modules* representing distinct functional areas of the node. Some modules offer capability that is substantially predefined and can only be configured through a set of built-in parameters. These include various

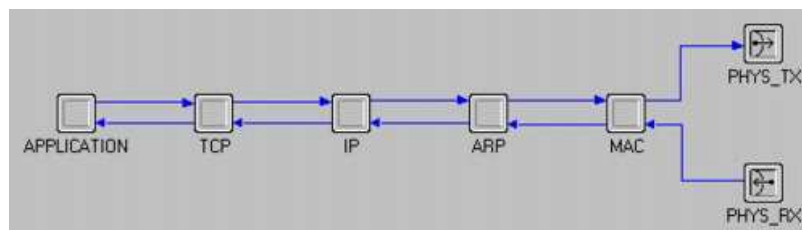
transmitters and receivers allowing a node to be attached to communication links in the network domain. Other modules, called *processors*, *queues*, and *external systems*, are highly programmable, their behavior being prescribed by an assigned *process model*. Process models are developed using the Process Editor.

A node model can consist of any number of modules of different types. Three types of connections are provided to support interaction between modules. These are called *packet streams*, *statistic wires*, and *logical associations*. Packet streams allow formatted messages called *packets* to be conveyed from one module to another. Statistic wires convey simple numeric signals or control information between modules, and are typically used when one module needs to monitor the performance or state of another. Both packet streams and statistic wires have parameters that may be set to configure aspects of their behavior. Logical associations identify a binding between modules. Currently, they are allowed only between transmitters and receivers to indicate that they should be used as a pair when attaching the node to a link in the Network Domain. Figure 3 shows a typical node model that includes the three types of connections.



**Figure 3: Node Model**

The modeling paradigm selected for the Node Domain was designed to support general modeling of high-level communication devices. It is particularly well suited to modeling arrangements of stacked or layered communication protocols. In the Node Editor, a device that relies on a particular stack of protocols can be modeled by creating a processor object for each layer of that stack and defining packet streams between neighboring layers, as shown in the following diagram for the familiar TCP/IP stack.



**Figure 4: Possible Representation of TCP/IP Protocol Stack**

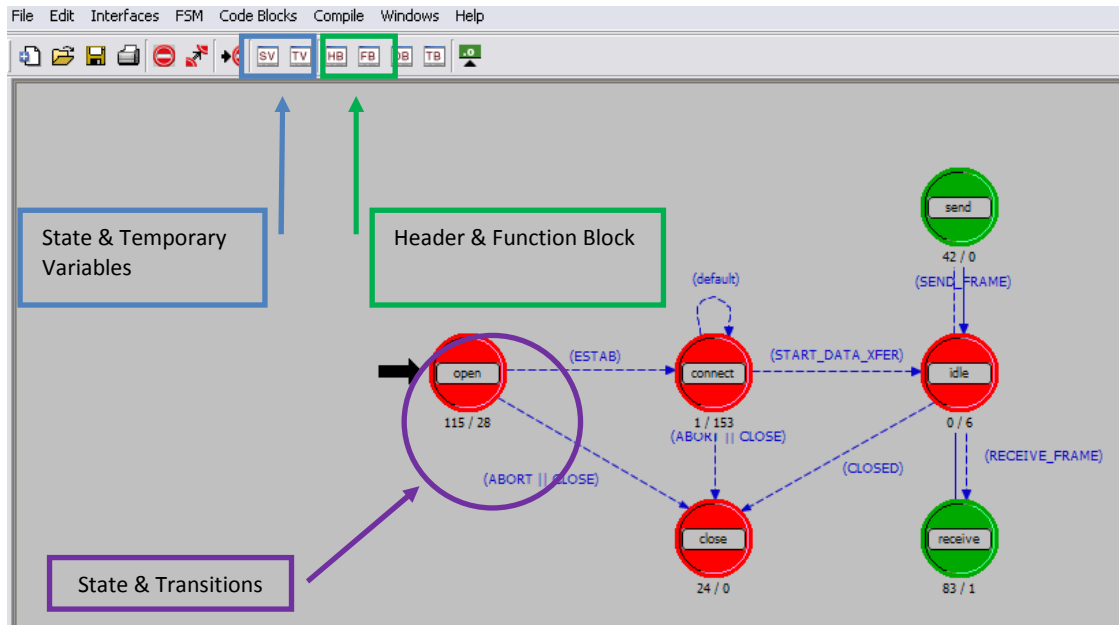
### 1.1.1.5 Process Domain

As indicated in 1.1.1.3, processor modules are user-programmable elements that are key elements of communication nodes. The tasks that these modules execute are called processes. Because it has a set of instructions and maintains state memory, a process is similar to an executing software program. Processes in Opnet Modeler are based on process models that are defined in the Process Editor. The relationship between process model and process is similar to the relationship between a program and a particular session of that program running as a task (in fact, the term process is used in many operating systems as well). Just as nodes created in the Project Editor are instances of node models defined with the Node Editor, each process that executes in a processor module is an instance of a particular process model.

The process modeling paradigm of Opnet Modeler supports the concepts of process groups. A process group consists of multiple processes that execute within the same processor or queue. When a simulation begins, each module has only one process, termed the root process. This process can later create new processes which can in turn create others as well, etc. When a process creates another one, it is termed the new process' parent; the new process is called the child of the process that created it. Processes that are created during the simulation are referred to as dynamic processes.

Figure 5: Process Editor shows a process model as it is defined in the Process Editor. The Process Editor expresses process models in a language called Proto-C, which is specifically designed to support development of

protocols and algorithms. Proto-C is based on a combination of state transition diagrams (STDs), a library of high-level commands known as Kernel Procedures, and the general facilities of the C or C++ programming language. A process model's STD defines a set of primary modes or states that the process can enter and, for each state, the conditions that would cause the process to move to another state.



**Figure 5: Process Editor**

Proto-C models consist of two basic component types: states and transitions, hence the name *state transition diagram* (STD). States are generally used to represent the top-level modes that a process can enter. Transitions specify the changes in state that are possible for the process. The state transition diagram representation of Proto-C is well suited to the specification of an interrupt-driven system because it methodically decomposes the states of the system and the processing that should take place at each interrupt. STDs developed in the Process Editor have a number of extensions beyond the capabilities offered by traditional state-transition diagram approaches. These include actions associated with each state, variable and attribute declarations, and common definitions of expressions and functions. The most relevant aspects of the process editor are described below:

- States** – In a general sense the word *state* refers to the information that a process may have accumulated over the time that it has existed. The process' state may not include all information to which the process has had access, but only that which it has chosen to retain. State information may be continually updated as new events occur and data becomes available. In terms of a state transition diagram (STD), the word *state* refers to an object that corresponds to one of the primary modes or situations that a process may find itself in. States are mutually exclusive and complementary, meaning that a process is always in exactly one state: more than one state may never be occupied at a time. The process can move between states in response to the interrupts that it receives. The transitions that depart from a state indicate which states may be occupied next, and the condition that each change requires. Specifications of actions may be associated with each Proto-C state. In Proto-C terminology, actions are called *executives*. The executives of a state are split into two sections, called *enter executives* and *exit executives*. Proto-C defines two types of states, called *forced* and *unforced*, that differ in execution-timing. In Proto-C diagrams, forced states are graphically represented as green circles, and unforced states are drawn as red circles.



**Figure 6: Forced and Unforced States**

Unforced states allow a pause between the enter executives and exit executives, and thus can model true states of a system. After a process has completed the enter executives of an unforced state, it blocks and returns control to the previous context that invoked it. If the process was invoked by the Simulation Kernel, blocking signifies the end of the current event and the Kernel may select a new event to begin its execution. At this point, the process remains suspended until a new invocation causes it to progress

into the exit executives of its current state. In all cases (except the initial invocation), when a process is invoked, it is poised to begin executing the exit executives of its current state. It then progresses until it completes the enter executives of an unforced state where it again blocks. Therefore if only unforced states are involved, the cycle is as follows:

1. exit executives of current state
2. transition to next state
3. enter executives of next state

- **Transitions** – Transitions describe the possible movement of a process from state to state and the conditions under which such changes may take place. There are four components to a transition's specification: a source state, a destination state, a condition expression, and an executive expression. The specification may be read as follows: when in the source state, if the condition is true, implement the executive expression and transfer control to the destination state. In the Process Editor, transitions are specified graphically. Each state may have any number of outgoing and incoming transitions depicted as directed arcs with the arrow pointing toward the destination state. A transition's condition is evaluated as a Boolean expression to decide whether or not the process should enter the transition's destination state. A process evaluates outgoing transitions after the exit executive statements of the source state have completed. For forced states, enter and exit executives are executed in immediate succession, and so both executive blocks and the transition evaluation can be viewed as one logical series of actions (although exit executives are usually empty for forced states). For unforced states, a pause occurs between entering and exiting executives, and so only the latter can be grouped with the transitions.
- **Variables** – Proto-C processes have access to several forms of memory to store information. Each form of memory has different properties that make it correct for particular uses. Some of these memory forms allow information to be declared and manipulated directly as ordinary names; these forms of storage are referred to as variables. Proto-C processes can make use of three distinct categories of variables called *state variables*, *temporary variables*, and *global variables*. The following table shows these types of variables and the sections of code where they can be accessed.

|                  |           | Visibility       |                             |   |
|------------------|-----------|------------------|-----------------------------|---|
|                  |           | State executives | Functions in function block | Functions in external files (.ex.c files) |
| Type of Variable | temporary | X                |                             |   |
|                  | state     | X                | X                           |   |
|                  | global    | X                | X                           | X   |

**Table 1: Variables**

- **Modularizing computations: the Function Block** – The most common and straightforward approach to modularizing recurring computations is to make use of the function call mechanism provided by the C and C++ languages. Functions (also referred to as procedures) are specifications of computations that may accept parameters, referred to as arguments. Arguments may be of any data type provided by C, C++, Opnet Modeler, or the user's custom definitions. Depending on the service that is required of them, functions may provide return values that result from their computations or simply do a series of actions and return no result. See a C or C++ reference text for more information about defining functions. In Proto-C, any number of functions may be specified within a section called the function block of a process model.
- **Attributes** – process model designers frequently specify characteristics of a process that require the flexibility to adapt to a variety of situations. For example, communication protocols may provide various options to their users, such as time-out durations, maximum number of retransmissions, window sizes, etc. Similarly, a computer operating system model may offer multiple alternatives for virtual memory size, maximum number of concurrent tasks, etc. Rather than design similar process models that differ in just these features, it is preferable to design parameterized process models. Those characteristics of the process that require variability can be viewed as parameters of the model. This technique fosters reuse of process models for various purposes by avoiding hardwired specification where possible. For instance, a process model that performs window-based flow control may be defined with the window size as an attribute, so that it is reusable in different situations requiring different values of the window size.

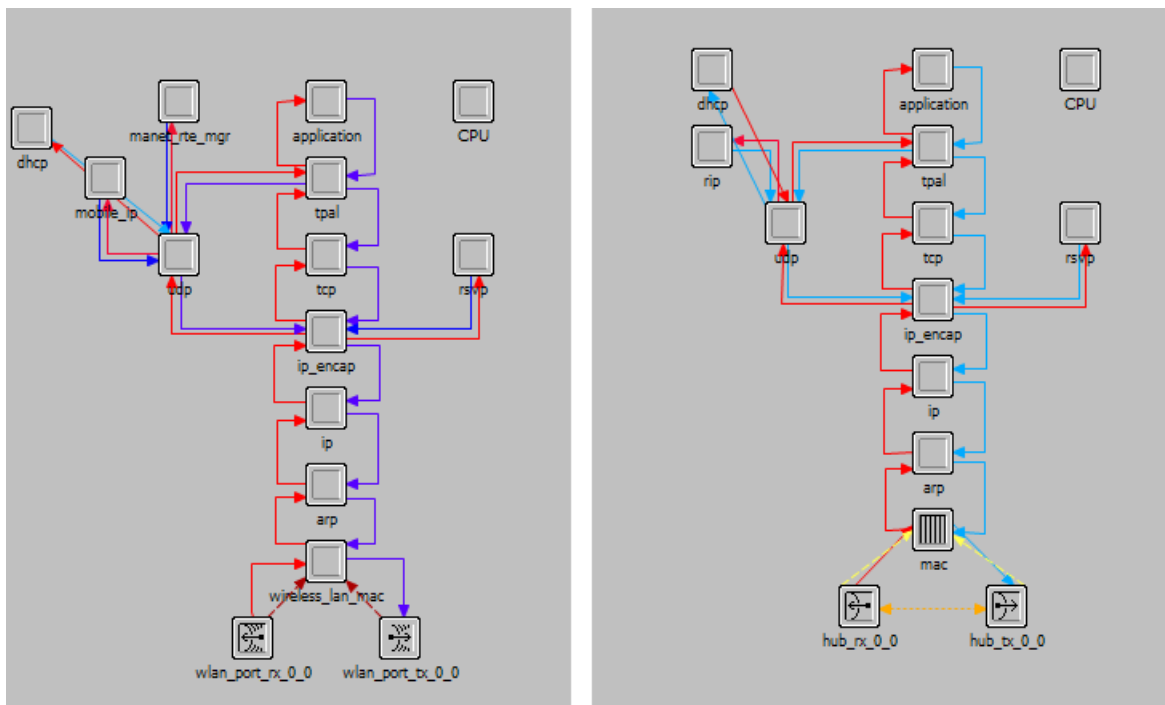
## 1.1.2 Inter-MAC implementation on Opnet

In order to simulate the behavior of an Omega Network inside Opnet Modeler comes up the necessity to create new node models i.e. the Omega Devices. Hence the implementation of the Inter-MAC layer in Opnet begins at the Node Domain. Given the fact that an Omega Device substantially is like any other classic workstation but has in addition the Inter-MAC layer, we created this new nodes starting from standard Opnet's node models. The Inter-MAC layer is implemented as a new module inside this workstation node models.

The main issue of this modification was the interconnection of the Inter-MAC module with the existing modules of the node model. This is described in section 1.1.2.1, and the description of the Inter-MAC process model is done in section 1.1.2.2.

### 1.1.2.1 The Omega device

An Omega device is a node model that integrates the Inter-MAC module in its stack of protocols. The construction of this device begins from the existing node models that have the classic ISO/OSI protocol stack. The most suitable node models in Opnet are the Ethernet/WLAN Workstations, because they have all the layers of the OSI model up to the application layer, which can be a FTP, e-mail, voice, video, or custom application. Figure 7 shows the original node models of Ethernet and WLAN workstations. It is important to remark that the ARP protocol occupies a distinct layer in Opnet node models stacks. The Inter-MAC layer will be inserted below the network layer and over the ARP layer. The choice to insert the process between the IP and ARP process models is due to the subsequent independence of each ARP table in interfacing with his own MAC layer. In fact, one main feature of Inter-MAC is to communicate at the same time with different technology MAC layers: it's therefore important that each MAC has its own ARP table to which communicate. Once the process is located in the middle of IP and ARP processes, next step is to correctly interconnect the Inter-MAC layer. The purpose is to make the Inter-MAC layer transparent to the IP layer, for this reason the Inter-MAC layer must implement all the interfaces that ARP has toward the IP layer and also the ones that IP has toward the ARP layer.



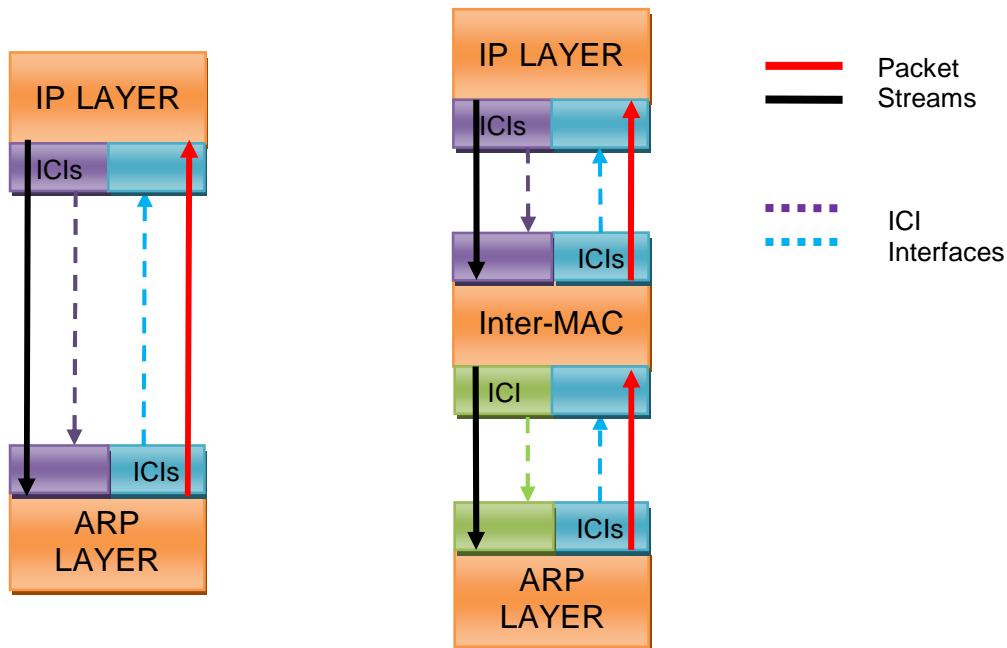
**Figure 7: WLAN/Ethernet Workstations**

In Opnet two processes can communicate with each other with different mechanisms.

- (i) Packet-Based communication – the most common one.
- (ii) Interface Control Information(ICI)
- (iii) Event States
- (iv) Statistic Wires
- (v) Link Models

Packets are transferred with packet streams, between the IP layer and ARP layer there are two packet streams, one for down going packets and another for the opposed way (in Figure 7 the packet streams are represented by the red and blue lines terminating with an arrow). Now these two packet streams will be split in the middle to interconnect the Inter-MAC layer. The Inter-MAC data plane will deal with these two packet streams. At one hand the data plane must encapsulate the incoming IP packets into the Inter-MAC frame and forward them to the appropriate ARP and at the other hand it must de-capsulate all the Inter-MAC frames coming from different ARPs, directed to the upper layers, and send them to the IP layer.

Interface Control Information structures or ICIs are data objects that resemble packets in that they consist of a list of data items. However, ICIs are simpler data structures than packets because they contain storage only for user-defined values. In addition, there is no notion of size or ownership for an ICI as there is for a packet. In fact, it is common for ICIs to be concurrently shared and modified/examined by multiple entities. So Inter-MAC has to recognize and read all the ICIs that the IP layer has defined for the ARP, and must implement the ICIs that the ARP layer has defined to communicate with the IP layer.

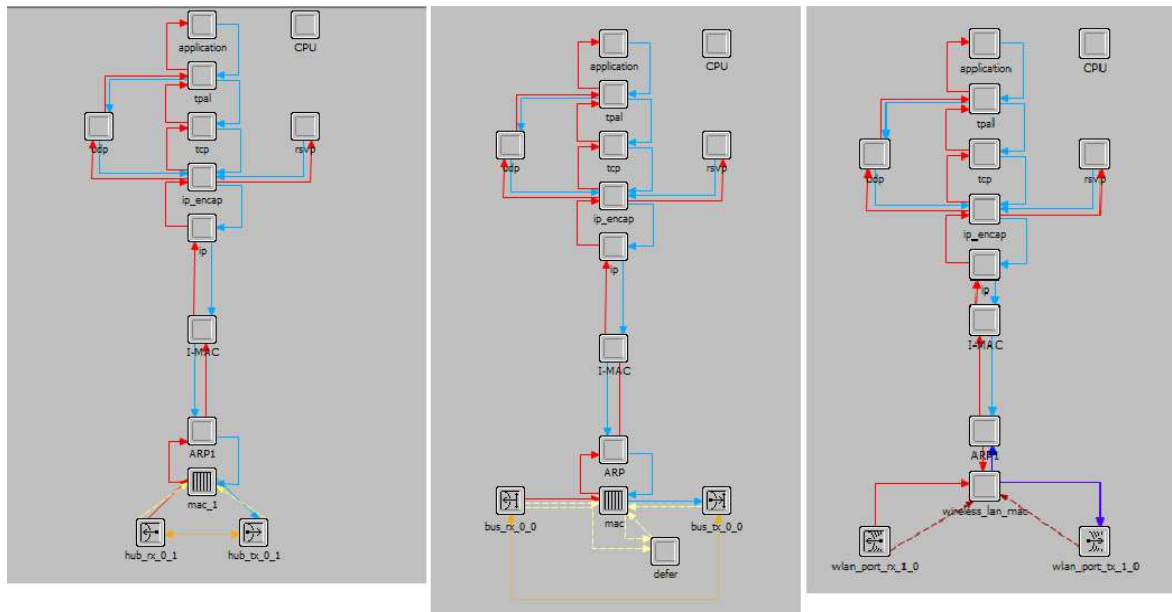


**Figure 8: Inter-MAC Interfacing**

Event states offer an alternative and faster method of associating data with an event than ICIs do. An event state is an arbitrary user-defined data structure that can be installed and thereafter is associated automatically with each generated event until a new event state is installed. Event states can be used for the same purposes as ICIs but the IP layer has not defined any event state toward the ARP layer neither statistic wires nor links. This interfacing is shown in Figure 8. This figure highlights the fact that it is created a new ICI structure just to communicate from the Inter-MAC toward the ARP layer. This new ICI structure differs from the one that the IP layer had specified because now the ARP does not have to make a research to find the next MAC address if it is not specified in the IP packet header. Every packet forwarded from the Inter-MAC data plane will have the next MAC address specified. There is needed also a modification in the ICI interfacing that the ARP has to make possible the recognition of the new ICI introduced by the Inter-MAC layer.

There is needed another modification: Each process model has a couple of functions (neighbor connection find) which target is to search for upper or lower layers attached to them and start the communication. Moreover, in the original node model each layer knows by means of proper information, which layer is located over it and under it. We can therefore summarize the task needed to solve this problem as follows:

- create appropriate function in Inter-MAC process model;
- modify pre-existent functions in the IP and ARP process model.



**Figure 9: Single technology Omega Devices: Ethernet, PLC and Wi-Fi**

The first task is accomplished by implementing the same functions present in IP and ARP layers, modifying parameters which give information about neighboring layers. Inter-MAC must know that its position is between layer 2 and layer 3 of the protocol stack.

The second task is related to the first, because once Inter-MAC functions are modified to recognize his neighboring layers, we must modify the function present in the layers interfacing with it.

Three Workstation Node Models are implemented with a single underlying technology:

1. Ethernet Inter-MAC workstation node model
2. WLAN Inter-MAC workstation node model
3. Bus Inter-MAC workstation node model

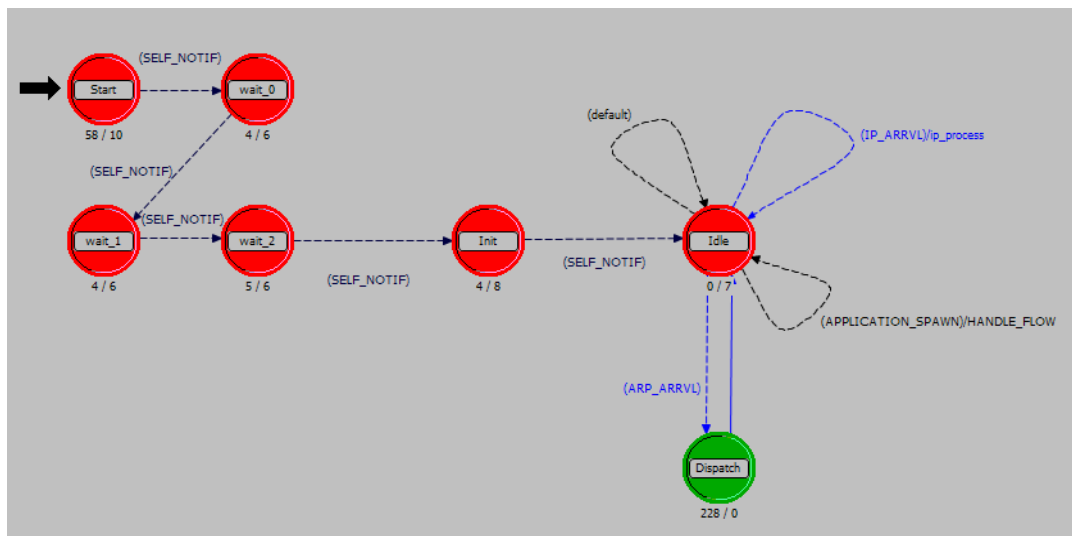
Since Power Line Communication, Hybrid Wireless Optics and Ultra Wide Band are not implemented in Opnet Simulator then only Ethernet, WLAN and Buses can be used in modeling network nodes. Next step is to connect to I-MAC module more than one single technology interface, as shown in Figure 10 where one BUS interface, one Ethernet interface and one WLAN interface are equipped in a network node. Inter-MAC code is implemented in a modular mode. Then, connecting to the I-MAC process one of the blocks below it (ARP, MAC and transmitter processes) the network node can correctly execute its functionalities.



All these engines are executed in separate processes and share a block of memory whose structure is interpreted in the `Intermac_Support.h` header file. The communication arguments needed when a process invokes another are also present in this header file. Generally, these arguments are blocks of memory and they are described by structures types.

Once all engines are created, the process passes to the Idle state and waits for different events that can be:

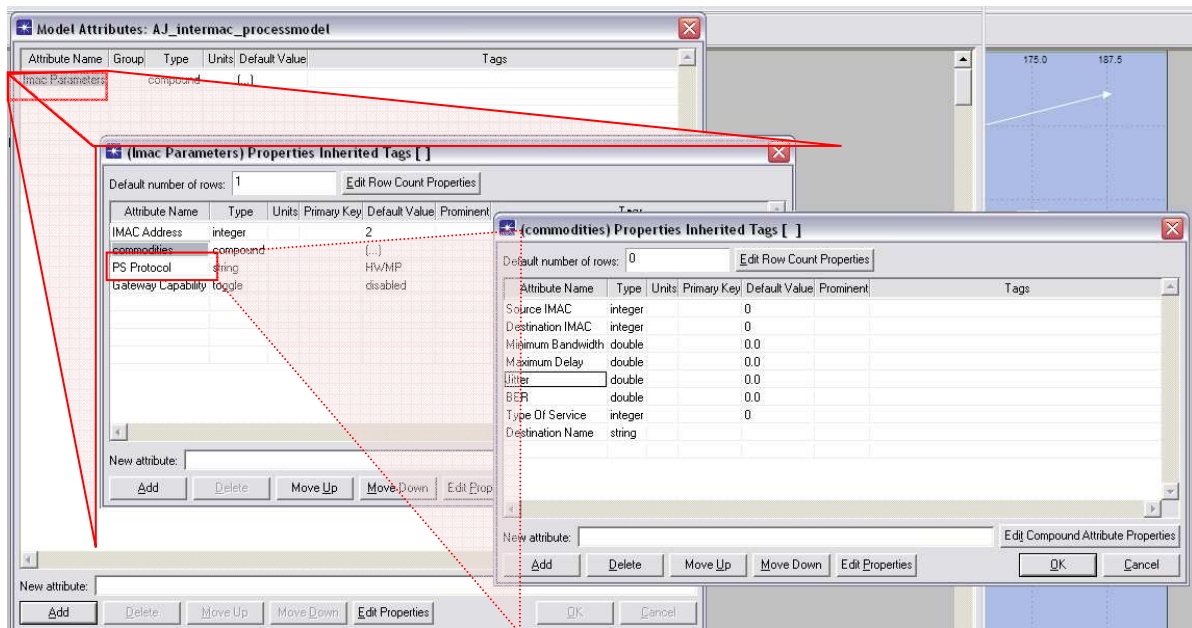
- IP packet arrival – Encapsulates these packets into the Inter-MAC frame and sends them to the Forwarding Engine.
- ARP packet arrival – Determines the type of message and sends it to the proper engine or if it is a data packet it is de-capsulated and sent to the IP layer.
- Signaling from the application layer – This interrupt is steered to the QoS Engine to deal with it. Actually in Opnet the application layer can communicate only the destination node, type of service and the application name, but does not provide any TSPEC information. This is the only static information that is entered into the scenario. The TSPEC are defined as attributes of each Omega Device.



**Figure 11: Root Process Model**

These are the only tasks of the Inter-MAC root process model.

The attributes of this process model are depicted in the following picture. The *Imac Parameters* is a compound attribute of four other *child* attributes. The *IMAC Address* is the attribute where the user sets the value of the Inter-MAC address of every Omega Device. *PS Protocol* is intended for future use, in order to switch the path selection protocol of the scenario if another type of path selection protocol is required in the simulations. *Destination* and *Source IMAC* together with the *Type Of Service* attribute are needed in order that the process can associate the application signaling to the commodity it belongs to if more than one application will run from the same node. The TSPEC are compound of four double values: *Minimum bandwidth*, *Maximum Delay*, *Jitter* and *BER*.



**Figure 12: Inter-Mac attributes**

### 1.1.2.3 Path Selection Engine Process

This process implements the HWMP protocol that is the protocol proposed for Wireless Mesh Networks standard IEEE 802.11s. All packet formats, messages of the protocol and all timing parameters used in these simulations are based on document P802.11s<sup>TM</sup> D3.04 [IEEE802.11s] .

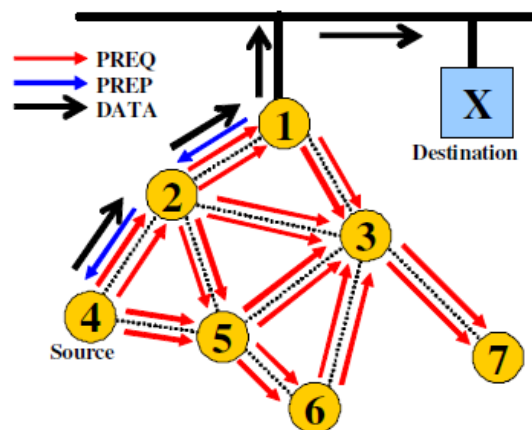
The Hybrid Wireless Mesh Protocol (HWMP) is a mesh path selection protocol that combines the flexibility of reactive path selection with proactive topology tree extensions. The combination of reactive and proactive elements of HWMP enables efficient path selection in a wide variety of mesh networks (with or without access to the infrastructure). HWMP uses a common set of protocol primitives, generation and processing rules inspired by Ad Hoc On Demand Distance Vector (AODV) protocol [RFC3561] adapted for MAC address-based path selection and link metric awareness. HWMP supports two modes of operation depending on the configuration. These modes provide different levels of functionality:

- *Reactive mode:* The functionality of this mode is always available. It allows mesh stations to communicate using peer-to-peer paths. The mode is used in situations where there is no root mesh node configured. It is also used if there is a root mesh node configured but a reactive path can provide a better path to a given destination in the mesh.
- *Proactive tree building mode:* In this mode, additional proactive tree building functionality is added to the reactive mode. This can be performed by configuring a mesh station as root mesh node using either the proactive PREQ or RANN mechanism.

These modes are not exclusive: reactive and proactive modes are used concurrently, because the proactive modes are extensions of the reactive mode. One example of concurrent usage of reactive and proactive mode is for two nodes that are part of the Omega network to begin communicating using the proactively built tree but subsequently to perform a reactive discovery for a direct path between each other. The use of a hybrid protocol on the Omega network allows communication to begin immediately thanks to the use of the proactive component in HWMP, reducing the initial delay that other path selection protocols as AODV introduce. Moreover, since the reactive component of the protocol begins after the initial data has begun transmission, it allows discovery of an optimal path for the flow. Also, any traffic that is addressed to an external network (e.g. Internet Traffic) will always have the optimal path since the Omega gateway (root node) builds a proactive tree creating optimal paths from every node to itself, further reducing packet overhead since the reactive component does not have to be initiated.

All HWMP modes of operation utilize common processing rules and primitives. HWMP information elements are the Path Request (PREQ), Path Reply (PREP), Path Error (PERR) and Root Announcement (RANN). The metric cost of the links determines which paths HWMP builds. In order to propagate the metric information between mesh nodes, a metric field is used in the PREQ, PREP and RANN elements. Path selection in HWMP uses a sequence number mechanism to assure that the mesh nodes can distinguish current path information from stale path information at all times in order to maintain loop-free connectivity.

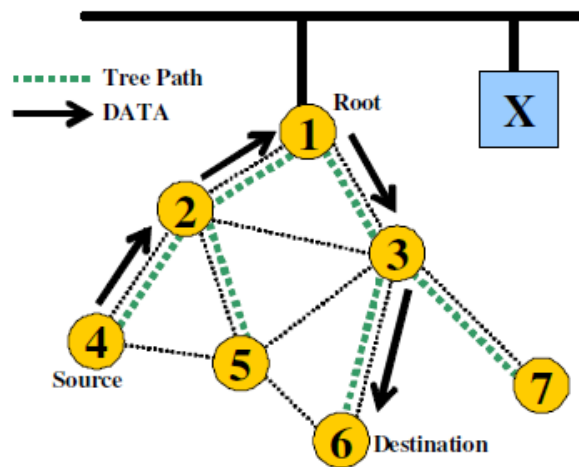
**Reactive Mode** If a source mesh node needs to find a path to a destination mesh node using the reactive path selection mode, it broadcasts a PREQ with the target mesh node specified in the list of targets and the metric field initialized to the initial value of the active path selection metric. When a mesh node receives a new PREQ it creates or updates its path information to the originator mesh node and propagates the PREQ to its neighbor peer mesh nodes if the PREQ contains a greater HWMP sequence number, or the HWMP sequence number is the same as the current path and the PREQ offers a better metric than the current path. Each mesh node may receive multiple copies of the same PREQ that originated at the originator mesh node, each PREQ traversing a unique path. Whenever a mesh node propagates a PREQ, the metric field in the PREQ is updated to reflect the cumulative metric of the path to the originator mesh node. After creating or updating a path to the originator mesh node, the target mesh node sends an individually addressed PREP back to the originator mesh node. The PREQ provides “Target Only” (TO) and “Reply and Forward” (RF) flags that allow path selection to take advantage of existing paths to the target mesh node by allowing an intermediate mesh node to return a PREP to the originator mesh node. If the TO flag is set to 1, only the target mesh node sends a PREP. The effect of setting the TO flag to 0 is the quick establishment of a path using the PREP generated by an intermediate mesh node, allowing the forwarding of data frames with a low path selection delay. The effect of setting the RF flag to 1 (in addition to setting the TO flag to 0) is the selection (or validation) of the best path after the path selection procedure has completed. If the RF flag is set to 1 (and the TO flag to 0), the first intermediate mesh node that has a path to the target sends a PREP and propagates the PREQ with the TO flag set to 1 to avoid all other intermediate mesh nodes on the way to the target sending a PREP. If the RF flag is set to 0 (and the TO flag to 0), the PREQ is not propagated by the mesh node. Intermediate mesh nodes create a path to the target mesh node on receiving the PREP, and also forward the PREP toward the originator. When the originator receives the PREP, it creates a path to the target mesh node. If the target mesh node receives further PREQs with a better metric, then the target updates its path to the originator to the new path and also sends a new PREP to the originator along the updated path. A bidirectional, best metric end-to-end path is established between the originator and target mesh node. The metrics considered in this mode can vary from scenario to scenario. If the scenario contains only wireless mesh nodes the recommended metric is the Air-Link-Time basically a sort of delay. In heterogeneous technologies scenarios additional metrics can be used such as bandwidth, jitter, BER etc. The only drawback of this mode is its higher relative initial delay of the flows which is explained by Figure 13.



**Figure 13: The problem of high initial latency in the on-demand routing mode**

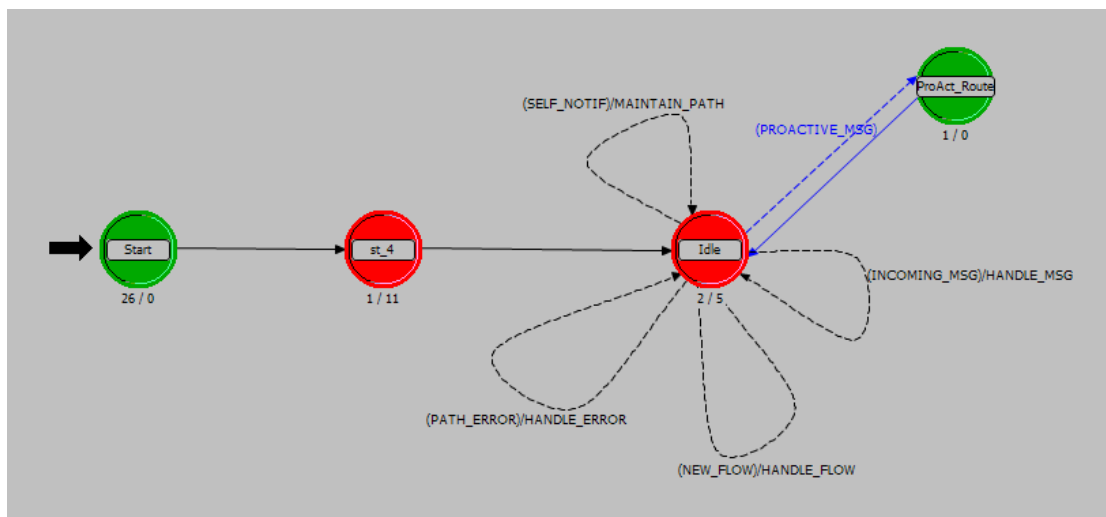
**Proactive Tree Building Mode** The PREQ tree building process begins with a proactive Path Request element sent by the root mesh node, with the Target Address set to all ones, the TO flag set to 1 and the RF flag set to 1. The PREQ contains the path metric (set to the initial value of the active path selection metric by the root mesh node) and an HWMP sequence number. The proactive PREQ is sent periodically by the root mesh node, with increasing HWMP sequence numbers. A mesh node receiving a proactive PREQ creates or updates its forwarding information to the root mesh node, updates the metric and hop count of the PREQ, records the metric and hop count to the root mesh node, and then transmits the updated PREQ. Information about the presence of and distance to available root mesh node(s) is disseminated to all mesh nodes in the network. Each mesh node may receive multiple copies of a proactive PREQ, each traversing a unique path from the root mesh node to the mesh node. A mesh node updates its current path to the root mesh node if and only if the PREQ contains a greater HWMP sequence number, or the HWMP sequence number is the same as the current path and the PREQ offers a better metric than the current path to the root mesh node. The processing of the proactive PREQ is the same as in the reactive mode previously described. If the proactive PREQ is sent with the “Proactive PREP” bit set to 0, the recipient mesh node may send a proactive PREP if a bidirectional path is required (for example, if the mesh node has data to send to the root mesh node that requires establishing a bidirectional path with the root mesh node). During the time a bidirectional path is required, the recipient mesh node shall send a proactive

PREP even if the “Proactive PREP” bit is set to 0. If the PREQ is sent with a “Proactive PREP” bit set to 1, the recipient mesh node shall send a proactive PREP. The proactive PREP establishes the path from the root mesh node to the mesh node. Given the fact that there is no information about the future flows that are going to be established in the network a simple metric can be used to build up the tree, such as the hop count.



**Figure 14: The problem of non optimum routing paths in the proactive routing**

This logic models the behavior of the Path Selection engine. In order to use the protocol’s specific messages there are defined additional types of packet suitable to carry this control messages.



**Figure 15: Path Selection Engine process**

Figure 15 depicts the process model of the Path Selection Engine. In the forced initial state *Start* the process initializes a part of its state variables. At the next unforced state it waits for the parent model to finish creating all other engines in every node in the scenario. Exiting this wait state the process reads the shared memory because at this moment all the other process handles are available.

Basically, the behavior of the process depends in the attribute *Gateway capability* of the node that it resides. This toggle attribute specifies if the node will be a gateway in the scenario and therefore will begin building the proactive tree. The gateway capability can be set to *disabled* in all the nodes in the scenario and in this way the proactive mode is not available. More than one node can be a gateway, but in the Omega scenarios it is assumed that there exists only one. The node that has been designated as a gateway begins sending periodically *Proactive Preq* messages. The nodes that receive them learn the best route to the gateway. Optionally, the nodes do not respond with *Proactive Prep messages* but to make possible the cooperation of the two nodes it is needed a path from the gateway to every node. After all the nodes have replied to the path requests messages the proactive tree is available at the root node. So in a scenario with at least a gateway the other nodes periodically receive *Proactive preqs* and reply to them. If a proactive path has been established in the scenario and the destination is a node inside the network it is gateway’s task to inform the source that can begin a reactive path request to find the optimal path to the destination. The gateway’s root process realizes that a new flow id is received and it is running on a proactive path so it creates a signaling message destined to the source in order to start the reactive procedure.

The reactive node can be triggered by two types of events:

- If no gateway is available in the network this process is invoked directly by the QoS Engine which in turn is invoked by the application layer. The QoS Engine will provide the destination address and the TSPEC read from the commodities attributes.
- If a proactive path is destined to an inner node the root process of the source after having received the signaling message from the gateway informs the QoS engine that a previous flow already running can find a better path with a reactive path request.

This process reads the current metrics of the links in the *local link table* that is maintained by the Monitoring Engine. Besides this indirect interaction with the Monitoring Engine, these two processes communicate together in other two cases. A node receiving or creating a path response must inform the Monitoring Engine to begin monitoring it after having accepted it. The Monitoring Engine process will be explained in the next section. Only reactive paths are being monitored because the proactive path is a best effort one. If any monitored flow TSPEC are not satisfied any more (by severe link congestion or by some type of failure) the Monitoring Engines invokes the Path Selection Engine in order to create a *path error message (PERR)*. The path error propagates eventually from any intermediate node to the source node of the flow affected. The source node starts another path requests and if an alternative route is available the traffic is re-routed. Upon the reception of a path error message the Path Selection Engine inform the monitoring engine to stop monitoring the flow for which it received the path error. In this way no other path errors are created.

This process output is the *Forwarding Table* which is used by the Forwarding Engine in order to forward packets to the proper node and interface. All the outgoing messages of this process are sent directly to the Forwarding Engine by invoking it. The received messages come from the root process, because it has the role of the dispatcher.

#### 1.1.2.4 Monitoring Engine Process

The Monitoring Engine is responsible for the monitoring of the status of links, neighbors and flows. It is the process that builds the *local link table* and populates its entries. The only tool available for the monitoring engine in order to accomplish these tasks is the probe frame. Varying the frequency of the probe frames this engine can monitor the status of links and flows.

From the first instant that is creating this process, it starts sending probe frames in order to discover the neighbor nodes. The probe frames go through only a single hop and their frequency is typically 1 probe per second. In this way every node can build the table of the metrics for all reachable one-hop neighbors. The probe frame contains inside a single value the creation time. This time together with the reception time can be used by the neighbor node to measure the delay and jitter of the sending node. The delay measurement is not a valid metric in real life scenarios because there is no synchronization between nodes. But here we have a simulating tool and only for simulation purpose we can use that value. Jitter is calculated as the packet delay variation and measured by storing consecutive reception instants of probe frames. In order to have valid jitter values some probe frames must be received. So from the beginning of the simulation it must past some time to have reliable jitter values. The bandwidth information is taken from the MAC module attributes. This is clearly a nominal value and is not an objective measurement. In fact the primary compared metric is delay and jitter. If there exist two paths with the same delay or nearly the same it is chosen the path that after adding the required bandwidth of the flow the remaining free band is bigger in percentage in relation with the nominal bandwidth. In Opnet the Bit Error Rate is a private value of the physical module and is used only for statistics of the physical layer.

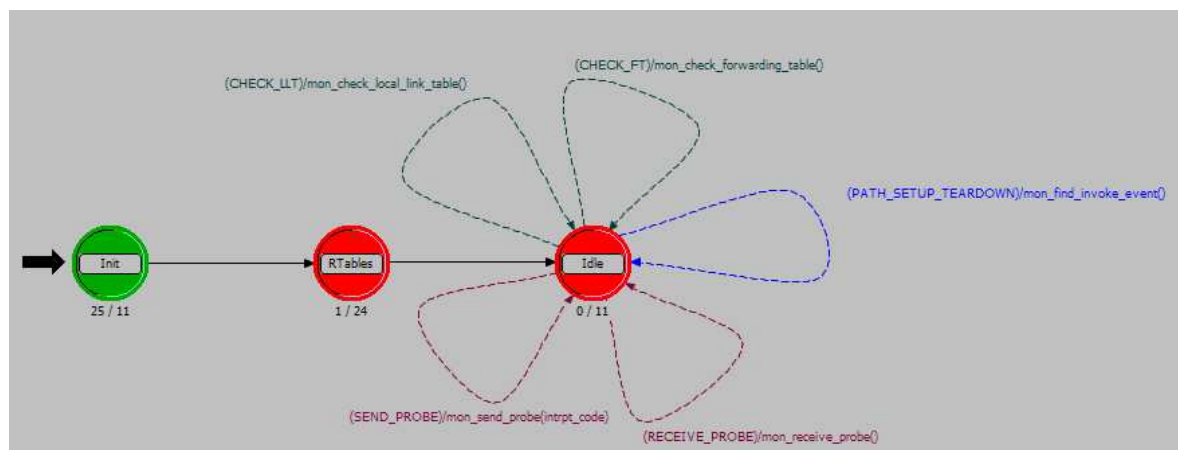


Figure 16: Monitoring Engine process

The flows to be monitored are added by the Path Selection Engine. The interfaces that have flows running on them either incoming or outgoing are probed with higher frequency. The monitoring probe frequency can be parameterized and must take into account two main factors:

- a. The traffic produced – the higher the probing frequency the bigger is the traffic produced.
- b. The delay of the link – can vary from a bunch of milliseconds to higher values such as 50 or even 100 ms in wireless cases. The Monitoring Engine must take into account this fact and must be aware not to give false alarms.

The detection of link failures and congestions depends on the probing frequency. So to be sure not to have false alarms the probe period is chosen to 50 ms and it is assumed that a link failure has occurred if for 250 ms no probe is received. Monitoring Engine invokes the Path Selection Engine to propagate a path error message if a link failure is detected. When a path is created or released the Monitoring Engine is informed by the Path Selection Engine. It reserves the required bandwidth of the flow in the interfaces that it goes in and out the node. This is done in order to calculate the metric that calculates the percentage of free bandwidth in a given link.

The process model of this engine is depicted in Figure 16. This process model has a forced initial state in order to initialize its state variables. At the first unforced state it waits for the root process to create the other engines and so to obtain their handles. In the *Idle* state this process handles the following events:

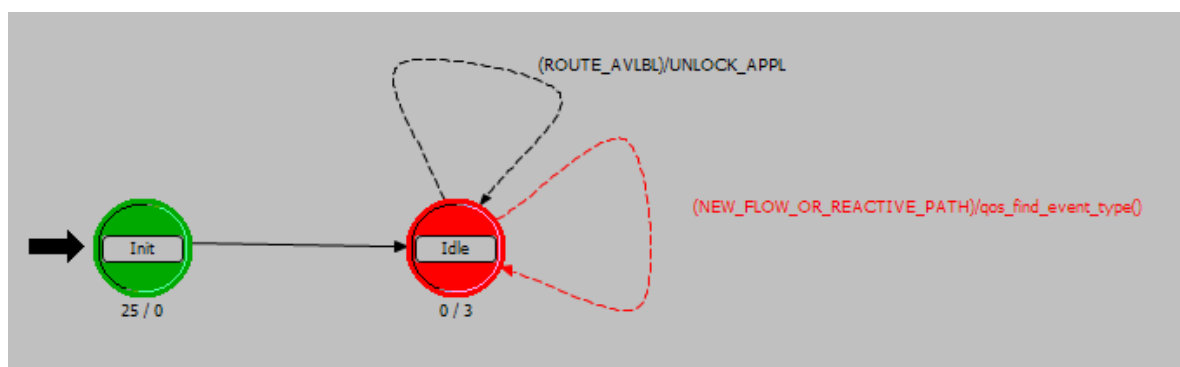
- i. Send Probe
- ii. Receive Probe
- iii. Check Local Link Table
- iv. Check Forwarding Table
- v. Path Setup/Teardown

The main output of this process is the *Local Link Table* which is used by the Path Selection Engine in order to evaluate the metrics of the available paths during path setups. The entries are periodically checked in order to find neighbors from which no probes are received. There is done even a check in the Forwarding Table entries. If an entry has not been used for a certain amount of time it is deleted by the Monitoring Engine.

#### 1.1.2.5 QoS Engine Process

This engine task is to deal with the signaling interrupts that come from the application plane. In this interrupts the destination address and type of service is specified. This process searching through the commodities attribute finds the proper commodity and stores the TSPEC requirements of this flow. If there already exists any proactive entry in the Forwarding Engine for the destination node this process directly informs by an interrupt the application layer in order to start sending packets. If no entry exists it invokes the Path Selection Engine providing the destination address and the required metrics of the flow to be established.

Another event that can happen is that a flow established on a proactive path must be re-routed in a reactive path. The QoS Engine is invoked by the root process that specifies for which flow the reactive path can be established. Its process model is represented in the Figure 17.



**Figure 17: QoS Engine Process**

#### 1.1.2.6 Forwarding Engine Process

This is an engine that belongs to the data plane. It has only a single task that is sequenced in the following steps.

- i. For every invoking process take the argument memory of the invocation and the identity of the invoking process. The identity of the invoking process is needed in order to interpret the memory structure.
- ii. Cast this memory in the appropriate structure by using the declarations included in the common header file *Intermac\_support.h*. The memory structure contains always a Packet component that is the packet to be sent.
- iii. Find the forwarding table entry. This search is done using only the Inter-MAC destination address for control packets but for data packets is used in addition the Flow ID.
- iv. Find the outgoing interface from the forwarding table entry.
- v. Send the packet to the ARP layer.

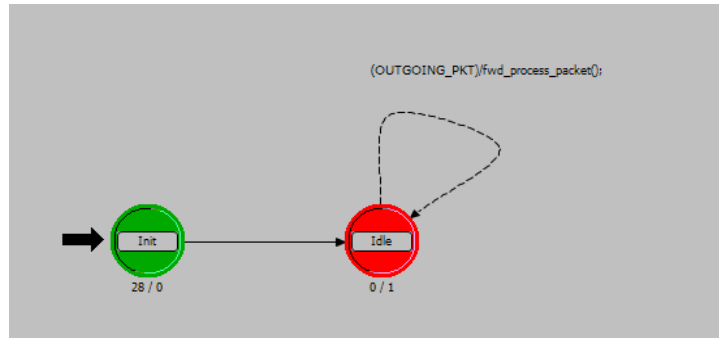


Figure 18: Forwarding Engine process

## 1.2 Simulation Setup

### 1.2.1 Simulation configuration

In order to build up a simulation and thus a new *Scenario* in Opnet some steps must be followed in order to configure it.

- a. **Scenario definition** – The first thing to do is to choose a scenario type that can be: Office, Campus, Enterprise, World or Home. Choosing home and its dimensions the next step is to place the Omega nodes in it. After the definition of the basic Omega node models a new palette of objects is available in Opnet. This palette includes the basic Omega nodes and also the new type of link which represent the PLC technology with 200 mbps capacity. Using the palette the nodes can be placed anywhere in the scenario. After this the nodes are inter-connected between them with the appropriate types of links. Figure 19 shows a simple scenario with 4 Omega nodes two of which are inter-connected with an Ethernet cable. The black line represents the PLC bus to which every node with a PLC interface can be connected with a bus tap. Furthermore, the personalized Inter-MAC palette is shown in this figure, too.

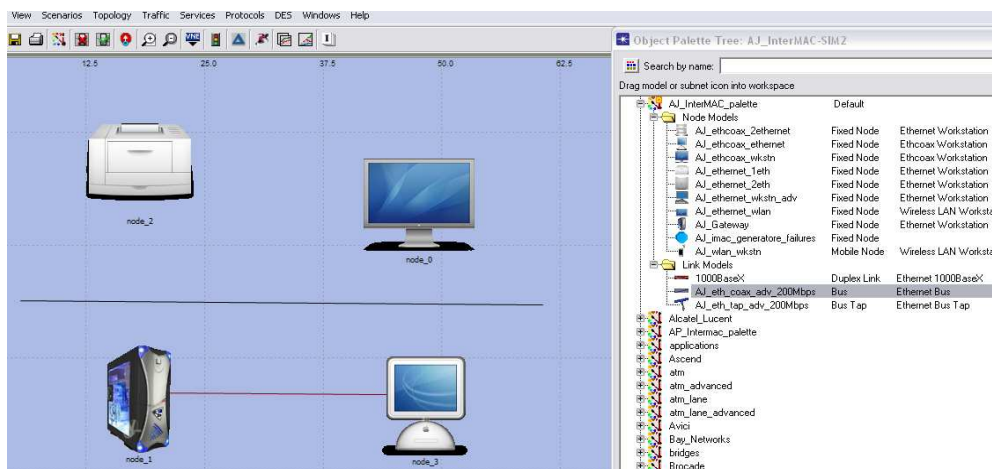


Figure 19: Placement of nodes and links

- b. **Attributes configuration** – All the Omega nodes have some attributes that have to be set manually. The first one is the Omega address. In Opnet it is chosen as an integer attribute in order to define only a unique identifier at the Inter-MAC layer. The Omega addresses cannot take values lower than two

because the value 1 is used as a broadcast address. Another attribute is the *commodities* attribute, which must be filled at every source node in order to provide the QoS Engine with the required TSPEC. This is shown in Figure 20.

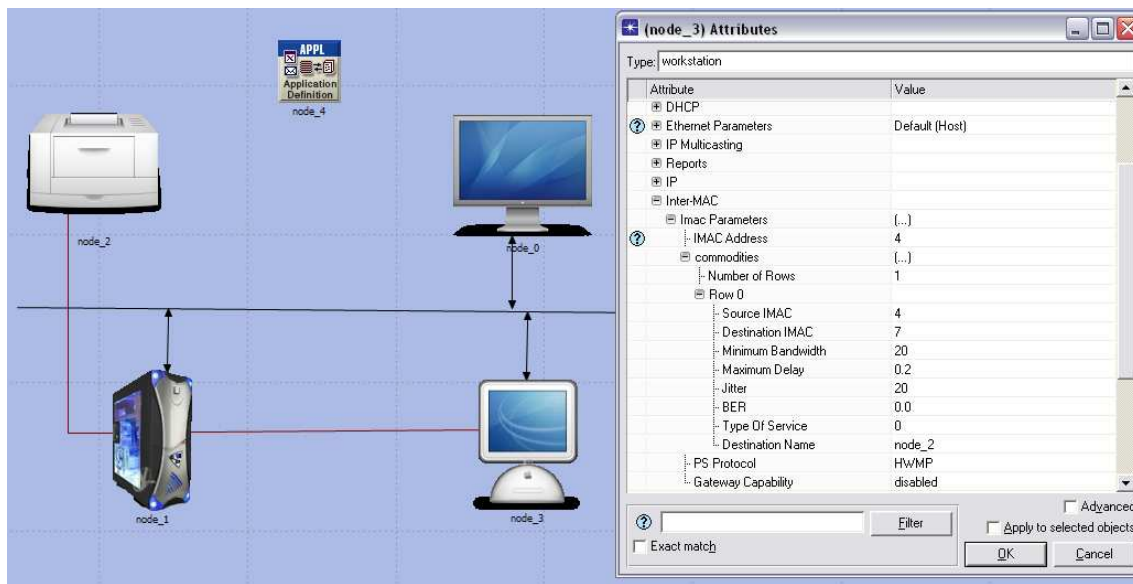


Figure 20: Inter-MAC attributes

- c. **Application definition** – The applications are defined in a special node: The *Application Definition* one which can be inserted from the network palette. Opnet Modeler provides several application configuration models and modeling frameworks, each targeted to speeding up model development for specific modeling requirements. Besides the default application there can be defined custom applications by taking as reference the existing ones or totally new applications. For example a video streaming application configuration is showed in the next figure. It can be noted that it is of the default type video conferencing. The basic attributes of this type are the frame inter-arrival time information, the frame size in bytes and the type of service. The size of the outgoing frames is set to NONE in order to model only a one way video streaming.

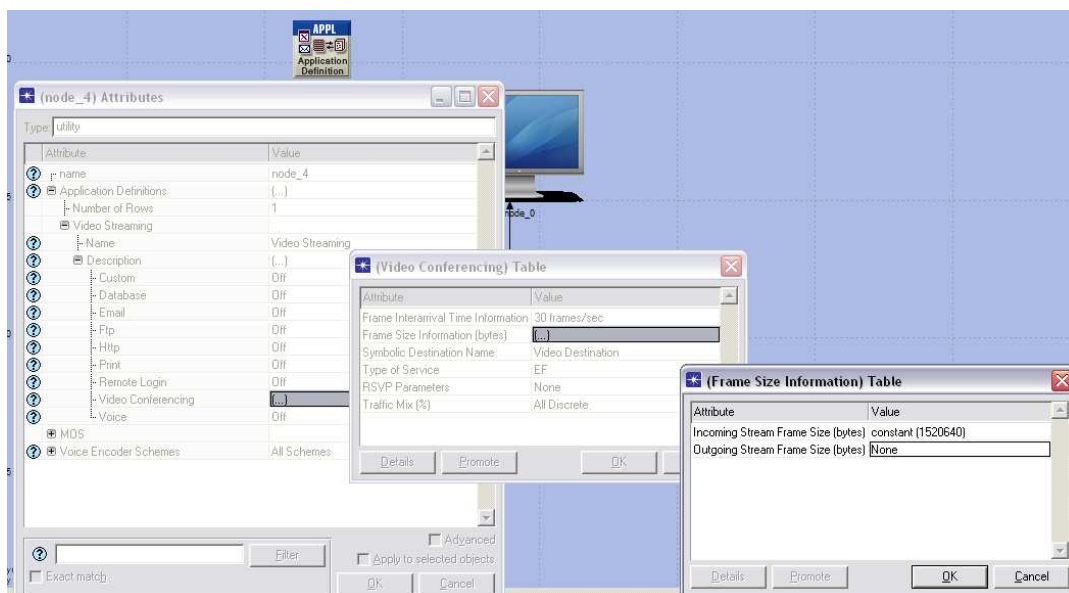
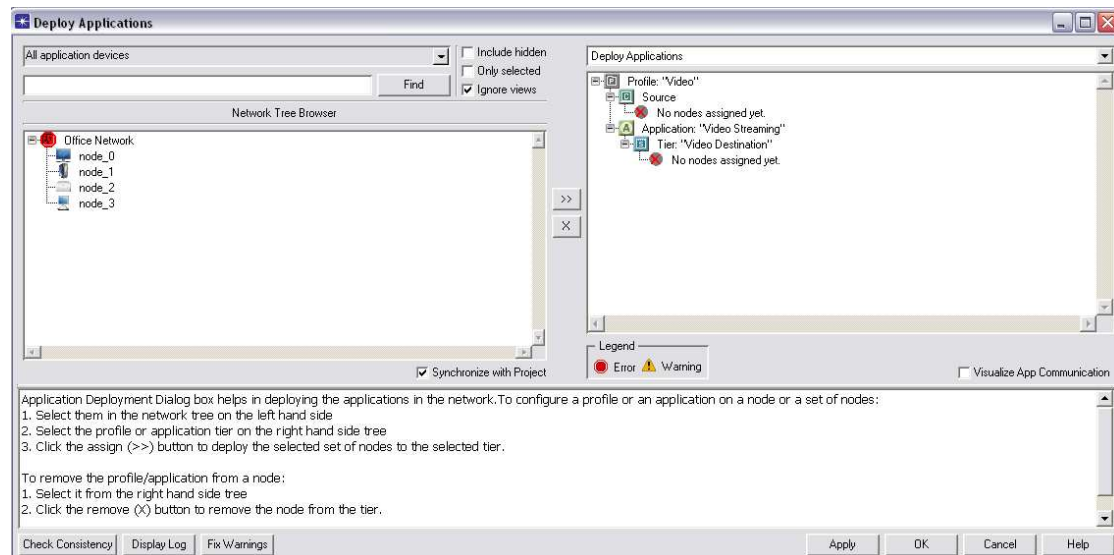


Figure 21: Defining a new application

After this operation the defined application must be deployed i.e. the source and destination node must be defined. There is another tool to do this:



**Figure 22: Deploying Applications**

The desired nodes selected on the left list can be assigned to the source or destination node on the applications on the right.

- d. **Configure wireless nodes** – In order to model a wireless mesh network the communication radius must be defined for wireless nodes. The propagation model used in indoor scenarios is the free-space propagation model. For every node is defined its receiver sensitivity and transmit power. The antenna gains are all set to 1 (0 dB) so the only variable parameter is the distance between two nodes. So the communication range can be chosen by varying the transmitting power and receivers sensibility.
- e. **Collecting statistics** – The remaining task to be done is to choose the desired statistics that are intended to be collated at every node. Statistics can vary from low level throughput to packets transmitted, received, lost, dropped and even higher layers statistics such as TCP retransmissions, packet end to end delay, variation delay etc.

Once this last task is done, the scenario is ready to be simulated. To start the simulation it must be launched: DES → Configure/Run Discrete Event Simulation... in the next window it can be chosen the simulation time, a seed value in order to randomize the simulation and the type of the execution Kernel. Normally, the Optimized Kernel is chosen, because it is faster and has limited console outputs. In the modeling and debugging phase the type of Kernel is used, i.e., the development one.

### 1.2.1.1 My media follows me – Scenario<sup>1</sup>

The scenario is displayed in the following figure. It is composed of 13 Omega nodes that have more than one physical interface and are inter-connected between them by heterogeneous technologies. The black line and its ramifications represent the PLC technology as usually, while the red links are gigabit Ethernet ones. The laptop, notebook, gateway, PDA and Smartphone node has also a wireless IEEE 802.11g interface at a rate of 54 Mbps. The white arrow represents the Smartphone's trajectory. The wireless communication range is shown by the red segment. So there will be a moment when the Smartphone can reach the Gateway and not the Laptop node and at another point the opposite will happen.

<sup>1</sup> Based on Omega Deliverable D1.1

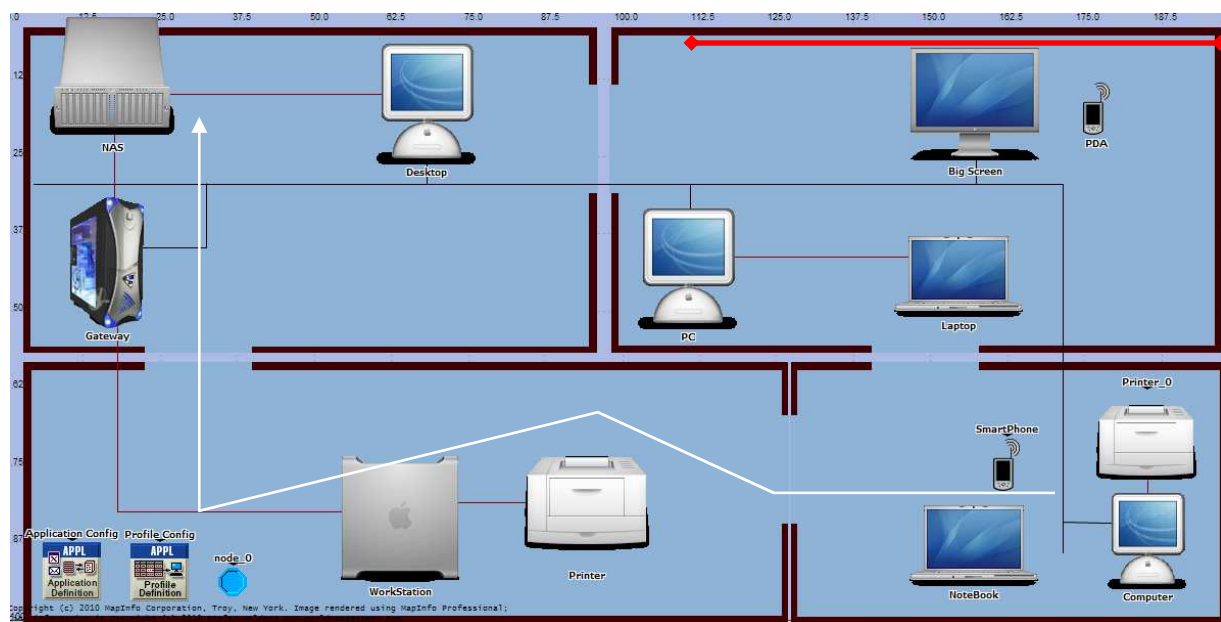


Figure 23: My media follows me

The following table summarizes the network inventory. Each node is represented by a digit which is the number of technologies it has of that specific type. For example, the Gateway node has one interface of every type and two Gigabit Ethernet interfaces. It can be reached by almost all nodes on the network.

| NODE        | Wi-Fi | PLC | Ethernet | NODE       | Wi-Fi | PLC | Ethernet |
|-------------|-------|-----|----------|------------|-------|-----|----------|
| Gateway     | 1     | 1   | 2        | Big Screen | -     | 1   | -        |
| NAS         | -     | 1   | 2        | PDA        | 1     | -   | -        |
| Desktop     | -     | 1   | 1        | Smartphone | 1     | -   | -        |
| WorkStation | -     | -   | 2        | Notebook   | 1     | -   | 1        |
| Printer     | -     | -   | 1        | Computer   | -     | 1   | 1        |
| Pc          | -     | 1   | 1        | Printer_0  | -     | -   | 1        |
| Laptop      | 1     | -   | 1        |            |       |     |          |

Table 2: Node Technologies in the Omega Network

So the objective of this scenario is to show that the Omega network is able to choose at any time the best path for each flow guaranteeing the quality of service. The simulation runs for 180 seconds and it contains four flows. The following table gives the details of every flow.

| Flow # | Source      | Destination | Init time          | Min Band   | Max Delay | Type            | Bidirectional |
|--------|-------------|-------------|--------------------|------------|-----------|-----------------|---------------|
| Flow 1 | Smartphone  | Gateway     | 7 <sup>th</sup> s  | 64 Kbit/s  | 100 ms    | IP telephony    | YES           |
| Flow 2 | PDA         | Gateway     | 12 <sup>th</sup> s | 3.6 Mbit/s | 200 ms    | Video Call      | YES           |
| Flow 3 | WorkStation | Desktop     | 30 <sup>th</sup> s | 50 Mbit/s  | -         | File Download   | NO            |
| Flow 4 | NAS         | Big Screen  | 40 <sup>th</sup> s | 20 Mbit/s  | 500 ms    | Video Streaming | NO            |

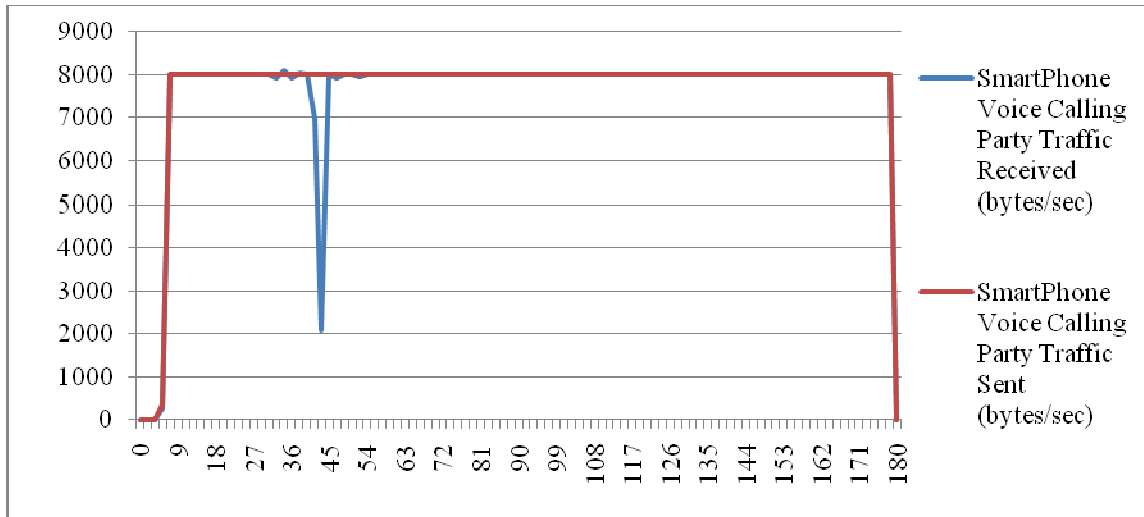
Table 3: Flows characteristics

As it can be seen clearly from Figure 23 the first flow at the beginning will pass from the Smartphone to the Laptop node going over the wireless interface. From the Laptop node utilizing the PLC the flow reaches the Gateway. About time 40 the distance from the Laptop node is great enough such that the delay is increased and it would be a matter of time that the Laptop node would not be reachable anymore. So when the Monitoring Engine realizes that the Laptop node is not reachable anymore it invokes the Path Selection Engine to begin a new path discovery. Even the second flow follows the same path to the Gateway but with only one difference the PDA doesn't move over time. The next flow is the FTP session (file downloading) that has as server the Desktop node and the client FTP is the Workstation node. The Workstation node requests every second, files of six mega

bytes of size from the server node. This flow requires a minimum bandwidth of 50 Mbps in order to not increment disproportionately the response time. The fourth flow is a video streaming that begins from the NAS node and passes over the Desktop node utilizing the Ethernet cable and then goes through the PLC to the Big Screen node. At the 80<sup>th</sup> second another event happens. It is scheduled a link failure for the Ethernet link that connects the NAS node with the Desktop node. So the third and the fourth flow must be re-routed to another path (if available). In fact it is re-routed to the Gateway node and from it to the Big Screen node.

**Wireless link handover**

The first thing to be measured is the impact of the re-routing of the flow from one path to another. There are two cases, one which the link changed is a wireless one and the second is wired. Naturally detecting a link in-availability in the wireless case is much more difficult than the other type. In this section we will give an overview of the wireless case. Figure 24 shows the data traffic sent and received at the application layer from the Smartphone node.



**Figure 24: Data traffic received/sent**

From the figure we can note that the amount of data received has a minimum near time 43. It does not reach zero, but it is not correct, because Opnet has made an average of the amount of data traffic received. In order to give a precise answer about the unavailable time it was registered at the IP layer all the instants of the incoming data packets. From the statistics it can be seen that the gap of time is nearly 1.5 seconds. Reading the outputs of the simulation console (substantially some *printfs*) in this time the contribution of the detection time is 0.7 seconds and the rest of time comes from the new path discovery. This is a relatively bad case because the wireless interface is used even by the other flow so it can also consider congested. The biggest problem lies on the transmission queues at MAC layer which are almost full of packets. So the control packets are sent with a major delay.

The next figures display the end-to-end delay and the packets drop because of the exceeding of the threshold of attempts for every packet. The end-to-end delay is about 3 ms in the old path because it passes over another node and after the re-route it is smaller than 1 ms. During the period that the terminal is at the edge of the distance of coverage of its next routing node the bit error rate is almost  $10^{-3}$  and then after the re-route it goes again to its normal level. These facts are shown in the following graphics.

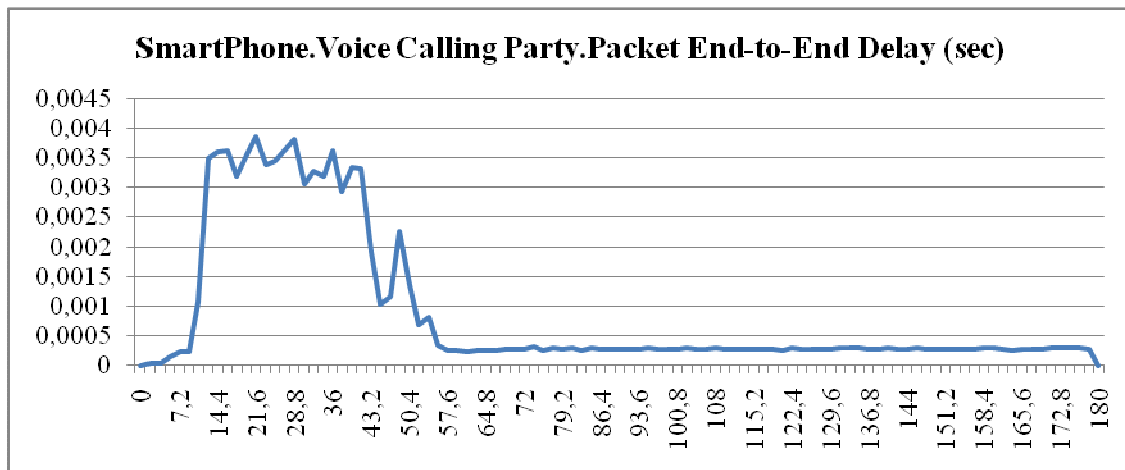


Figure 25: Smartphone end-to-end delay

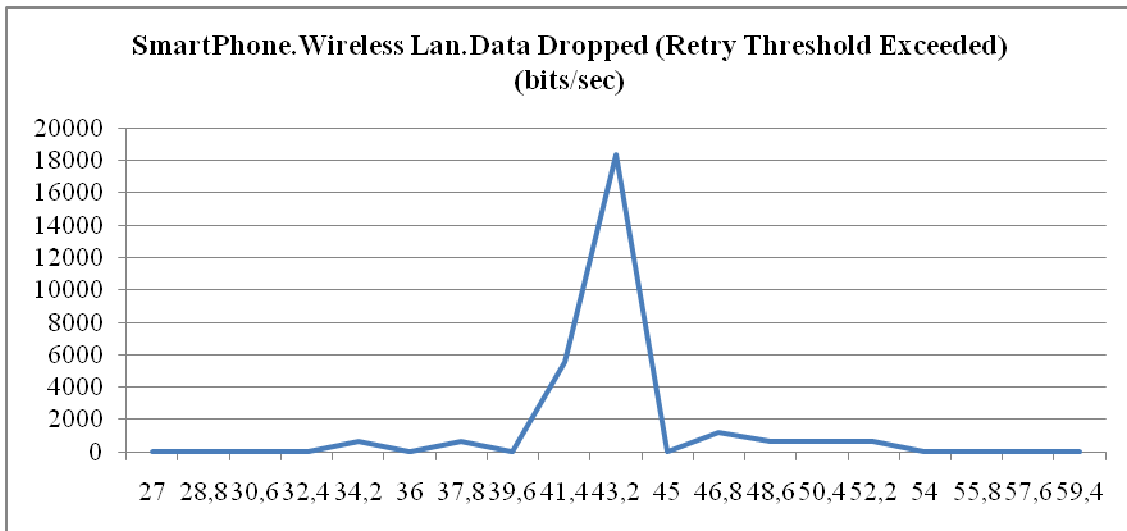


Figure 26: Data packet dropped

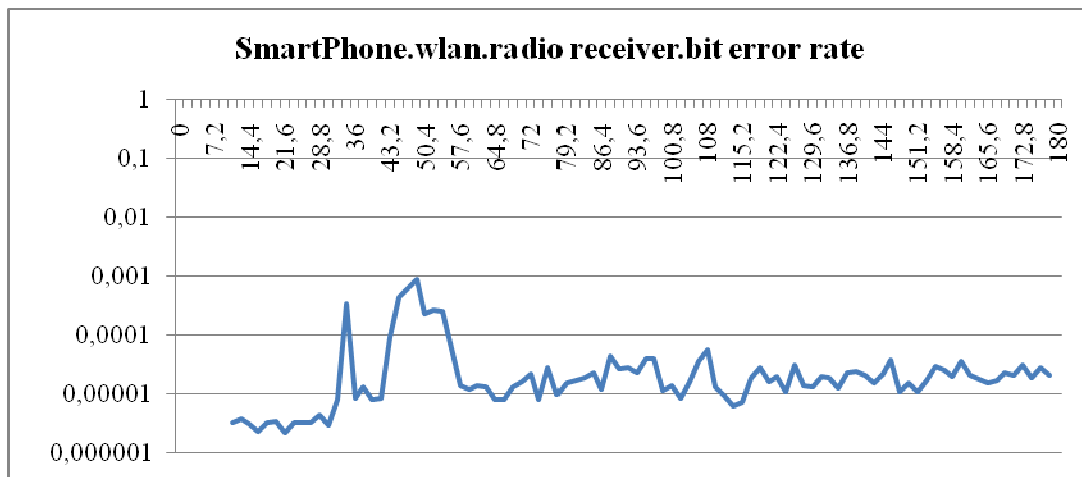


Figure 27: Smartphone bit error rate

The next task was to assure that the quality of the other flow that passes over the wireless link was not affected. The next figures show this. There is only a slight reduction in the amount of data received because of the Laptop node was tempting to send data to the more distant Smartphone node. The same phenomenon happens even to the end-to-end delay statistic, i.e. it increases a little around time 40. The interval of time on which the old path was not available is shown by two vertical lines on the graphics.

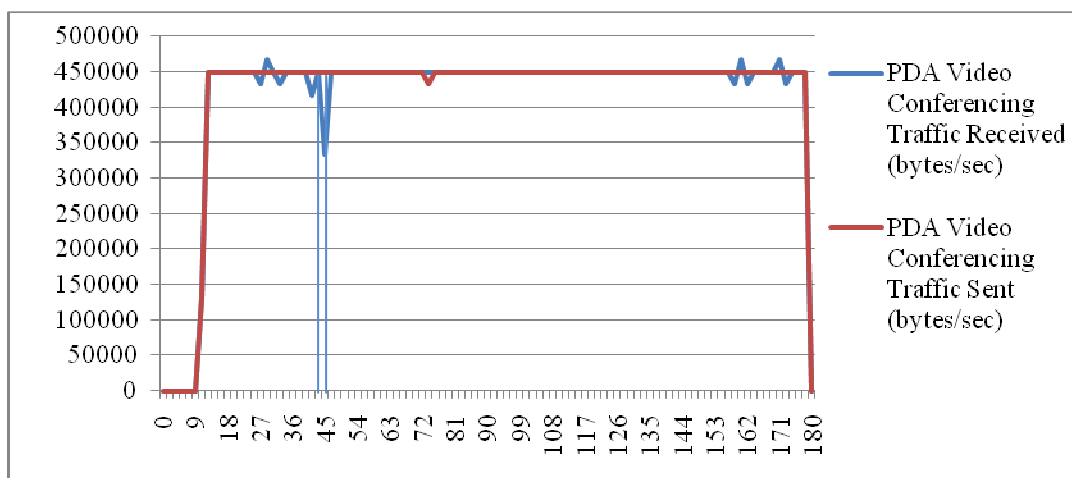
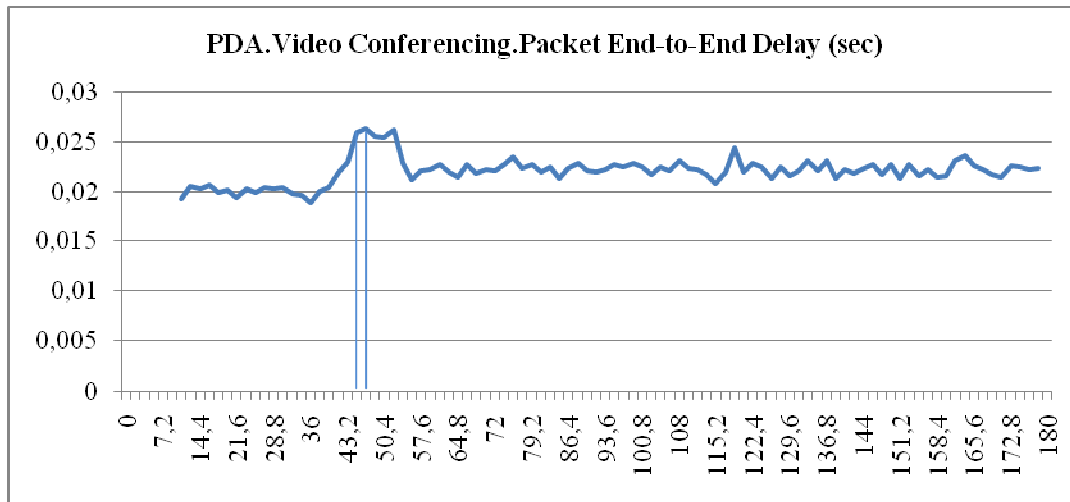


Figure 28: PDA data traffic received/sent

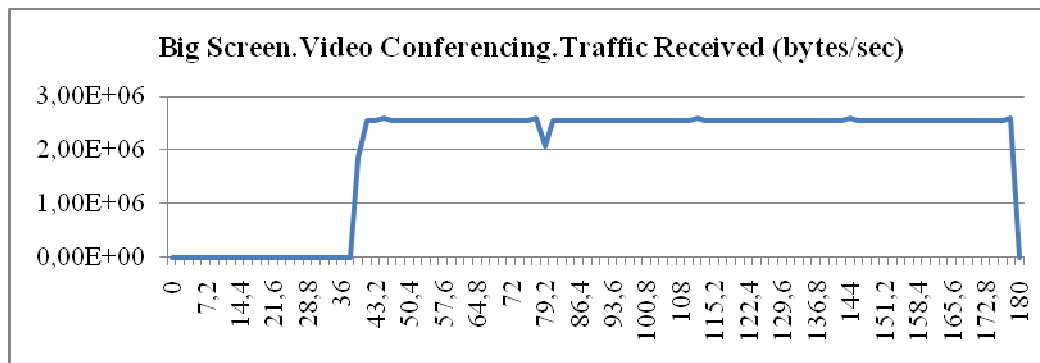


**Figure 29: Packet End-To-End delay**

The end-to-end delay of data packets at the application layer remain at 20 ms reaching a maximum of 26ms at time 43. The delay is a little bigger after the 40<sup>th</sup> second because in the PLC bus is running an additional flow.

### Link failure

We now take a look at the other re-routed flows. This case was an hard failure of a link at a certain time. Given the fact that it is a wired link the detection and the subsequent re-routings happened much faster. In fact again measuring all the instants when an IP packet was received at the IP layer at the Big Screen node the gap of the re-routing time is about 380 ms. At this time the detection contributes of 250 ms and the rest is the time of the path discovery. Figure 30 shows the amount of data received by the Big Screen node.



**Figure 30: Big Screen traffic received**

As it was mentioned before the interruption of the flow isn't visible in this graph because of the data merging and averaging that Opnet does in the statistic collection. Opnet collects the statistics with the bucket mode i.e. given the time of the simulation, the number of samples produced from the statistic, and the number of output values of the statistic it collects an equal amount of samples in each bucket and divides the sum with the interval of time. The number of output values or buckets is constant. This statistics regards the UDP streaming flow.

The other flow under exam is the FTP session application. Even this other flow was redirected because it was going over the failed link. In fact the path of this flow was composed by three links beginning from the Desktop node the flow went through the NAS node and then to the Gateway node and in the end to the WorkStation, going only over Ethernet links. After the failure it is passed through a path composed of two links, the first from the Desktop Node to the Gateway using the PLC bus and the second using the same Ethernet wire of the precedent path. For this flow it was measured the download response time i.e. the time necessary in order to download file. The file size is 6 Mbytes and in the beginning the response time is about six seconds for each file producing a 50 Mbit flow. During the re-routing the response time of some files increases at maximum of 8.5 seconds and then remains at a constant level of 6.5 seconds. The greater response time after the re-route is due to the second path which has less capacity. In fact, it goes through the PLC link that is utilized at 50% of its capacity.

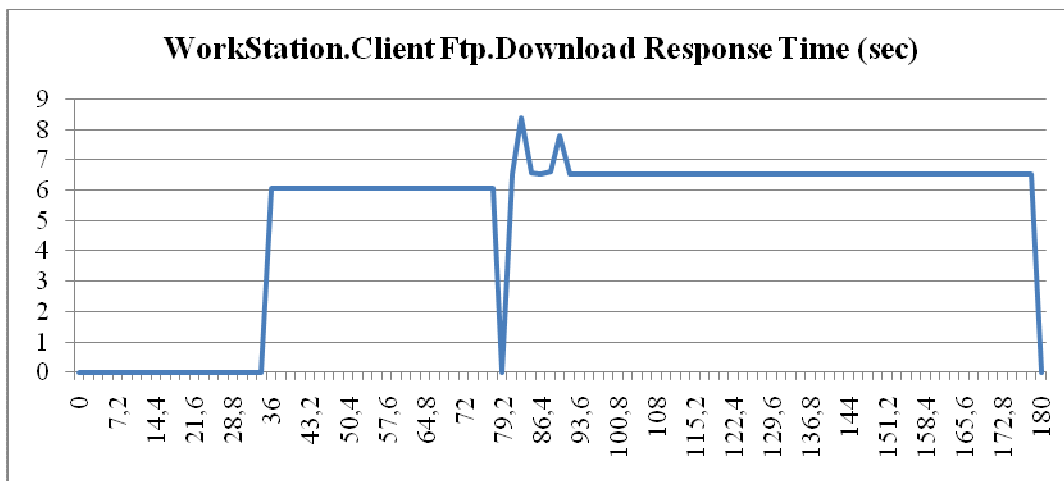


Figure 31: Download response time

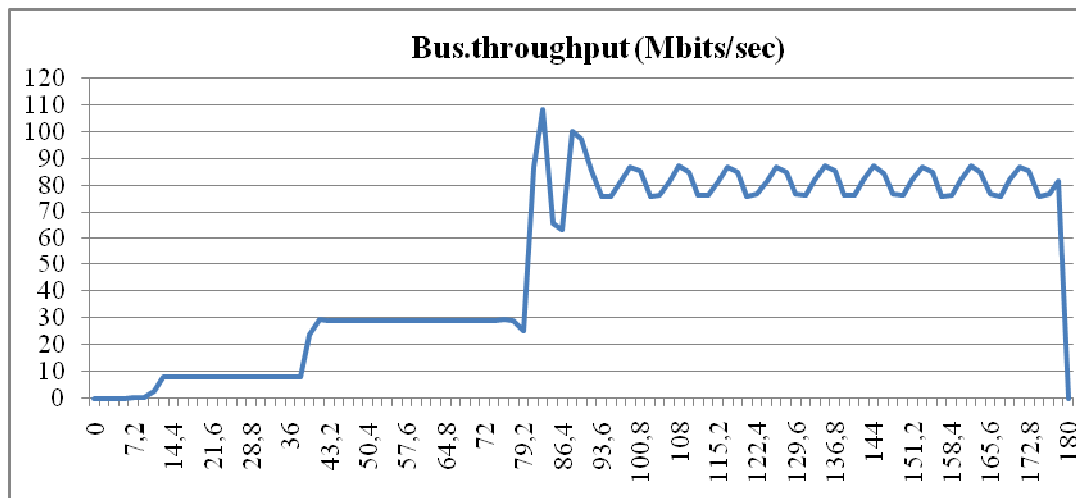


Figure 32: Bus PLC throughput

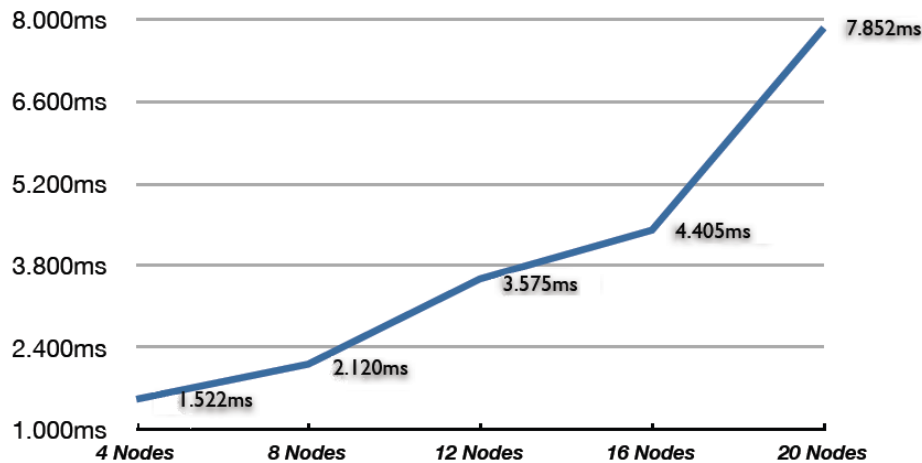
## 1.3 Comparison between reactive and hybrid mode

### 1.3.1 Path setup and convergence time test

The main purpose of this test was to measure convergence times of both Proactive and Reactive modes in HWMP. In order to compare the results, the tests were set up using 5 different scenarios which have 4, 8, 12, 16 and 20 nodes. For each scenario, two separate tests were made: one with the Root node disabled (that is, using just the Reactive mode of HWMP) and one with the Root node enabled (that is, using both Proactive and Reactive modes of HWMP). Before going into the results, it is important to notice that the times to be presented represent two different situations. In the first case, when using just the Reactive mode in HWMP, the time shown represents the time required by the protocol in order to find a path and notify the source node to start transmission of data; that is, it represents the *Path Set-Up Time*. In the second case, the time depicted represents the time required by the Proactive mode in HWMP to find a path to all nodes involved in the topology; that is, the *Proactive Mode Protocol Convergence Time*. In a Real-Life scenario, the Proactive tree would have been already set-up by the time the application starts, so the initial delay for the application is practically zero (taking only into consideration the delay caused by path set-up). The simulations considered one or two background flows that started around 5 seconds after simulation start. The main flow to which it was measured the Path Set-Up Time started 7 seconds after simulation start. The background flows were created in order to use around 60% of channel capacity of a common channel between background flows and main flow. The background flows were FTP traffic, requiring the transfer of files every 10ms, with a file size dependent on the channel capacity. The main flow was a Video Conference requiring 10 Mbps of band.

#### 1.3.1.1 Reactive Mode Path Set-Up Time

The following figure shows the results obtained when measuring the time required to establish a path using the Reactive mode in HWMP.



**Figure 33: Path Set-Up Time using the Reactive mode of HWMP.**

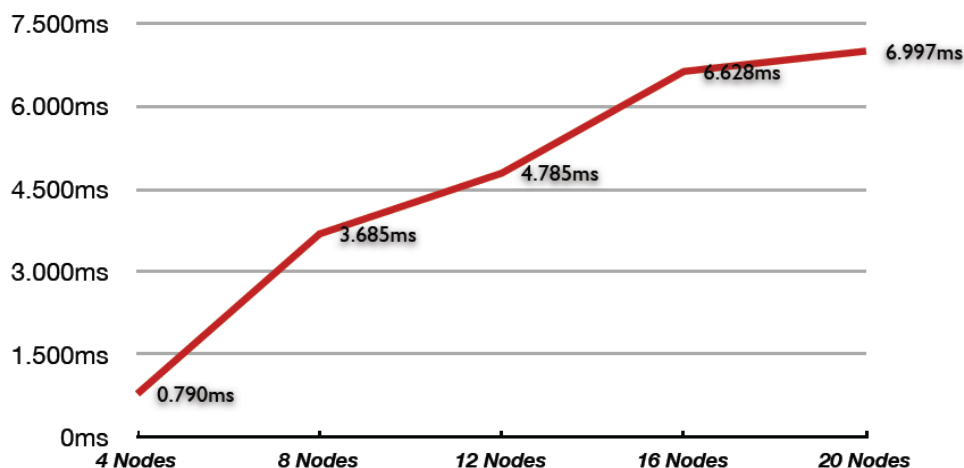
The figure above shows that when using the Reactive mode of HWMP only, the initial delay to start the application does not exceed ten milliseconds in the scenarios tested. This time is a function of the number of nodes, the actual traffic of the network, the number of hops from source to destination and the type of the links. The order of delay for the application to start is acceptable and imperceptible.

### 1.3.1.2 Proactive Mode Protocol Convergence Time

After simulating the convergence time of the Proactive mode of HWMP, it was found that the necessary time to establish a path to every node grew in a linear manner, starting from 0.790ms in the scenario with 4 Nodes and ending at 6.997ms in the scenario with 20 Nodes. There are two facts that must be taken into account when analyzing the result:

- The lower time required to find paths to every node in the network is due to the fact that in the Proactive mode in HWMP there is no Wait Window at the destination, meaning that when a PREQ arrives the node forwards it and immediately replies with a PREP packet back to the Root node.
- Since the Proactive mode starts right after simulation start time, there is no background flow set in the network yet, reducing the link delays and therefore convergence time.

Figure 34 shows the results obtained when measuring the time required by the Proactive mode in HWMP to converge.

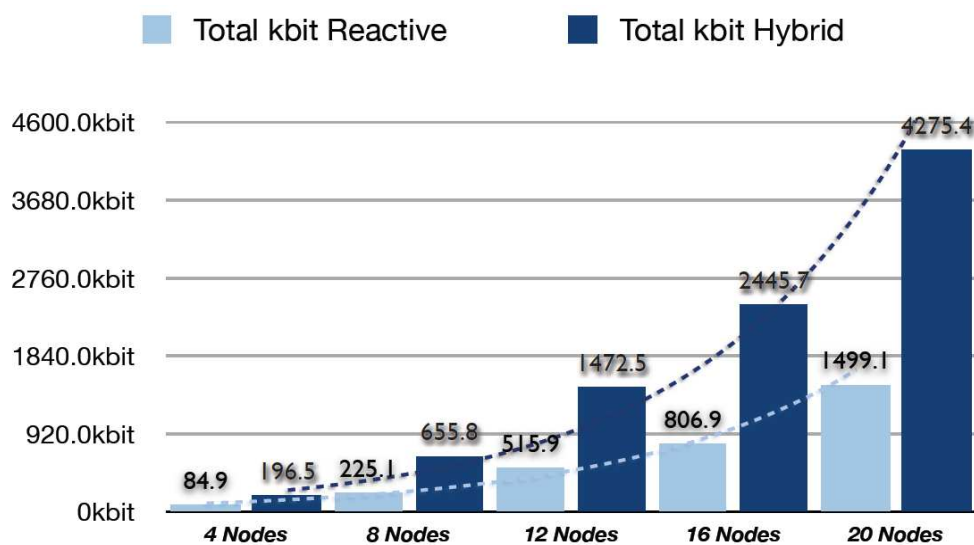


**Figure 34: Convergence Time using the Proactive mode of HWMP**

## 1.3.2 Protocol Overhead of HWMP

The main purpose of this test is to measure the total amount of control bits generated using HWMP as the path selection protocol. As done in the previous test, the used scenarios were those of the previous section, which consist of five scenarios with 4, 8, 12, 16 and 20 nodes. The test was separated into two main parts: the first only considers the Reactive mode in HWMP, while the second considers both Reactive and Proactive modes in

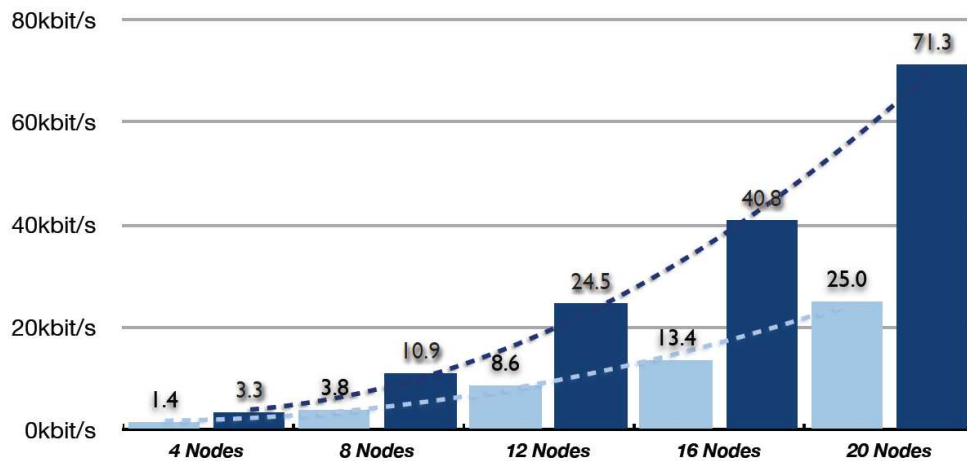
HWMP. The network was set-up to have no background flows, and to have the main flow (Video Conferencing at 10 Mbps) starting seven seconds after simulation start time. The simulation was set to run for 60 seconds. Given the fact that the default path maintenance time is two seconds, the number of reactive attempts was 27, while the total number of hybrid attempts was 57 (30 for the proactive mode and 27 for the reactive mode). The results that are shown are only a representation of the performance of HWMP when used as the Path Selection protocol for the Omega network. These results can vary significantly depending on the network topology and since each node utilized in the simulation can have up to three technologies and therefore links. The first result presented in Figure 35 is the *Control Overhead*. Here we can see the total amount of control bits received because of PREQs and PREPs packets whether the Root node was enabled (Hybrid mode) or not (Reactive mode only). It can be observed from the graphic that the values follow a non linear trend, incrementing the amount of control bits when the number of nodes in the network is incremented. This result can be explained as follows: increasing the number of nodes  $N$  the number of connections between nodes is given by  $N(N-1)/2$  in a full mesh topology. This approximates very well the trend of the graphics taking into account that it is not a quite full mesh topology. The data shown here are the control traffic received by the nodes and thus expresses the total control traffic processed by the nodes. If we had to consider the utilization of all the links of the network the situation would change. Broadcasting a control message to all nodes in the network means that each node makes a broadcast to each of its interfaces but only once. Thus the complexity of this operation is given by  $O(N * \text{Avg}(I))$ . Where  $\text{Avg}(I)$  is the average number of interfaces of the nodes and it's a constant value. So the utilization of the overall network capacity increases in a linear manner with  $N$ .



**Figure 35: Control overhead in kbit for the Reactive and Hybrid modes of HWMP**

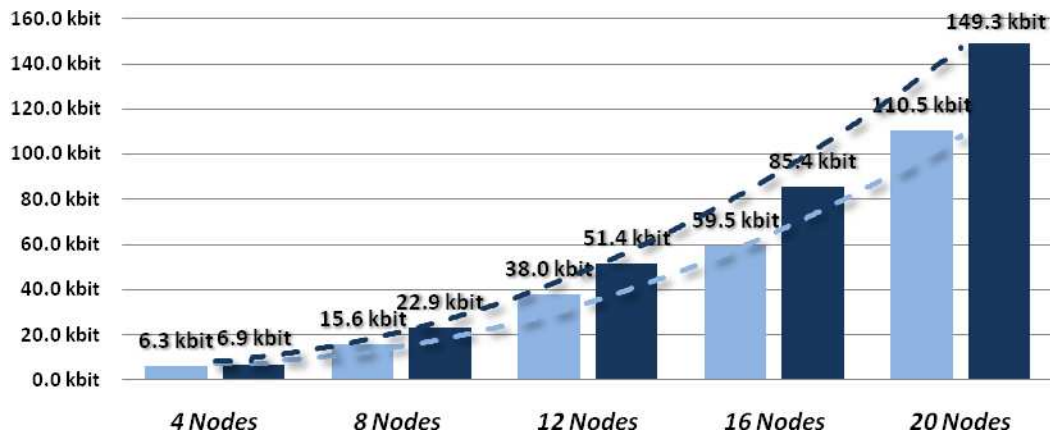
In Figure 36 the overall link utilization in kbps for each simulated scenario is shown. The overall overhead generated by a single flow requiring to set-up an optimum path is shown in light blue. Being a reactive path request, the link load generated is carried in its majority by the links involved in the path that is being set-up. Instead, shown in dark blue, is the overall link load generated when using both proactive and reactive modes of HWMP which contains the additional load caused by the proactive mode. Because even the proactive mode's packets are sent in broadcast, the link load caused is "spread" throughout the network. The results shown put in evidence two important facts:

- The average overhead generated by HWMP (Reactive or Hybrid) increments as the number of nodes in the network increments. Moreover, the increment is more pronounced when considering the Hybrid mode since more PREQs and PREPs are sent throughout the network in broadcast;
- Compared to the Omega network requirements, the obtained values are rather small and should not significantly influence the network performance.



**Figure 36: Control overhead in kbit/s for the Reactive and Hybrid modes of HWMP**

The next result expresses the number of kbit of control overhead received for each time a path needs to be found. One attempt is defined as the initiation of a request to find a path in the network, this can happen either because a new flow needs to be established or because the flow has already been established and the path has to be maintained. Moreover, the initial setup and subsequent path maintenance of the Proactive tree in HWMP is also considered as an attempt. As said before, the total attempts were 57 (30 for the proactive mode and 27 for the reactive mode). Figure 36 shows the average control overhead of HWMP per attempt.



**Figure 37: Average control traffic for attempt**

## 2 Demonstrator results

In this section the lessons learnt by the simulation results have been applied in real environment implementing the data plane and the control plane functionalities. The performance have been measured and compared with the theoretical results obtained during the simulation. While the simulation focused on the whole Omega network, simulating both the data and the control plane, in this section different demonstrator have been implemented to measure particular performances of data and control plane.

### 2.1 Data Plane

The target of setting up a demonstrator in order to evaluate the performance of Inter-MAC data plane was twofold. On the one hand, it was necessary to find a cheaper solution to create Inter-MAC enabled nodes for the v2 demonstrator which is being coordinated by WP7 and which will be shown in the 3<sup>rd</sup> Omega Open Event by March, 2011. So, v1 was split into hardware-dependent and hardware-independent parts making feasible the portability to PC platforms at the expense of performance. On the other hand, the best technological choice (i.e. Data Plane running on Power PC processors on FPGA boards) was optimized targeting the best performance.

In particular, the following demonstrators have been implemented:

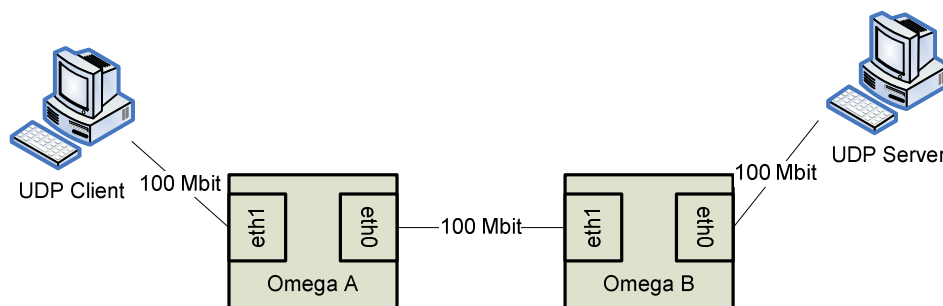
1. Performance of Data Plane on FPGA and PC platforms:
  - a. PowerPC on FPGA.
  - b. Linux PC kernel implementation.
  - c. Linux PC userspace application.
2. Performance of Data Plane optimized for the System-on-Chip on a Virtex-4 FPGA of the RAPTOR-X64 rapid prototyping board.

In the following sections each demonstrator is described and the most relevant results highlighted.

#### 2.1.1 Performance of Data Plane on FPGA and PC platforms

##### 2.1.1.1 Description of demonstrator

In order to evaluate the performance of the data plane, we set up an Omega network depicted in Figure 38. In this scenario we connected two Omega nodes using 100 Mbit/s Ethernet interfaces. Moreover, we connected two legacy devices – UDP server and client – to the Omega network. During the experiment UDP client sent continuously data to UDP server with a predefined throughput, using the iperf application.



**Figure 38: Testbed configuration**

We examined and compared three different software solutions of Omega devices:

1. PowerPC processors on FPGA boards.
2. Linux PC kernel implementation.
3. Linux PC user space application.

In the following paragraphs we present the results of all solutions.

##### 2.1.1.1 Configuration of demonstrator and measurements

The tools used to generate traffic and to measure the round trip delay are: iperf (client/server application) and ping (ICMP frames) running on the legacy devices.

### 2.1.1.2 Results of PowerPC on FPGA demonstration

First of all, it must be noticed that, although FPGA allows a high speed forwarding when using a hardware description language (e.g. VHDL), we used a software solution (ANSI C) only. The theoretical maximum throughput of the interfaces we used in the experiment is 100 Mbit/s. However, application cannot achieve such a high throughput, e.g. due to additional protocol overhead (i.e. interfaces send additional protocol data apart from payload). Nevertheless, we noticed that PowerPC solution achieves a very high throughput, i.e. approx. 90 Mbit/s with less than 1% packet loss rate. The end-to-end delay from UDP client to UDP server was approximately 1-2 ms.

Each time UDP client sent a frame to UDP server, Omega A received the frame, added the Inter-MAC header and forwarded to Omega B. Clearly, adding a new header to each received frame results in additional delays. We measured the time from the frame reception to the frame transmission (adding Inter-MAC header) for each forwarded frame. The average forwarding time was approximately 70  $\mu$ s. In other words, FPGA implementation needs about 70 microseconds to forward a frame to the Omega network. Thus, theoretically FPGA can forward more than 14.000 frames per second. In the best case, i.e. each frame is 1500 bytes long (max. MTU of Ethernet), Omega nodes running on FPGA may achieve a throughput of 170 Mbit/s. It shows that the 100 Mbit/s interfaces is the bottleneck of our scenario.

### 2.1.1.3 Results of Linux kernel demonstration

In the next step we implemented the data plane on a PC running Linux. To achieve a high throughput we run the data plane within the kernel space. In this way, upon receiving a frame, the PC with the data plane did not have to pass it to the user space, which might involve considerable delays. We used the topology depicted in Figure 38.

In general, we achieved the results very similar to the FPGA solution, i.e. throughput of 90 Mbit/s (packet loss less than 1%) with end-to-end delays of 1-2 ms. As we achieved almost the maximum throughput of 100 Mbit/s interfaces, we expect to achieve better results when using Gigabit Ethernet cards.

### 2.1.1.4 Results of Linux user space demonstration

We evaluated the similar scenario as previously, Figure 38, with the data plane running on Linux in user space. Each time PC receives a frame (in kernel space), it must deliver it to the application (user space). As PC performs it for each frame received, it resulted in considerable latencies in our scenario. Thus, Omega nodes achieved 0.2 Mbit/s throughput only. Obviously, such a solution cannot fulfill the requirement of gigabit Omega network. However, the application running in user space has a very important advantage in the development process. It allows testing and debugging in a way much easier than a kernel module.

## 2.1.2 Performance of the Data Plane optimized on RAPTOR-X64

### 2.1.2.1 Description of demonstrator

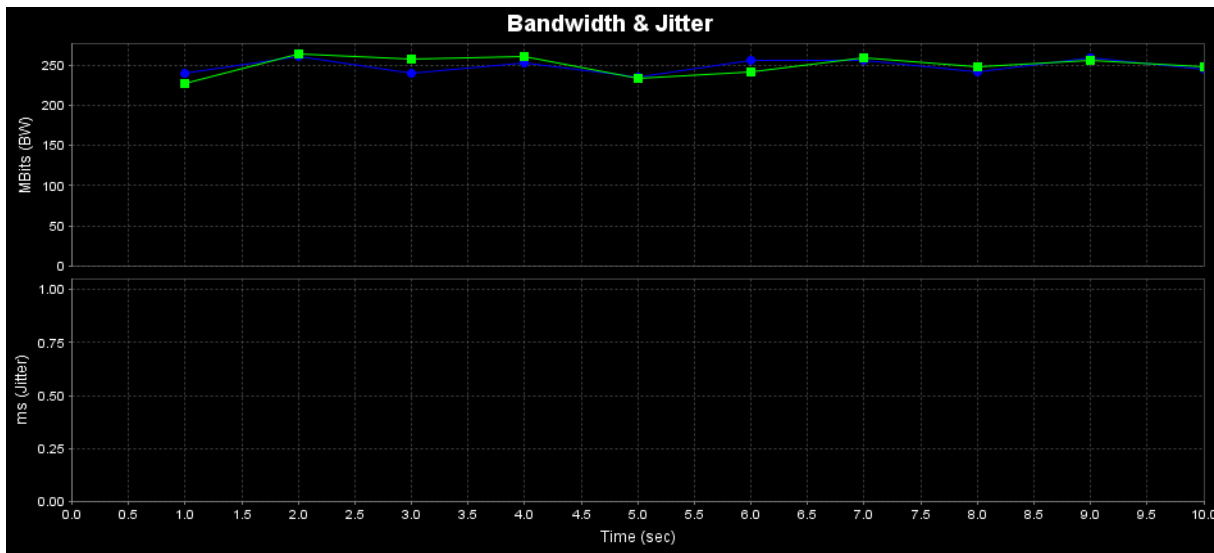
The demonstrator consists of the Data Plane code optimized and running natively on an embedded PowerPC 405 processor in a custom System-on-Chip environment configured on a Xilinx Virtex-4 FX100 FPGA. The FPGA is part of the RAPTOR-X64 rapid prototyping system. It is connected to two Gigabit-Ethernet PHY ports and two Fast-Ethernet PHY ports. The two Gigabit-Ethernet ports were used in the experiments. One was configured as the north interface, while the other one was configured as a south interface of the Data Plane running in Omega Extender mode, meaning that it accepts legacy traffic (non-Inter-MAC).

### 2.1.2.2 Configuration of demonstrator and measurements

The following, are measurements done on an Omega extender with one north interface device and one legacy south interface device. The throughput measurements are done without any software-intrusive profiling code, and without any debug outputs. Measurements were done with Linux or Windows running on the two connected devices, running the open-source tool iperf in server mode or client mode.

### 2.1.2.3 Results

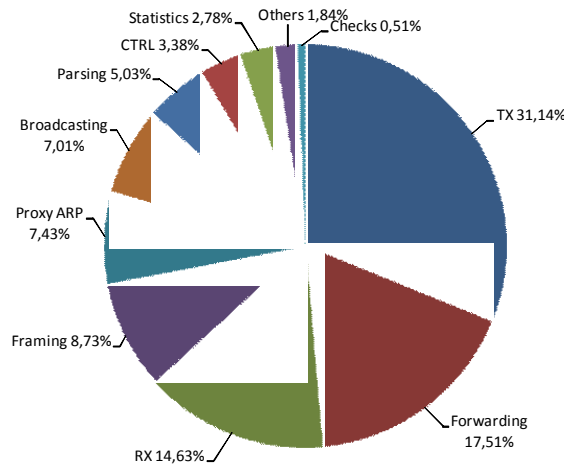
On average the processing of each frame requires 10.556 clock cycles in terms of the PowerPC processor clock, running at 300 MHz, which would result in approximately 340 Mbps Ethernet payload throughput, if the frames used for measurement are 1500 Bytes large. Due to additional necessary management and maintenance tasks the throughput is only around 250 Mbps (cf. Figure 39). The round-trip latency is usually around 125  $\mu$ s, with a jitter of 50  $\mu$ s.



**Figure 39: TCP troughput measurement with iperf (output of jperf)**

The throughput is limited by frame rate, not bit rate. The frame rate is limited by the available processing capacity and the number of cycles per frame. In the following the number of cycles spent for each task is analyzed.

The time spent for the processing of each frame is dominated by tasks related to frame transmission (TX), frame reception (RX), and frame forwarding decisions. TX and RX are tasks involving many accesses to slow off-chip memory and hardware control registers. Frame Forwarding is involved with many table searches (Forwarding Table, Legacy Device Attachment Table, Forwarding Exception Message Status Table ...), and data structure processing.



**Figure 40: Profiling output for the Data Plane (RAPTOR-X64 optimized version)**

Minor, but notable parts are either involving expensive off-chip memory accesses (Framing, Proxy ARP, Parsing) or large control flow structures (Broadcasting, Parsing). Access to on-chip memory is less costly, so Control Plane related tasks, statistics and frame checks are quite fast.

The measurements show that the most improvement can be achieved by moving the RX and TX related tasks to hardware, by implementing them as finite state machines. A table search engine, implemented as a hardware accelerator may improve the duration of the Forwarding task. These optimizations are expected to at least double the achievable frame rate, thereby doubling the achieved throughput to around 500 Mbps. Even further improvements are unlikely to be achieved with the existing solution. Partitioning the program to several processors (organized in a pipeline) or running the same program on several processors in parallel (pool of processors approach) would be the most convenient way of further improving the throughput to 1 Gbps or more.

## 2.2 Control Plane

In the context of an Omega network the most time-critical action to be performed in the control plane is the path selection. In chapter 1 several solutions have been evaluated both reactive and proactive.

An important requirement on an Omega network is the fast reaction to link breaks, i.e., a short convergence time. As already shown in section 1, a simple setup of a wireless mesh network is used to investigate the general timing behavior of the Hybrid Wireless Mesh Protocol (HWMP) by simulation. In this section, a simple setup of a wireless mesh network with four nodes is used to investigate the general timing behaviour of the reactive mode of the Hybrid Wireless Mesh Protocol (HWMP). In the Omega network, an extended specification of HWMP will be used.

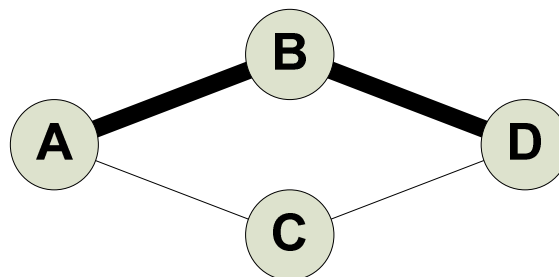
### 2.2.1 Protocol Procedures

The Hybrid Wireless Mesh Protocol (HWMP) is a hybrid path selection protocol for wireless mesh networks. It is currently standardized in the IEEE 802.11s task group [802.11s d3.05]. The basis of HWMP are reactive path selection mechanisms that set up a path in an on-demand manner. If a path to a target node / destination node is needed the originator node / source node starts a path discovery by broadcasting a Path Request (PREQ) message into the mesh network. The target node responds with a Path Reply (PREP) message, which is sent to the originator node by unicast. The PREQ and the PREP set up the reverse path (from target to originator) and forward path (from originator to target) respectively.

If a node detects that the link to the next hop of a path is unavailable, the node issues a Path Error (PERR) message. The PERR contains all destinations that have become unreachable by this so-called link break. The PERR is sent to all precursors on the paths to these destinations and reaches the source nodes eventually. A source node that receives a PERR will start a new path discovery to the target node / destination node that has become unreachable by the link break.

### 2.2.2 Measurement Setup

As an experimental setup, four mesh nodes (A, B, C and D) were used. Node A sends a traffic stream to node D, using either node B or node C as an intermediate node (see Figure 41).

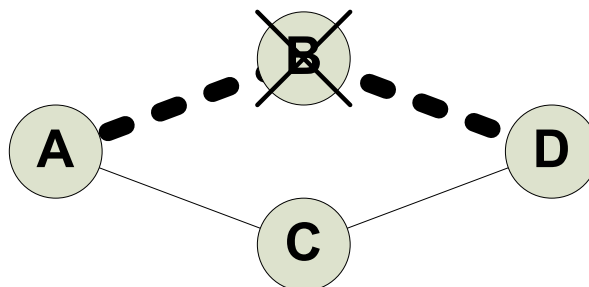


**Figure 41: Experimental wireless mesh network for measurement of HWMP convergence time**

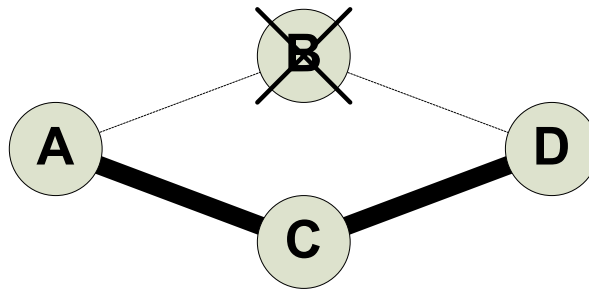
The testbed consists of four wireless mesh nodes (Netgear WNDR3700 WLAN routers), operating with the IEEE 802.11g data rates. Open11s [Open11s] an HWMP implementation very close to the IEEE 802.11s standard, has been used for the path selection protocol implementation.

The traffic consists of a UDP data stream with a data rate of 1 Mbit/s with a duration of 10 s. It is generated using the program iperf as server on node D and as client on node A.

After unplugging node B a link break is provoked, as shown in Figure 42. Node A will then detect the link break. A PERR message is only sent virtually. Node A will start a new path discovery to node D which will result in path A-C-D (see Figure 43).



**Figure 42: Failure of node B causing link break on path A-B-D**



**Figure 43: New path between nodes A and D through node C**

The convergence time is measured between the last successful transmission of B to D until the first usage of node C for frame forwarding.

### 2.2.3 Measurement Results

An average time gap of 674 ms passes until the traffic stream switches from an unavailable active link to the other intermediate node. This convergence time can be divided into three distinct parts (see also Figure 44):

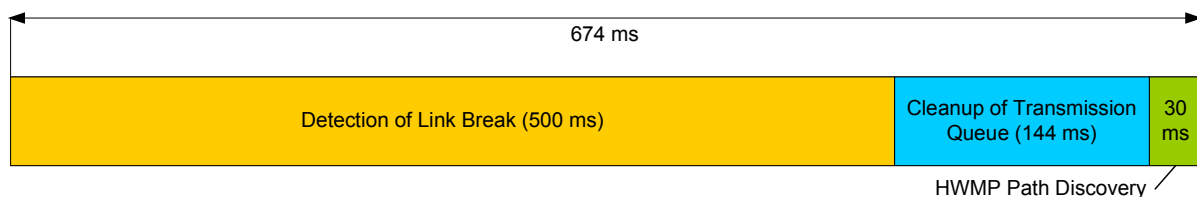
- Detection of link break;
- Cleanup of transmission queue;
- Path discovery.

The detection of a link break is independent of the used path selection protocol. Different strategies are possible in order to determine a link as unavailable. In these experiments, a link has been considered unavailable after three consecutive unsuccessful transmission attempts.

The cleanup of the transmission queue is independent of the used path selection protocol. It depends on the implemented hardware architecture of the communication interface and its processing power. Many hardware architectures have separated modules for the processing of the (upper parts) of the MAC protocol and for transmitting the actual frames. The MAC processing contains the processing of the upper MAC procedures including frame compilation, path discovery and PERR processing. The compiled frames are inserted in the transmission queue. However, the actual transmission of a frame depends on the state of the wireless transmission medium. All frames with the unavailable node B as receiver have to be removed from the transmission queue first, before the new path can take effect. In our measurements, the frames are removed from the transmission queue, the receiver address has been corrected according to the new path to node D and the frames have been reinserted into the transmission queue.

The path discovery is the only part that depends on the used path selection protocol.

Figure 44 shows the time-sharing of the three different parts of the convergence time of HWMP in our measurements. The actual convergence time of the path selection protocol HWMP, which is the time for calculating a new path, is only a minor part of the whole process of finding an alternative path. The new path discovery needs only 30 ms whereas most of the time is needed by processes out of reach of the path selection protocol.



**Figure 44: Convergence Time of HWMP after link break**

Further performance data of the measurements are:

|                         |        |
|-------------------------|--------|
| throughput [kbit/s]:    | 1047   |
| throughput [packets/s]: | 89.4   |
| transmitted packets:    | 894    |
| packet loss ratio:      | 0.48 % |

### 2.2.4 Summary

The actual determination of an alternative path after a link break is actually only a rather small portion of the convergence time. As already shown in section 1.3.1.1 at page 32 most of the time is used for mechanisms outside of the path selection protocol. Consequently, strong care must be taken for the fast and correct detection of link breaks. Furthermore, the fast cleanup of the transmission queue is also important for a minimum convergence time.

## 3 Conclusion

This chapter compiles main conclusions and learnt lessons derived from above reported simulation and demonstration activities grouped into two categories: conclusions on control plane and conclusions on data plane of Inter-MAC technology.

### 3.1 Conclusions on control plane performance

The core of Inter-MAC technology and key for the success of the future home network architecture developed in Omega is the path selection algorithm. Moreover, it is the process which consumes more processing time. That is why this algorithm has been evaluated with simulation and real test-beds. Furthermore, simulations allowed to compare two different approaches: pure reactive mode and hybrid (proactive + reactive) mode.

In the light of the results shown in Chapter 1 and section 2.2, the following conclusions have been achieved:

- Theoretical considerations derived from the simulation results:
  - Given the maximum number of nodes assumed in a home environment, the reactive part of the path selection algorithm can manage easily an Omega Network and any eventual re-routes from failures and congestions.
  - The convergence time of the protocol is less than 1 second. The total re-routing time – adding the detection time – in the worst case is about 1.5 seconds in a path with wireless links.
  - The reactive part can be further modified in order to suit perfectly to the Omega Network. In fact, it is addressed for Wireless Mesh Networks. For example the fundamental metric would be the delay and thus the first Path Requests that arrives at the destination node represents the path with the best metric. So, it is not necessary to spent a big waiting window at the destination node for other Path Requests going on alternative paths.
  - The combination of the two modalities improves the initial delay during the path setup by reducing it from a maximum of 110 ms to practically zero. But from our point of view, it does not justify the increased complexity of the Path Selection Engine and the additional overhead of the protocol. So, the best solution of compromise in an Omega Network is adopting a reactive approach.
- Empirical considerations derived from measurements performed in “physical” test-bed:
  - The experiments carried out in the real test-bed showed that the actual determination of an alternative path after a link break is actually only a rather small portion of the convergence time and most of the time is used for mechanisms outside of the path selection protocol. Consequently, strong care must be taken for the fast and correct detection of link breakages.
  - Furthermore, the fast cleanup of the transmission queue is also important for a minimum convergence time.

### 3.2 Conclusions on data plane performance

Demonstration activities carried out with different versions of data plane (c.f. sections 2.1.1 and 2.1.2) and running on different platforms, entail the following conclusions:

- The maximum throughput achieved with an optimized version of data plane running on RAPTOR board is only around 250 Mbit/s. The round-trip latency is usually around 125  $\mu$ s, with a jitter of 50  $\mu$ s.
- This throughput is limited by frame rate which, at the same time, is limited by the available processing capacity and the number of cycles per frame. Further analysis show that a throughput of 500 Mbit/s could be achieved with the following improvements:
  - Moving RX and TX related tasks to hardware, by implementing them as finite state machines.
  - A table search engine, implemented as a hardware accelerator.
- Furthermore, with a multi-processor approach, a throughput of 1 Gbit/s could be bordered or even exceeded.
- When looking for a cheaper solution for proof of concept or debugging of Inter-MAC technology:

- It has been proven that data plane running within the kernel space is able to deal with almost the maximum throughput of 100 Mbit/s interfaces, being expectable to achieve better results when using Gigabit Ethernet cards.
- Data plane running in user space, as user application, cannot fulfill all the requirements of a gigabit Omega network. However, it is very valuable for testing and debugging.

Summarizing, the simulations helped to identify the feasible concepts, while the demonstrators evidenced that what has been done in a simulated world can be realized in a real prototypal setup. Moreover the demonstrators showed that the prototypal performances are encouraging and further optimization of C-code as well as hardware implementation will lead to even better performances.

## 4 References

- [OMD53] OMEGA Deliverable D5.3 “Inter-MAC protocol entities interfaces specification”, January 2010, available on <http://www.ict-omega.eu/publications/deliverables.html>.
- [OMD52] OMEGA Deliverable D5.2 “OMEGA Inter-MAC architecture design”, June 2009.
- [OMD11] OMEGA Deliverable D1.1 "Final usage scenarios report", September 2008, available on <http://www.ict-omega.eu/publications/deliverables.html>.
- [OPNET] Opnet Modeler, [http://www.opnet.com/solutions/network\\_rd/modeler.html](http://www.opnet.com/solutions/network_rd/modeler.html), last access May 2010.
- [IEEE802.11s] Amendment to standard IEEE 802.11: “Extended Service Set Mesh Networking”, IEEE P802.11s/D3.04 Std., September 2009
- [802.11s d3.05] Amendment to standard IEEE 802.11: “Extended Service Set Mesh Networking”, IEEE P802.11s/D3.05 Std., November 2009
- [RFC3561] IETF RFC 3561 "Ad hoc On-Demand Distance Vector (AODV) Routing", July 2003.
- [Open11s] <http://open80211s.org/>, last access May 2010.