Wireless IPv6 simulator: SimulX

Nicolas Montavont
ENST Bretagne - GET
2, rue de la chataigneraie
35576 Cesson-Sévigné
E-mail: nmontavo@enst-bretagne.fr
Tel: +33 (0) 299 127 023

Julien Montavont
LSIIT (UMR7500) - ULP
bld Sebastien Brant
67400 Illkirch - France
E-mail: montavonj@lsiit.u-strasbg.fr
Tel: +33 (0) 390 244 590

Safaa Hachana
ENST Bretagne - GET
2, rue de la chataigneraie
35576 Cesson-Sévigné
E-mail: shachana@enst-bretagne.fr
Tel: +33 (0) 299 127 035

Keywords: IPv6, wireless and mobile communications, multihoming, cross-layer design

Abstract

This paper highlights the needs for wireless and mobile IPv6 simulation. In terms of programming, modularity, extensibility and re-usability are the main features that we are looking for. On the technical side, mobility, multihoming and cross-layer optimization are the goals we have identified. Based on these recommendations, we show that the existing simulators are not satisfying our goals, and thus we propose a new simulator namely *SimulX*. SimulX is a very promising simulator, which already provides the main wireless technologies and multihoming features in a simple programming environment.

1 INTRODUCTION

The IP technology, and more generally the TCP/IP concept is gaining more and more popularity. While in its initial design TCP/IP was made for peer-to-peer long distance communication, it is now a generic transport for a huge variety of applications, such as video on demand, web browsing, sensors or home networking. Its applicability goes beyond any expectation, and the TCP/IP model is adopted in many domains (e.g., the IMS framework in cellular network). The success of the TCP/IP model is probably its concept of abstraction where each layer provides a particular service which is implementation free to another layer. This independence between layers allows specific technology switching while keeping the remaining stack unchanged. This has favoured the evolution of link layer technologies, especially wireless technologies.

A new wireless world is emerging. Significant progresses have been made in wireless technologies, in terms of bandwidth and range, which enable to get rid of cables. However, many issues remain unresolved and need more research. Firstly, the wireless nature of communication medium allows user to be mobile while being connected to the network. Node mobility is a hard issue because it goes against the initial concept of IP networks; the IP network was built upon the idea that a single entity can be located in the topology and identified through a single address. Now because nodes are mobile, a specific support is required to identify a node on one hand, and on the other hand to locate this node at any moment [6]. Many protocols have been designed for such support [10], [9], but this is still an important area of research.

Secondly, not a single wireless technology enables wide area coverage at high bandwidth. Instead of converging to a single wireless technology, ubiquitous access is granted through the usage of multiple heterogeneous wireless technologies. Each technology provides attracting features, but significant drawbacks make necessary the diversity of access technologies on nodes [13]. Therefore, it is envisioned that mobile devices will integrate multiple heterogeneous interfaces, and will thus need to manage such multihoming configuration in order to take full advantage of multihoming (e.g., redundancy, backup, aggregated bandwidth). The new version of IP (version 6) [5] includes basic support for multihoming, but a full and optimized support is still an important research area and require further analysis.

In this context of the next generation Internet, analysis tools are mandatory. Any new protocol or application needs to be fully tested before its deployment in the Internet. Such research can benefit from simulation tools. However, the existing simulation environments fail to provide sufficient support for multihoming and mobile heterogeneous communication. We describe in this paper the required features that a simulation framework should provide today and how current tools do not meet these requirements. We thus propose a new simulation framework called *simulx* which responds to these goals and provides the necessary tools to simulate heterogeneous networks. The key features of SimulX are modularity, ease of use and implementation, graphical tools and multihoming support in its core.

The rest of the paper is organized as follows. In the next section we first describe the goals that a network simulator must show today. We choose to emphasis first the requirements for the simulation of heterogeneous networks without focusing on any existing simulation environment because we do not want to influence our analysis. These requirements consist of an independent research on its own. Then, in the next section, we analyze the two major network simulators that are available (i.e., NS and OMNeT++) and show that they are not providing the required features. Consequently, we de-

 ISBN 1-56555-312-8

tail in the next section the design of SimulX and highlight its main features. In the following section, we show an analysis example on how useful can be this new simulation framework and we finally conclude the paper by some additional remarks.

2 NEW SIMULATOR MOTIVATIONS

An accurate simulation environment is as much important as protocol standardization. Simulations represent an unavoidable step in a protocol design. Before the wide deployment of a new protocol, validation and performance evaluation is needed. Where some aspects of Internet protocols can be demonstrated by mathematical materials, or small-scaled experimentations, the robustness, scalability and side-effects may be hard to measure in a testbed, if not impossible. A simulation plateform can then be the best tool that fits the need for extensive testing of newly specified protocols. Of course, having accurate simulation environment and results do not prohibit further experimentation testing to validate and improve the given results once simulations are done.

This section discusses the requirements for an accurate simulation environment. We are neither focusing on a particular protocol nor a particular simulator, but we specify the required features that a simulation framework should bring to ease the development and testing of Internet protocols. Requirements are separated into three categories, i.e., requirements for programming, required features for simulating next generation network and required technologies.

2.1 Requirements for programming

A simulation framework should the following features:

Research and educational purposes: it is a common understanding that a simulation framework is a tool for research purposes. A simulation environment is needed to validate protocol design and performances, and especially address scalability issues. It allows concentrating on some aspects of the protocols, or to simply validate a protocol in various environments and scenarios. In addition to the research area, it is important to consider the educational aspect of a simulation framework. With an appropriate graphical interface and comments within the code, it can easily show protocols behaviors, complex topologies or corner cases.

Ease of development: the simplicity of using, adding and modifying a protocol implementation in a simulation framework is a crucial criteria of success. The fact that a simulation environment is easy to master will contribute to its adoption in the research and educational community. The simulation core and the surrounding modules must be self-contained and must not restrain additional development. The addition of any mechanism must be easy, and must not require to know all the simulation tool in details, but rather the specific API between key elements for the target development.

Open-source: Opening the project to the research community is an important feature to acquire popularity, but also to enhance the implementation.

Modular simulation: modularity is the key feature for extensibility and re-usability. A well-adapted modularity allows the addition of features easily, and enhances existing implementation without modifying the whole code. Appropriate API is needed between modules that allow transparency of code. A module thus appears as a black box, with a set of tools to access and control it.

Inputs and outputs (i/o): inputs and outputs are the basis of a simulation environment. For most users, i/o will be the only tools they will have to manage. In order to satisfy every single need, these i/o must be heterogeneous and diversified. A graphical interface facilitates simulation configuration and network topologies and helps in identifying configuration paths (re-play action). Scenario scripts are very powerful to generate thousands of runs of the same simulation, or to modify a single parameter from one simulation to another. Results must also be logged into files that are easily readable, and on which scripts can extract useful information for simulation statistics.

2.2 Requirements for next generation networks simulation

A simulation framework must be adapted to the target demonstration, but it must also provide an accurate environment to ensure that outputs of the simulations are valid. Here is the list of features that we believe to be required for the simulation of Internet protocols:

Completeness: in order to provide accurate simulation models, it is important to provide a complete and non-lossy set of protocols that allow a precise representation of the reality. *The degree of precision* needs to be determined according to the context of simulation. The simulation environment must be close enough to real context in such a way that the results will represent well the protocol performance if it was deployed in its context of execution.

Cross-layer optimization: while modularity is a great feature which allows extensibility, cross-layer optimization is needed. Cross-layer optimization is the extended interaction among layers to trigger and optimize protocols. These API must be available in a simulation framework to take advantage of information at different layers, from different modules.

Wireless communication: today's and especially tomorrow's Internet is wireless. Wireless technologies are getting very efficient and thus very popular. The freedom offered by wireless communication is limitless and services over wireless networks are growing very quickly (e.g., voice or video-conference). An accurate Internet model must now offers wireless environment with different degree of granularity.

These degrees would adapt the complexity of wireless models to the target scenario that is evaluated.

Heterogeneous environment: in the recent development within the Internet community, at least one observation is unanimously considered: wireless systems fail to converge to a single technology, and ubiquitous Internet can still be reached through the heterogeneity of networks[13]. Therefore, a simulation framework must provide a heterogeneous environment, where devices are multihomed and equipped with multiple interfaces.

Node mobility: node mobility is an inherent feature of wireless environment. As the communication means become wireless, nodes are free to move while being connected to the Internet. Although at some point the Internet was considered as relatively fixed, now the main hypothesis is that all communication device, i.e. server, end-node or sensor, may potentially be mobile.

2.3 Targeted technologies

While the previous sections give the main features that an accurate simulation framework should show, this section gives the set of protocols that need to be provided within a simulation framework for the Internet protocols. In order to validate a protocol, it is important that the simulator core provides accurate models for basic protocols, in order to rely on a solid ground. This basis of protocols enforces the reliability and the credibility of the simulation tool by providing accepted protocol models. The main technologies that are considered are the followings:

TCP/IP model: a simulation framework dedicated to the Internet protocols must rely on the TCP/IP model. The TCP/IP stack is made of 5 layers that are independent one from each other. A specific API is designed for upper and lower layers, and interactions between layers are well determined. Each layer provides a service to another adjacent layer. This model needs to be respected, and the common protocols used in the Internet must be provided within the simulation framework. For example, IP, UDP, TCP, SIP must be provided by the core simulator, as well as their main mechanisms (e.g., DNS, DHCP, ICMP).

Medium Access Control: in order to accurately represent today's digital communication, several communication technologies are needed. The traditional simple link model has reached its limits, and now specific link layer technologies are needed to simulate heterogeneous networks. Ethernet, Wi-Fi, Wimax, Zigbee, Bluetooth are emerging technologies that are building the future Internet. Dedicated models are needed for these wireless communication systems.

Channels: a main issue in wireless simulation is the physical channel model. While in some sets of simulation runs, a simple physical channel model is sufficient, in most cases, the physical channel must be well designed in the simulation framework to provide correct broadcast systems, accurate range, and interface modeling, even across different technologies (operating in the same band). Thereafter, for more accurate physical simulations, extensive channel modeling is needed with appropriate propagation model and obstacle modeling.

Mobility model: as mentioned in the previous section, node movement is a required feature. However, movement involves mobility management support, handover algorithm, neighbor discovery mechanisms, etc. In order to correctly represent trusted mobility models, mobility pattern and real city map should be provided within the simulation environment. A user could then choose its mobility pattern and management protocols for running specific simulations.

These required features and identified technologies represent the basis of a strong and accurate simulation framework. Would it be for the ease of use, or for research community approval, a simulation tool which success to include these components could really improve research, and provide researchers with accurate and trusted models. The next section presents the most popular open-source simulators for the Internet protocols. It particularly highlights why these simulation frameworks fail to meet the goals presented in this section. These observations will lead us to propose a new simulation framework in section 4.

3 EXISTING SIMULATORS

This section gives a brief overview of available simulation tools by picking out two of the most representative ones, namely NS and OMNeT++. The purpose of this section is double. First, we identify the properties of each of these simulators and whether they are appropriate to study new mobility protocols and wireless networks. Second, we highlight what are the main issues with the existing simulation frameworks especially when it comes to wireless networks simulation, which may require starting over with a new network simulator.

3.1 NS-2

The network simulator NS[3] is an object-oriented discrete event simulator targeted at networking research and available in the public domain. The current version is the number two, but a new project has just started for developping the version three.

3.1.1 NS-2 monitoring support

User can interact with NS-2 through the OTcl (Object Oriented Tool Command Language)[4] interpreter which provides procedures to define arbitrary network topologies. For collecting trace data on a simulation, NS-2 uses both traces and monitors. NS-2 trace files can also be viewed with the

Network Animator (NAM) which is a Tcl/TK based animation tool. However mobile simulation are not supported (node movement).

3.1.2 NS-2 internal architecture and topology modeling

NS-2 code source is a combination of two languages: C++ and OTcl. This is supposed to offer a compromise between performance and ease of use. Being a discrete event simulator, the core of simulation of NS-2 is composed of three main C++ classes, namely the class *Event*, the class *Handler* modeling entities that generate and consume events, and finally the class *Scheduler* which is in charge of scheduling and dispatching events. The simulation is configured, controlled and operated through the use of the interfaces provided by the class *Simulator*.

NS-2 topology model consists of the interconnection of network elements created through the stand alone OTcl classes: *node* and *link*. The *classic node* structure is composed of two important *NsObjects*: the *Classifier* and the *Agent* which enable network and transport protocol simulation. The *classic node* structure does not model low layer protocols but as the need for wireless modelling become strong, NS-2 now proposes a new mobile node structure that represents the OSI model. The class *MobileNode* extends the basic capability of the *Node* class by adding functionalities of a wireless and mobile node such as a link layer (*LL*), and a *MAC* layer implementing the specifications of the IEEE 802.11 standard. Furthermore, a physical network interface is used by the mobile node to access the wireless channel. To interconnect *MobileNodes*, NS-2 defines a new kind of link which is *WirelessChannel*. It is supplied with three radio propagation models: shadowing model, free-space model and Two Ray Ground model.

3.1.3 Available models

At the time being, NS-2 has a large number of protocol models, mostly centered on TCP/IP. It is well-suited for packet switched networks and wireless ad-hoc networks, and is used mostly for small scale simulations of queuing and routing algorithms, transport protocols, congestion control, and some multicast related work.

3.1.4 Main characteristics

From the short description of its architecture given above, it is quite clear that implementing a new protocol in NS-2 is not a straightforward process since it involves adding C++ code for the protocol's functionality, as well as updating key OTcl configuration files. In addition, the learning curve for NS-2 is steep and debugging is difficult due to the dual C++/OTcl nature of the simulator. The provided documentation does not help from this point of view. Indeed, it is often limited and out of date with the current release of the simulator. As a net result, NS-2 is not so easy to use in the perspective of contributing new models, protocols, and studying different scenarios at different levels of detail.

Furthermore, although the project started since 1989, the simulator is not stable. This is due to the incorporation of contributions from different sources in addition to the continuous changes in the trends of network community requirements and the submersion of new technologies. For instance, as we interest particularly in wireless network simulation, and since wireless modules were added lately to NS-2 library, we wondered about the reliability of wireless networks implementation in NS-2. Indeed, we tested some wireless scenarios with NS-2.28 using tutorials given within the package and detected some anomalies:

- Among the three pretended radio propagation models, only the Two Ray Ground model is really functional.
- Nodes are designed to move in a three dimensional topology. However the third dimension is not taken into account in height of antenna calculation. Moreover, whatever the configuration, the height of the antenna is set statically to 1.5 m.
- The most serious problem we found is that wireless network model with infrastructure does not work at all. Indeed, the access point is totally ignored, and considered as an ordinary node. Only ad-hoc mode is supported contrary to the pretensions of the simulator.

In addition to the biased IEEE 802.11 model, node multihoming is not supported. Due to the model of a node (or mobile node), it is not possible to set multiple interfaces on a single node. In our previous studies, we succeded to set multiple interfaces on a single node, but it was by using several nodes within a single node. Moreover, it was not possible to set multiple 802.11 interfaces, because of the channel modeling: a single node could not be operating on two different channels. All these flaws, in addition to others that we did not mention in the present document affect dramatically the credibility of NS-2, and show that it is not recommended for wireless network simulation.

A more troublesome limitation of NS-2 is its large memory footprint and its lack of scalability. In addition, NS-2 is reported to be in general quite slow from a computational point of view. But against all the issues we brought out, NS-2 remains the most used simulator for studies on transport protocols. Furthermore, it has become popular in the research field of mobile ad-hoc networks too.

3.2 OMNeT++

The "Objective Modular Network Testbed in C++" known as OMNeT++[11] is a public domain discrete-event simula-

tion package. It offers a C++ simulation class library and GUI support. It is not specifically designed for telecommunication networks, but for any system which can be mapped to active components that communicate by passing messages.

3.2.1 OMNeT++ monitoring support

An OMNeT++ simulation script consists of a collection of hierarchically imbricate modules, their parameters, their connections, and their behaviors. It is spread over multiple files; First, the NED (NEtwork Description) [12] language is used for topology description within a NED file. Second, *.h and *.cc files are required to define the modules classes. Finally, a default configuration file for the simulation program (*.ini) is required. Obviously, writing a simulation scenario for OMNeT++ is not such a simple task.

User can choose between two user interfaces to run an OMNeT++ program: command-line and graphical interface. Although the command line interface is rather uncomfortable and used only to run the program with maximum speed, the nice graphical interface offers good potentialities for monitoring and analysis of the simulations.

3.2.2 OMNeT++ internal architecture and topology modeling

OMNeT++ is component-based, totally modular and open-architecture tool. A module is a C++ object, having well specified interface and state, and implementing a specific functionality. There are two main types of modules in OMNeT++: simple modules and compound modules. Unlike NS, there are no predefined network devices in OMNeT++. In fact, OMNeT++ models a system (e.g. a network) by imbricating hierarchical modules. This allows the user to reflect the logical structure of the actual system in the model structure. Modules communicate via messages passing either directly to their destination modules or along a predefined path, through gates and connections. Modules at the lowest level of the module hierarchy are to be provided by the user, and they contain the algorithms in the model.

3.2.3 Available models

OMNeT++ is not specifically designed for telecommunication networks and even less for wireless telecommunication networks. However, thanks to the numerous side works on OMNeT++, the simulator has acquired a very large set of network simulation module libraries that belongs to different projects within a relatively short time. For instance, we can find implementations for TCP/IP network models. LAN/MAN protocols such as Ethernet are also available for OMNeT++. In addition, mobility, ad-hoc and wireless protocols are supported in OMNeT++. Nevertheless, attempt to integrate all these separated contributions usually fail because of their incompatibility with each other..

3.2.4 Main characteristics

Being highly modular and well structured is a big advantage for OMNeT++ when it comes to implementing new protocols. Extensibility can even be confused with usability since a user is required to define its own modules and classes in C++ language. Unfortunately, the problem of incompatibility between modules (most of the time because they are developed separately) remain a major issue.

On the other hand, according to the well-planned conception of its kernel of simulation and to the youth and modernity of its architecture, it is reasonable to expect a good scalability of OMNeT++. Besides, for a good CPU resource management, simple modules appear to run in parallel during simulation execution, since they are implemented as co-routines. OMNeT++ also supports parallel simulation [14]. In addition, a consistent documentation is delivered with OMNeT++ package. However the incompatibility between crucial modules and the difficulty to run simulation make the OMNeT++ package no suitable for next generation IP simulation.

4 NEW SIMULATOR DESIGN

According to our analysis, none existing simulator proposes a framework that fullfills the motivations depicted in section 2. Actually, the simulators presented in the previous section are hardly extensible to respond to the needs expressed in section 2. In this section, we present the philosophy and the main components that are building our new network simulator, namely *SimulX*. It relies on three concepts: abstraction and modularity, wireless and mobile communication and re-usability. Note that the inputs / outputs design is presented in the next section.

4.1 Simulator kernel and events

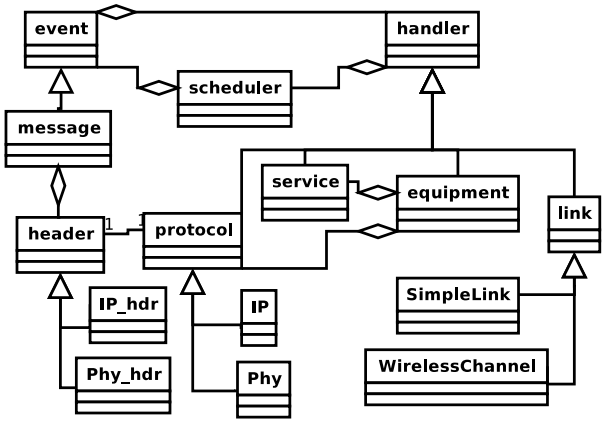

Figure 1. SimulX class diagram

SimulX is an event-driven C++ simulator. The main class hierarchy is depicted in Figure 1. An event is defined as a

 ISBN 1-56555-312-8

condition that occurs at a given simulated time witch causes changes in the state of the system. The C++ class *Event* represents the parent class of all events in the program. Events may be scheduled, defered (e.g., backoff timer), cancelled, or directly sent to another destination manager. Events can be of three different types (via C++ derivation); either an event is an internal event of a protocol (e.g., timer), and represents a transition of the given protocol. Or, an event can be a message (class *Message*) that is received or sent by a protocol; A message can be viewed as a framework to exchange information between protocols, and to simulate a data unit exchange between nodes or layers (e.g., IP packet). It is composed of a stack of headers and contains an attibute which indicates the direction of the message (up or down). Finally an event can also be an indication of movement of a node (or set of nodes). When this event is treated, the corresponding device(s) is (are) moved to a new location.

Events are globally managed by a *Scheduler*, and locally treated by a *Handler* (see Figure 1). The Scheduler is the main entity in the simulator that schedules and dispatches events in a centralized manner. Within each simulation run, a single instance of the Scheduler manages the time of the simulation, and the operations on the list of events, namely the *Calendar*. The Calendar is a heap list of pointers on events. A heap list was chosen, in order to optmize insertion. The execution of Events consists of finding the event that has the smallest execution time, and to call its Handler. By ordering the list when an event is inserted allows executing an event very quickly (o(1)). Via the Scheduler's tools to manipulate the Calendar, it is very easy to use a new structure instead of heap, or to enhance some particular methods in the future. The objective of this first version of SimulX is to propose an events management that is useable for the initiation of the project. Further research on this aspect is our considered as our future work.

The *Handler* is the consumer of an event. Every event contains a reference (i.e., pointer) to the entity in charge to handle (consume) it at the appropriated time. A Handler is usually a protocol which will manage the event. For example, if the event is a message coming from upper layers, the protocol must initiate its forwarding procedure and schedule its processing. We could say that the Handle function is the entry of a protocol. In SimulX, we implemented the Handler class as a virtual class which has a single function *handle*. Every protocol inherits this methods and thus manages events. When a protocol manages an event, it directly accesses to the object through its address. Thus it can modify any field of the event, and re-schedules the event (for itself or another protocol). This allows using efficiently the memory when a message goes down the TCP/IP layers. The only exeption to this mechanism is the broadcast of a message. When several entities must receive the same message at the same time, the message is duplicated to all receivers.

4.2 Protocols as modules

SimulX consists of a set of modules with pre-defined inputs/outputs. Modules are mainly protocols which are all derivated from the mother class Handler which represents a high level abstraction of a protocol. This mother class defines the basic functions that a protocol will need, i.e., handle, send and receive an event. This design allows plugging protocols together in an arbitrary manner, which allows very fine granularity of simulation complexity: if a certain layer is not needed in a device for the expected demonstration, this layer can be omitted. Moreover, for each layer of the TCP/IP stack, there is a generic implementation with limited capabilities which can be used in addition to more complex model. This allows having first a common mother class for a protocol with the basic functions, and second to use a simple model of a specific layer for specific simulation. A special protocol is also defined in the simulator and is called *Service*. A Service is different from a protocol in the sense that it is an end-point where an application can be defined. It has not any up target.

Protocols are connected together within a node via two pointers arrays. The pointers array *up_target* and *down_target* contain the set of addresses of immediate upper and downer protocols, resepctively. When there are multiple protocols above or below a given layer, there is a selection algorithm that determines which is the next protocol (e.g., two mac interfaces for a single IP instance).

4.3 Building devices

The nature of a device is determined by the set of protocols it is made from. When a new device is set up, the user chooses the set of protocols that compose the device. This set of protocols will then determines whether the device is a single host, a server, an access point or anything else. This comes from the fact that the design choice for SimulX is based on the possibility to do anything with any kind of nodes. For example, it would have been bad not to make a router mobile.

However, it is still possible to create specific nodes via dedicated functions. In order to ease the use of SimulX, there is a number of pre-defined devices that can be created by setting only a few parameters. For example, to simulate a IEEE 802.11 access point, a user can simply use shortcuts buttons that configure a standard access point.

4.4 Links

Devices are linked together via *Links*. A link is an entity that collect messages and forward them to the attached interfaces. Indeed, a link has a list of interfaces that are physically (wired) or virtually (wireless) connected to it. Several links models are defined, such as point-to-point link or wireless links, and their behaviors are very different. For example, a *SimpleLink* with a specified delay can be used to connect together two devices.

A wireless link is more complex. A wireless link consists of a channel in the radio frequency, and determines the signal attenuation at the receiver. It also includes a function to determines the affected nodes when a signal is transmitted by a node on a specific channel. Adjacent channls interferences is also taken into account, by calculating the interference caused by a signal on interfaces attached to adjacent channels.

4.5 Librairies

Specific librairies to import codes from other programs will be available in SimulX. The main idea behind SimulX is not to start over a new project, but to put several pieces together in an efficient way. The aim of dedicated librairies is to be able to include other simulator's or operating system's code into the SimulX framework. At this time, we are developping a librairy to import NS's code, and we are working on the integration of IEEE 802.16 [1]. In the next section, we highlight inputs and outputs of SimulX.

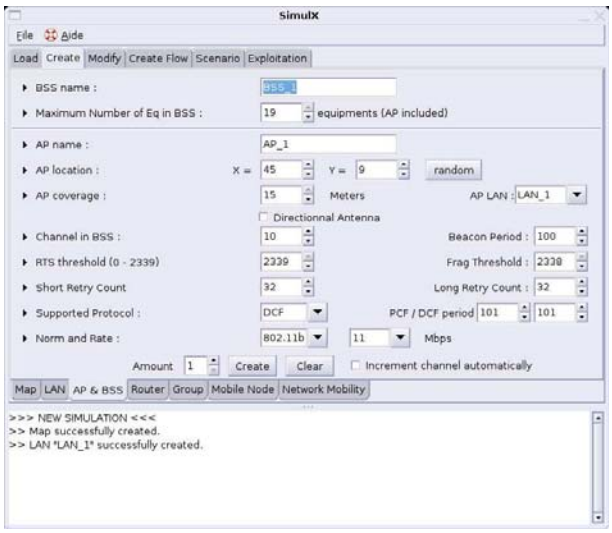

Figure 2. Screenshot of the SimulX graphical interface

5 GRAPHICAL INTERFACE

A significant benefit of SimulX is its graphical interface, which provides a simple manner to create simulation scenarios from scratch. Various network simulators require users to be familiar with the simulator itself before to fully exploit its features and create simulation scenarios that match their needs. By contrast, SimulX removes this constraint by providing a more accessible method to users to create simulation scenarios.

The graphical interface of SimulX is developed with the GTK+-2 library (Gimp Tool Kit version 2) and lies on a tabbed browsing interface. All parameters that SimulX implements are configurable through the graphical interface. Once a user wants to set up a new scenario, the first step is to define the size of the area in which the devices take place. Then, the user configures the number of IPv6 subnets and their bandwidth. The next tabs refer to the creation of the network devices: Wi-Fi access points, IPv6 routers, fixed and mobile nodes. Note that the interface allows the creation of several devices of the same type simultaneously. Once the environment of the simulation scenario is set, the user can add data flows between nodes. At last, the movements of the mobile nodes can be defined.

When the scenario is complete, the user can select among all of the available log files (which provide specific simulation results) those that match his request. Then, the scenario can be saved for later application, or/and the simulation can be started. All previously saved scenarios (described into ASCII files and located in a specific directory) can be reloaded into the graphical interface in order to replay/modify these scenarios. The graphical interface of SimulX is illustrated in Figure 2.

In addition to the various set of configurable simulation parameters, a map providing a visible overview of the current status of the scenario is available. This map is developed with the OpenGL library (Open Graphics Library). The map displays all of the simulation devices, the wire and wireless links between these devices, and the wireless area coverage of each wireless device (i.e. access points and mobiles nodes) separately or not. The map also displays a grid in which the grid lines are placed every meter (for X-axis as well as for Y-axis). Each device is located into one of the squares resulting from the grid lines. Taking benefit from OpenGL features, the user can zoom in or out on the map. In addition, the simulation devices (e.g. access points, mobile nodes, access routers, etc) can be moved over this map through the well known *drag and drop* method. As a result, this map enables the user to keep an eye of the current status of its scenario throughout the creation process. It also provides an accessible method to interact with the scenario in order to readjust it. Figure 3 represents the map included in the graphical interface of SimulX.

An additional feature of the graphical interface of SimulX is the visualization of the entire simulation process. Upon simulation completion, the user can replay this simulation and visualize all events within a second map. This map is also developed with the OpenGL library. Node movements and packet transmissions are observable in real time according to the simulation timescale. During the replay process, the user can pause, speed up or down the progress of the simulation to catch a specific event or a series of event. Therefore, the replay feature enables the observation of the behavior of network protocols or protocol optimizations with ease, rather than laborious analysis of log files.

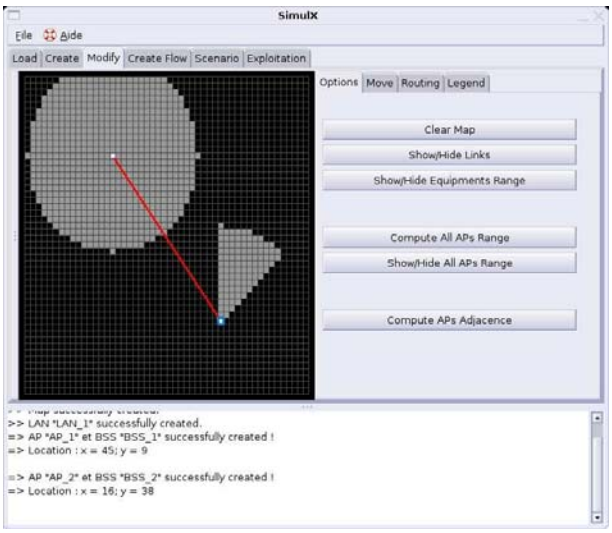

Figure 3. Screenshot of the map available in the graphical interface of SimulX

6 ACHIEVED RESULTS

6.1 Available protocols

In the current state of SimulX [2], the following protocols are already available:

- True IEEE 802.11 model and IEEE 802.11i
- Standard Layer 2 handover and optimization (anticipated scanning, accces point cache, geolocation assisted handover)
- Mobile IPv6 and optimization (Fast RA, L2 triggering, anticipated handover)
- IPv6 protocols (Neighbor Discovery, DNA)

6.2 Usage example

The main purpose of a network simulator is to provide a first overview of the behavior of a network protocol or optimization. Among the various experiments that SimulX enables, we choose to focus in this section on an example that best reflects the SimulX capabilities. Note that SimulX is daily used in our research and results presented in several publications rely on SimulX outputs, such as those in [7] and [8].

When considering the mobility of users among wireless networks such as Wi-Fi, one of the main issue is the handover latency, which refers to the connection loss time of a node while roaming from one wireless access point to another. In this particular domain, the literature is full of new proposals which enhance the handover process as it is described by the standards. As SimulX fully implements the IEEE 802.11 standard and the Mobile IPv6 protocol (which are the common solutions to enable mobility in Wi-Fi IPv6 networks), it is particularly well adapted to simulate the handover process at layer 2 and 3 of the TCP/IP model. Furthermore, it is quite easy to implement in SimulX new handover procedures and optimizations taken from the literature in order to experiment and compare their performances.

As previously mentioned, SimulX can generate log files which include the outputs of a simulation. Among them, specific log files include directly the significant information when considering the handover latency and results related to. Thereafter it is quite simple to compare several protocols or optimizations by analyzing the data provided by these log files. In research community, however, the results are generally presented as figures or tables. But the accessible format of the log files resulting from SimulX allows the employment of simple scripts to readjust the content of the log files to fit the syntax required by the plotting tool used to generate the figures. Figures 4 and 5 provides examples of such figures based on SimulX log files and generated with the plotting software Gnuplot.

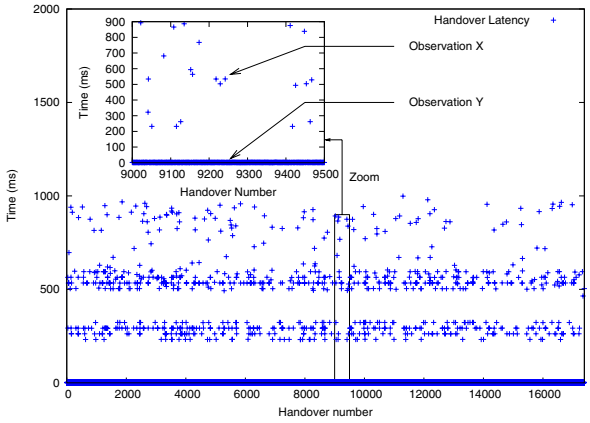

Figure 4. Handover latency

Figure 4 shows the handover latencies resulting from all of the handover procedures performed in a simulation. Each dot represents the delay taken by a single handover. The main benefit of such figure is to provide an overview of the data distribution whereas average values may be meaningless.

Figure 5 illustrates the impact of a handover on data reception. The dot type refers to a specific handover management or optimization. Each dot represents the reception of a data packet by the mobile node, at the time indicated in the Y-axis. Such type of figure is common and points out the gaps representing missing packets due to the connection break when a handover occurs.

These figures were obtained through various handover mechanisms that are provided with SimulX. These methods are described in a single file *handover.cc*, where operations of a mobile node and triggers are defined. From a user point of view, the methods to choose between is very convenient, as a

single field has to be specified in a node configuration.

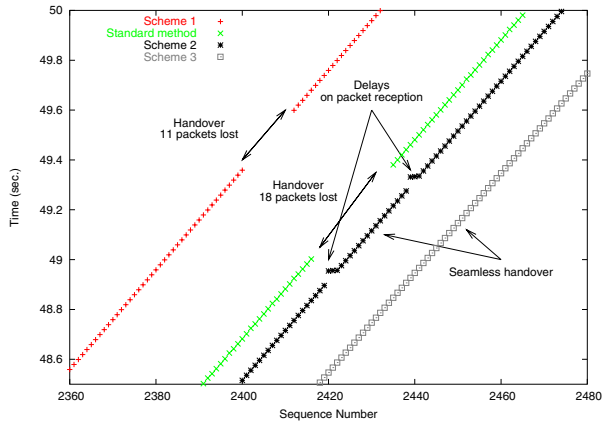

Figure 5. Impact of handover on flows

7 CONCLUSION

In this paper we have studied what a simulation framework should provide for IPv6 research, and proposed a new simulation tool called *SimulX*. The next generation Internet will be mobile and extensively use multihoming to provide redundancy, failure recovery and ubiquitous Internet. In parallel, wireless communications become more and more common day after day. It is crucial for the development of tomorrow's protocol to provide accurate framework to evaluate emerging solutions.

This paper has two purposes. First, through a comprehension of the current trends of the Internet, it describes the features that an evaluation tool must support. These requirements are defined independently of any tool, and can be used in other context (e.g., testbed experimentation). The second important aspect of the paper is the overview of SimulX. SimulX is a new C++ event-driven simulator which provide an open framework for the evaluation of multihoming and mobile protocols in an IPv6 world. While still being developed by different laboratory, some features are already available, such as IEEE 802.11 or Mobile IPv6. These first protocols demonstrate the power behind SimulX.

Our future work will focus on three aspects. First, we are now integrating other layer 2 technologies in order to set up the multihoming support. We are focusing on IEEE 802.15, IEEE 802.16 and cellular networks. Second, we are working on librairies which will enable to import codes from other tools. As a first step, we plan to merge codes from NS-2 and OpenBSD. Third, we are still working on the optimization of SimulX core and continuously update the graphical interface to include the last changes we made.

REFERENCES

[1] IEEE 802.16 implementation in NS-2, http://w3.antd.nist.gov.

[2] Next Generation networks simulator SimulX, http://simulx.u-strasbg.fr.

[3] The ns manual(formerly ns notes and documentation).

[4] C. J. Lindblad D. Wetherall. Extending tcl for dynamic object-oriented programming. July 1995.

[5] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6), Internet Engineering Task Force Request For Comments (RFC) 1883, December 1995.

[6] Geoff Huston. Multi-homing and identity in ipv6. Internet Society Publications, June 2004.

[7] J. Montavont, N. Montavont, and T. Noel. Enhanced Schemes for L2 Handover in IEEE 802.11 Networks and their Evaluations. In *Proceedings of the 16th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*, volume 3, pages 1429–1434, Berlin, Germany, September 2005.

[8] T. Noel N. Montavont. Fast movement detection in ieee 802.11. *Wireless communication and mobile computing, special issue on Mobile IP*, to appear.

[9] Pekka Nikander, Jukka Ylitalo, and Jorma Wall. Integrating Security, Mobility, and Multi-Homing in a HIP Way. In *in Proc. Network and Distributed Systems Security Symposium*, pages 87–89, San Diego, CA, February 2003.

[10] Charles Perkins and David B. Johnson. Mobility Support in IPv6. In *ACM/IEEE International Conference on Mobile Computing and Networking*, 1996.

[11] A. Vagra. The omnet++ discrete event simulation system. *European Simulation Multiconference (ESM 2001)*, June 6-9 2001.

[12] A. Varga. Using the omnet++ discrete event simulation system in education. *Education, IEEE Transactions on*, 42(4):11pp., 1999.

[13] Gang Wu, Paul J.M. Havinga, and Mitsuhiko Mizuno. Wireless Internet over Heterogeneous Wireless Networks. In *IEEE GLOBECOM, IEEE Computer Society ISBN 0-7803-7208-5*, pages 1759–1765, San Antonio, November 2001.

[14] G. K. Egan Y. A. Sekercioglu, A. Vagra. Parallel simulation made easy with omnet++. *European Simulation Symposium and Exhibition*, 2003.