Design and Evaluation of Host Identity Protocol (HIP) Simulation Framework for INET/OMNeT++

László Bokor Szabolcs Nováczki László Tamás Zeke Gábor Jeney

Budapest University of Technology and Economics, Department of Telecommunications
Magyar Tudósok krt.2., H-1117 Budapest, Hungary
Tel: (+ 36 1) 463-3261, Fax: (+ 36 1) 463-3263
{goodzi, nszabi, koci, jeneyg}@mcl.hu

ABSTRACT

Host Identity Protocol (HIP) decouples IP addresses from higher layer Internet applications by proposing a new, cryptographic namespace for host identities. HIP has great potential in means of mobility and multihoming support, security, and performance, such making it quite a promising candidate as the basic architecture of the Future Internet. However, HIP is still in development and very early standardization phase: the protocol is continuously evolving due to its adaptivity to functional changes and extensions. Aiming to completely understand the protocol's behavior, its applicability to wide-scale usage and to analyze current and future improvements and enhancements, it is crucial to develop a proper, RFC-compliant, extensible simulation model for Host Identity Protocol. In this paper we present the structural design and the functional details of our HIP simulation framework (called HIPSim++) integrated into the INET/OMNeT++ discrete event simulation environment. In order to evaluate the accuracy and preciseness of HIPSim++, we designed a real-life HIP testbed and compared the simulation outcomes with the reference results obtained from this HIP testing architecture. Our analysis show excellent accuracy and consistent operation of the simulation framework in terms of handover metrics (latency, packet loss, throughput) and behavior when compared to the real-life experiences of the HIP testbed.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

I.6.5 [**Simulation and Modeling**]: Model Development - *Modeling methodologies*

I.6.6 [**Simulation and Modeling**]: Simulation Output Analysis

I.6.8 [**Simulation and Modeling**]: Types of Simulation – *Discrete event*

General Terms

Measurement, Performance, Design, Experimentation, Security.

Keywords

Protocol Simulation, OMNeT++, INET, C++, Host Identity Protocol (HIP), Simulation framework, Real-life HIP testbed, Handover performance.

MSWiM'09, October 26–29, 2009, Tenerife, Canary Islands, Spain.

1. INTRODUCTION

In the current Internet architecture, nodes are identified by IP addresses depending on the actual topological position of the nodes. Therefore IP addresses are simultaneously describing both the location in the network and the identity of a particular node. The semantic overload of the IP addresses causes problems. The most prominent among them is mobility management: when the node changes its attachment point to the network (and thus its IP address), active sessions (which are mostly connected to the TCP/IP numbers) are interrupted. Obviously users want seamless handovers with continuous connections and sessions, so engineers must find an answer here.

The main problem is the duplicate role (or semantically overloaded nature) of IP addresses: they both identify the user itself and its location. From the data link layer, we see that the main role of IP addresses is to support the routing of IP packets (locator role), but from the transport layer's point of view the IP addresses are mainly responsible for the identification of the sessions (identifier role).

There are many existing solution to divide these two roles of the IP addresses: they are mainly introduced in the context of IPv6 mobility management. Nevertheless, they all provide a solution to separate the functionalities somehow. One of the existing solutions is called Host Identity Protocol (HIP). HIP introduces a new layer between the network layer (IP) and the transport layer. The new HIP layer uses dedicated, cryptographic identifiers (called Host Identity), which represents the identifier role in the sessions: transport protocols rely on these HIs, not the IP addresses. Thus, IP addresses remain only geographical locators; they purely identify the location of the nodes and no longer take care of the identification. With HIP, mobility and even multihoming is not a problem any more: nodes can simply change their IP addresses without loosing active connections.

However, HIP is also useful for some more advanced functionalities. For instance, with portable HIs it is quite easy to support application mobility: applications can be transformed from one node to the other by passing the HIT identifier between the nodes. The authors of this article believe that HIP can be applied for a plenty of purposes and it can provide nice solutions in different areas. This is the reason why the development of a complete HIP simulator framework and the extensive validation of the developed protocol are needed.

This paper is organized as follows. Section 2 summarizes the principles of Host Identity Protocol. The most important acronyms and concepts of HIP are introduced here. Section 3 is about the HIP simulation framework (HIPSim++) we have developed. It is based on OMNeT++, and the INET framework of

OMNeT++ has been used as a basis of development platform. All interesting details of our simulation model are given in this section. In Section 4, the real-life HIP testbed is described, together with the HIP implementation (InfraHIP) we used for validating our model and implementation. Section 5 discusses and evaluates the results obtained from simulations and real-life measurements. The comparison shows that the simulator we have developed seems to properly model the HIP protocol. Finally, Section 6 concludes the document, and flashes some possible future works.

2. OVERVIEW OF HOST IDENTITY PROTOCOL

In this section we shortly summarize the HIP concept and functions, which were implemented in HIPSim++ so far. Note that our goal here is to give an overview of HIP's features rather than to present a detailed description, which is out of scope of this paper and can be found in [1][2][3][4][5][6][7].

As mentioned in the introduction, HIP aims to separate the different roles of IP addresses. While IP continues to act as pure locator, HIP introduces a new, globally unique namespace (the Host Identity namespace), which is a pool of identity representations called Host Identifiers (HIs). These namespace elements are of cryptographic names used to identify nodes. Every HIP node has at least one HI and implements the functions required to handle the new namespace. The scope of the protocol includes the modifications and new methods that integrate the concepts of HIP into the existing Internet architecture. These functions form a new protocol layer, which resides between the transport and network layer. HIP separates application and transport layer connections from IP addresses thus enabling effective application of communication security techniques and mobility management [1].

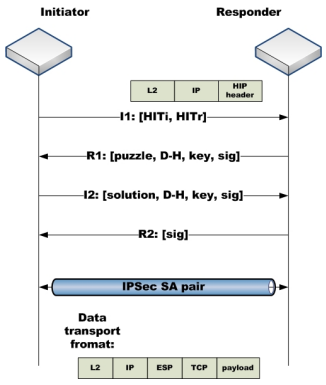

Figure 1. The HIP Base Exchange

HIs could be of any globally unique namespace but during the protocol design they were decided to be cryptographic names, namely asymmetric key pairs. This enables the integration of strong security features such as authentication, confidentiality, integrity and protection against certain kind of Denial-of-Service (DoS) and Man-in-the-Middle (MitM) attacks and some other security threats. However, HIs are rarely used in HIP protocol packets. Instead a 128 bit long representation called the Host Identity Tag (HIT) is applied in HIP control messages. The HIT is created by taking a one-way hash on the HI. For local means and to enable IPv4 compatibility it is necessary to also have 32 bit representations of His; these are the Local Scope Identifiers. HIP related information is passed by HIP headers, which have a form of a standard IPv6 extension header.

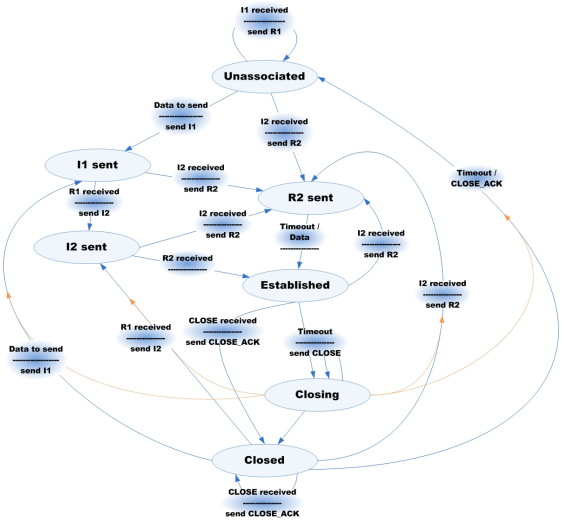

Figure 2. The HIP state machine

A HIP association can be established between two nodes (i.e. an Initiator and a Responder) by a four way end-to-end security handshake, the Base Exchange (BE) (see Figure 1). The BE authenticates the peers by asymmetric keys and implements a Diffie-Hellman key exchange to create symmetric keys for later payload encryption. Moreover, a special puzzle-solution mechanism is applied to protect the responder against certain DoS attacks. As a result of a successful HIP Base Exchange an IPSec Security Association pair is created between the peers. After the BE payload data is passed between the peers using the Encapsulating Security Payload (ESP). Note that HIP related info (i.e. the HIP header) is only applied to HIP control packets but not in case of data transfer messages.

The inside protocol behavior is based on the protocol's state machine [2], which controls a HIP association during its lifetime. Figure 2 summarizes the HIP state machine. The picture shows the states and the state transitions a HIP association can suffer as long as it exists. We also indicated the correspondent trigger that starts an actual transition.

Using HIP mechanisms, the application data is transferred between the nodes through a special IPSec ESP tunnel. A new transport mode of ESP was designed especially for HIP [3]. This so called Bound-End-to-End-Tunnel (BEET) mode integrates the ESP tunnel mode with the low overhead transport mode. Using BEET mode the outer IP header of the ESP packet holds the IP addresses of the peers but the inner header is missing. Instead the Security Parameter Index (SPI) is used to identify the correspondent HIP association by reception at the destination.

A HIP association can be refreshed applying the UPDATE mechanism of the protocol (Figure 3) [6]. The mobile simply sends an UPDATE HIP control packet to all of its peers with a special parameter, the LOCATOR that holds the new IP address(es). Before updating its association, the peer verifies the new address by sending an UPDATE packet to the mobile node,

which requests it to echo back some nonce information. An address becomes verified if this nonce was echoed back from that address. A HIP node can communicate with unverified addresses too but only for a limited time. This is controlled by the Credit-Based Authorization (CBA) mechanism. This protects the updated peer against redirection-based flooding attacks when using unverified addresses. A counter is maintained for every HIP association. The counter is increased by every incoming packet with the size of the packet. The counter is decreased by every outgoing packet with the size of the packet if an unverified address is used. The next packet can be sent out only if the counter is greater than the size of the packet. Otherwise it must be dropped or buffered until sufficient amount of credit is gathered again. The counter is periodically decreased with a constant rate even if there are no outgoing packets. This is called Credit Aging and helps to prevent nodes against attackers who may collect large amount of credit during a long time to spend them very fast possibly performing some kind of DoS attack.

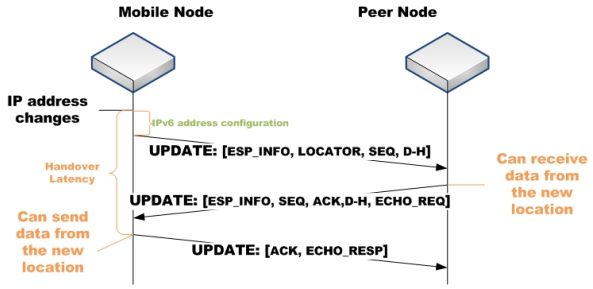

Figure 3. The UPDATE mechanism

The initial reachability of mobile nodes is not solved by the basic readressing functionality of HIP. Thus the basic protocol was extended by a new registration mechanism [5] and the HIP rendezvous service [7]. Special HIP nodes, called Renezvous Servers (RVSs), offer their features for mobile nodes to store their actual HIT-IP mappings and to make them available to potential communication partners. Mobile nodes register and update their IP address at RVSs with the registration mechanism and store the IP address of the RVS in DNS system [4]. If a peer wants to establish a HIP association with the mobile, it learns the actual RVS IP address with a DNS query and sends the first packet of the BE to the RVS. The server relays the packet to the actual IP address of the mobile node. The rest of the connection build up goes like usual. Figure 4 shows the registration and the RVS mechanisms.

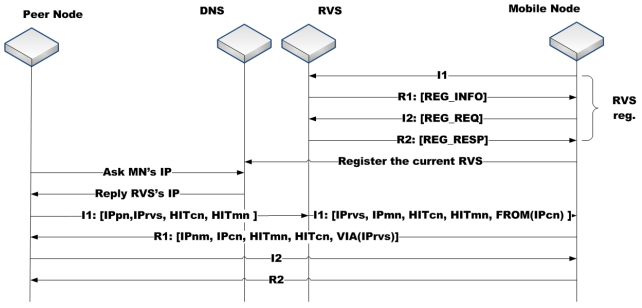

Figure 4. The registration and RVS extension

3. INTRODUCTION TO THE HIPSim++ FRAMEWORK

3.1 Fundamentals of INET/OMNeT++

In this section the basics of INET/OMNeT++ are introduced in order to give an overview about the fundamentals of the simulation environment we used to develop our Host Identity Protocol simulation framework.

OMNeT++ is a discrete event simulation environment for modeling communication networks, IT systems, queueing networks, hardware architectures, multiprocessors, distributed or parallel processes and other systems [8][9]. OMNeT++ aspires to be the optimal solution between open-source, research-oriented simulators (like NS-2 [10]) and the high-priced commercial softwares (like OPNET [11]). Therefore OMNeT++ is public-source, and under the Academic Public License it is free to use for non-profit aims.

OMNeT++ is component-based and has a modular structure: a simulation model consists of modules communicating with message passing (see Figure 5.). The active modules are named as simple modules (they are atomic elements written in C++, using the OMNeT++ simulation class library), while the modules composed from simple modules are termed compound modules (where the number of levels in the module hierarchy is not limited).

Modules communicate with messages thus message sending and receiving are the most frequent tasks in simple modules. Messages contain common attributes (like timestamps) and also arbitrary ones (i.e. any other kind of user data). Simple modules typically use gates (input and output interfaces of modules which can be linked with connections) for sending messages, but direct send to destination modules (using an invocation from the OMNeT++ simulation kernel) is possible as well. OMNeT++ messages can be easily defined by specifying the fields and other possible message content in `.msg` files and by letting OMNeT++ to take care of creating the necessary C++ classes from the `.msg` definitions.

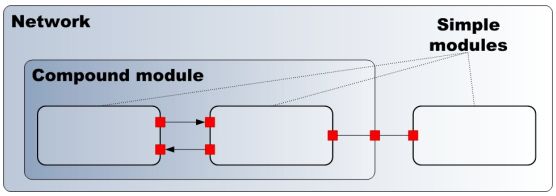

Figure 5. Hierarchy of modules in OMNeT++

A special topology description language called NED (NEtwork Description) is applicable for users to define the structure of simulation models (the modules and their interconnection) in OMNeT++. A typical `.ned` description file consists of simple module declarations (i.e. description of the module's interfaces), compound module definitions (i.e. declaration of the module's external interfaces and definition of submodules and their interconnection) and network definitions (i.e. compound modules that are self-containing simulation models). In this way model behavior and model topology are separated: behavior is defined in C++ code, while topology is determined by the NED language. Simulation parameters (i.e. initial parameters of simulation runs which are independent both from the C++ and the NED codes) are

specified in `.ini` files. Separating initial inputs in this way enable users to run simulations for each one of the interested parameter combination without modifying the existing codes.

Since the first release of OMNeT++, several simulation models have been developed by various research teams and individual contributors for different areas like wireless and mobile networks, ad-hoc and sensor networks, IPv4 and IPv6 networks, wireless access technologies, optical networks, MPLS, file systems, P2P architectures, etc. In the area of communication networks currently the most powerful and most widespread simulation model framework for OMNeT++ is the INET Framework which grew from the IPSuite originally developed at the University of Karlsruhe [12].

INET Framework contains detailed and accurate models for IPv4, IPv6, TCP, UDP, MPLS, RSVP, LDP protocols and several applications like telnet, video streaming, etc. The framework also includes link-layer models as PPP, Ethernet and 802.11b/g (both ad-hoc and infrastructure modes). Static routing can be set up using network autoconfiguration, but dynamic routing provided by routing protocol implementations (OSPF, RIP) also can be used. INET supports wireless and mobile simulations as well: models for wireless communication and mobility have been integrated from the Mobility Framework [13]. As the INET Framework builds upon OMNeT++, it uses the same concept: models consist of modules communicating by message passing. Protocols are usually represented by simple modules in which external interfaces (gates/connectors and parameters) are described in a `.ned` file, and the implementation is prepared as a C++ class with the same name (e.g. IPv6, ICMPv6 modules). The NED language makes possible the integration and free combination of such modules in order to create more complex entities as hosts, other network devices and whole network topologies without writing new C++ code or recompile the simulation. Wide scale of pre-assembled hosts, different routers, switches, access points, and other models can be found in the `/Nodes` directory of INET (e.g. StandardHost6, Router6, WirelessAP, etc.), but it is also possible to assemble new ones from the existing modules, and even to implement your own modules and to use them as supplemental building blocks of new simulation models. Network interfaces (Ethernet, 802.11b/g, etc.) are usually compound modules, and are constructed of a queue, a MAC, and potentially other simple modules. IPv4 and IPv6 network layers are also examples for compound modules in the INET Framework.

Modules of INET provide different functions. There are modules which implement protocols (e.g. *IPv4*, *IPv6*), perform autoconfiguration of whole network topologies (e.g. *FlatNetworkConfigurator6*), store and manipulate data (e.g. *InterfaceTable*, *RoutingTable6*), manage communication between modules in a publish-subscribe manner (e.g. *NotificationBoard*), monitor radio channels in wireless simulations (e.g. *ChannelControl*) and implement mobility of hosts (e.g. *RandomWPMobility*), just to list only the most important ones.

In the INET Framework protocol headers and packet formats are presented in message definition (`.msg`) files, which are automatically translated into subclasses of the OMNeT++'s generic *cMessage* class.

Several authors have contributed with various extensions to the INET Framework. OverSim [14] is designed to model overlay networks and P2P protocols in the INET Framework. AODV-UU, DSR-UU and OLSR protocols are also available as additions for INET [15]. xMIPv6 (Extensible Mobile IPv6) is a simulation model for the INET Framework (named together as INETwithMIPv6) that has been implemented with strict conformance to the IETF specifications of Mobile IPv6 and its protocol family (HMIPv6, FMIPv6, NEMO BS, PMIPv6 etc.) [16] [17].

Our contribution, HIPSim++, is also an extension on the top of INET being developed as part of our ongoing researches in the areas of HIP-based micro-mobility, network mobility and per-application mobility. We hope that HIPSim++ will provide a flexible toolset for validation of HIP protocol extensions and also will make significant impact as the reference HIP simulation model in the INET/OMNeT++ world.

3.2 HIPSim++ basics

In order to provide an extensible and precise simulation model for the Host Identity Protocol, we developed our HIPSim++ framework on the top of the 20081128 version of INETwithMIPv6 [16] [17]. Considering that the 32bit LSIs are designed for local communication only (i.e. the benefits of the HIP scheme can't be exploited totally on IPv4), our HIP framework uses the IPv6 networking stack of INET such fulfilling the requirements of global HIP communication based on the 128bit HITs. Transparency of the novel HIP layer is another important requirement which should be guaranteed for practical reasons, but current HIP RFCs define only a few guidelines to support the transparent behavior of HIP in the current networking architecture. These guidelines were also followed during our development work.

Despite the fact that HIP relies on the functions of IPSec, a full implementation of IPSec and relating algorithms is not part of our simulation model: HIPSim++ does not possess properly realized Diffie–Hellman mechanisms, RSA engine, cryptographic hash functions and puzzles because precise mapping of all the security algorithms is out of scope of our current efforts. The main design goal of HIPSim++ was to accurately simulate core HIP instruments focusing on the advanced mobility and multihoming capabilities and wireless behaviour of the protocol and providing only skeleton implementation of the above mentioned mathematical apparatus.

The simplest scenario of introducing HIP into the ISO-OSI architecture is when applications continue to use IP addresses, and HITs (or LSIs) only appear in the newly introduced HIP layer. Besides the integration of the Host Identity Layer, no other modifications are to be applied in the current protocol stack if such a scenario is implemented. However this is an easy way to introduce basic HIP functions, it also restricts HIP's general benefits of mobility and multihoming support. Therefore our implemented HIP layer registers HIT-IP bonds for every communication session, and when packets from the transport layer arrive, destination and source HITs are replaced by destination and source IP addresses. Higher layers know only about HITs and Port numbers: they are using HITs instead of IP addresses. By realizing this scenario, all the advantages and benefits of applying HIP can be exploited and also HIPSim++ can be easily used in the existing INET-based simulation models.

Current version of HIPSim++ can be downloaded from our website [18].

3.3 Main Modules of HIPSim++

3.3.1 HIP module

3.3.1.1 Description and tasks of the module

The core of our HIPSim++ implementation is the HIP layer module named as *HIP module* which creates a daemon instance called *HIPSM* for every new HIP session. This daemon is responsible for all mechanisms of the HIP State Machine (HIP SM) described in [2], e.g. for handling HIP Base Exchange and HIP mobility functions. One such daemon instance cares of one SA, which will be identified by the local SPI. HIP SM daemons are registered by destination and source HITs (and SPIs) in the *HIP module*. HITs have to be provided by the applications (or rather the transport layer), therefore HIP-capable DNS extensions [4] are also integrated into HIPSim++. The *HIP module* is also responsible for managing changes occurring in the states and addresses of host interfaces.

3.3.1.2 Main methods of the module

- *handleMessage*: Leads the incoming packets towards the appropriate methods (packets can be arrived from the transport and the network layers).
- *handleMsgFromTransport*: Checks whether there is an existing HIP SM for the packet's destination HIT (*hitToIpMap* structure) and forwards the packet towards the appropriate HIP SM.
- *handleMsgFromNetwork*: If the arrived packet is a HIP I1 (see Figure 1), then creates a new SM. If another HIP signaling packet has arrived, then searches for the appropriate SM and forwards the packet to it. If a HIP data message comes in, then the method gets the packet out from the ESP and forwards it to the appropriate SM based on the SPI value.
- *handleAddressChange*: This method is applied by the HIP module in order to gather information about lower layer events (like IP address changes) using the capabilities of the INET's *NotificationBoard* object. After processing such lower layer information, HIP UPDATE mechanisms can be initiated at the relevant SMs.

3.3.1.3 Main structures, variables of the module

- *std::map<IPv6Address,HitToIpMapEntry *>hitToIpMap*: This structure assigns SM identifiers and host locators belonging to HIP connections to destination HITs.
- *std::map<InterfaceEntry *, IPv6Address>mapIfaceToIP*: This structure stores all the interface information (including IP addresses). The data stored in this structure is continuously updated by the HIP module.

3.3.2 HIPSM module

3.3.2.1 Description and tasks of the module

The *HIPSM module* implements the main functions of the HIP State Machine. In our model transitions of HIP State Machine assume that packets are successfully authenticated and processed. This behavior is in consistence with the standards, therefore our skeleton implementation of security algorithms do not hamper our model to accurately simulate HIP mechanisms. One instance of *HIPSM* represents and manages one HIP connection with one Security Association. *HIPSM* handles transitions occurring during HIP Base Exchange, RVS registration, UPDATE mechanism, etc. and generates HIP messages according to the state transitions. *HIPSM* module also handles changes in partner IP addresses (sets the locators by receiving and processing UPDATE messages), but the actual storage happens in the main HIP module's *hitToIpMap* structure.

3.3.2.2 Main methods of the module

- *handleAddressChange*: If an ADDRESS_CHANGED message is received from the *HIP module*, *HIPSM* starts the UPDATE procedure in which an UPDATE message containing the current local locators will be sent towards the partners.
- *handleMessageLocalIn*: Handles packets received from the upper layers. If no HIP connection has been set up for a destination HIT of an incoming packet, then the method starts the HIP BE and stores this first message (*triggermsg*). If a packet arrives for a "BE in progress" HIP connection, then this packet will be discarded. After a successful BE every corresponding packet will be extended with an ESP header containing the appropriate SPI value.
- *handleMessageRemoteIn*: Deals with packets coming from the network. If a corresponding BE or UPDATE procedure is in progress, then this method will generate the appropriate answer messages. If no BE or UPDATE procedures are running for that particular packet, then the ESP packet will be decapsulated and the result will be passed to the *HIP module* for further processing.
- *handleCreditAging*: Implements the basic procedures of the HIP's Credit-Based Authorization (CBA) approach designed to prevent redirection-based flooding attacks. The method is called after receiving *creditMsg* periodical self messages and uses CBA *CreditAgingFactor* and *CreditAgingInterval* parameter values proposed in [6].

3.3.2.3 Main structures, variables of the module

- *int currentState*: Variable for maintaining the current state of a HIP State Machine. Possible HIP SM states are stored in an enumerator called *States*.
- *cMessage* triggerMsg*: The message initiating a HIP BE is stored in this object till the BE finishes.
- *cMessage* creditMsg*: Periodical self message for CBA mechanisms.
- *std::list<IPv6Address> * srcAddressList*; *int currentIfId*: Variables for storing source interfaces and IP addresses. The start of an UPDATE procedure updates these variables.

3.3.3 RvsHIP module

3.3.3.1 Description and tasks of the module

The *RvsHIP module* is derived from the *HIP module* in order to extend the basic HIP capabilities with the RVS functions by handling the incoming registration messages according to [5] and by forwarding I1 messages [7] to the appropriate HIP responder chosen from the registered ones.

3.3.3.2 Main methods of the module

- *handleMsgFromNetwork*: If the destination HIT of an arrived HIP I1 packet is the RVS's own HIT, then registration mechanisms are to be initiated and a new HIP SM is to be created. HIP SM daemons in the RVS are responsible for handling HIP UPDATE messages and corresponding procedures for registered HIP nodes. If the destination HIT of an incoming I1 differs from the RVS's own HIT, then it must be

modified and forwarded towards the appropriate HIP node in the registration list according to [7].

- *alterHipPacketAndSend*: This method modifies the assigned I1 packet: a FROM parameter containing the original source IP address of the HIP packet will be added and the source IP address in the original IP header will be overwritten with the IP of the registered HIP node owning the destination HIT.
- *handleAddressChange*: There is no need to gather information about lower layer events in the RVS, thus this method is overloaded with an empty function.

3.3.3.3 Main structures, variables of the module

RvsHIP module does not possess structures or variables differing from the ones already introduced in the *HIP module*.

3.3.4 DnsBase module

3.3.4.1 Description and tasks of the module

The *DnsBase module* is a simple UDP application which realizes basic DNS server functionality for name resolution of HIP hosts and implements the new Resource Record (DNS HIP RR) defined in [4]. The module resolves domain names to HITs and IP addresses and in case of mobile HIP hosts also provides RVS information. Note, that reverse DNS lookups are not supported in the current version of HIPSim++.

3.3.4.2 Main methods of the module

- *LoadDataFromXML*: Reads initial DNS database containing Resource Records of every host in the simulation from an `.xml` configuration file. Resource Records of a particular HIP host are within the `<DNSEntry>` tag where `<Address>`, `<HIT>`, `<NAME>`, `<RVS>`, etc. tags contain the different fields of a DNS HIP RR.
- *handleMessage*: Processes incoming DNS query messages and answers them by sending DNS responses with the appropriate Resource Records of the queried hosts.

3.3.4.3 Main structures, variables of the module

- *std::vector<DNSData *> DNSVector*: Vector for storing all entries of the whole DNS database after processing `.xml` configuration file.
- *struct DNSData*: Structure for storing entries of one host. Contains IP address, HIT, domain name, and the HIT/IP of the host's RVS (if it has any).

3.4 Special Nodes in HIPSim++

HIP RFCs and Internet Drafts define three main types of nodes, namely the Initiator, the Responder and the Rendezvous Server. For introducing name resolution functions, also DNS server entity is to be used in a HIP architecture. All the above HIP nodes have been realized in HIPSim++ based on the existing INET modules (the `.ned` definitions are located in the `/Nodes/IPv6` directory of INET) and the newly introduced *HIP*, *HIPSM*, *RvsHIP*, and *DNSBase* modules.

3.4.1 Wired HIP Initiator/Responder (HipHosts6)

Wired hosts implementing HIP Initiator and/or Responder functions (i.e. HIP hosts) are derived from the INET's existing *StandardHost6* compound module by inserting the *HIP module* between the transport and the network layers. This node represents a basic HIP host with HIP mechanisms, HIP-based UDP/TCP applications but without support of mobility (Figure 6). The physical network interface is one Ethernet card, but in general any kind of network interface model can be used. HIP hosts contain a single instance of the HIP layer compound module which executes HIP procedures and creates HIP SM daemon instances for every HIP connection.

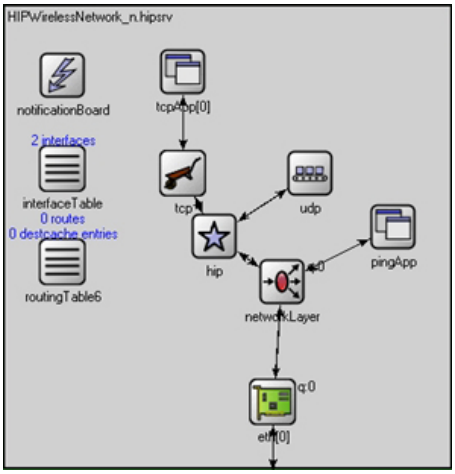

Figure 6. Wired HIP Initiator/Responder (*HipHosts6*)

3.4.2 Wireless HIP Initiator/Responder (WirelessHipHosts6)

It is also possible to integrate WLAN for the physical interface of a HIP host. In this case we are talking about *WirelessHipHost6* nodes which also comprise Mobility Agent in order to realize mobile operation (Figure 7).

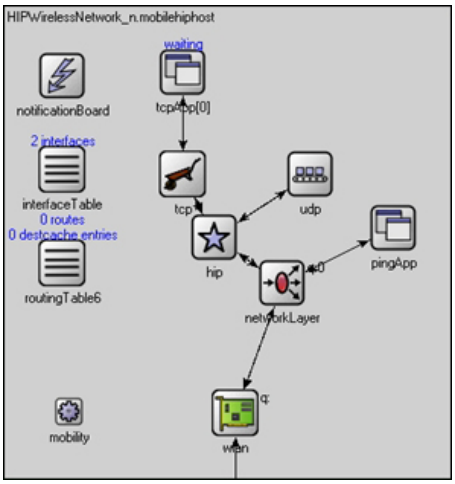

Figure 7. Wireless HIP Initiator/Responder (*WirelessHipHosts6*)

3.4.3 Wireless HIP Initiator/Responder with multiple interfaces (WirelessMultihomeHipHosts6)

In order to exploit the multihoming capabilities of Host Identity Protocol, the number of physical interfaces of a HIP host can be increased freely in HIPSim++. *WirelessMultihomeHipHost6* are implementing HIP hosts with multihoming capabilities, where communication partners are continuously updated about the locator (i.e. IPv6 address) changes of all the active interfaces, and

the most appropriate interface is used for data transmission (Figure 8).

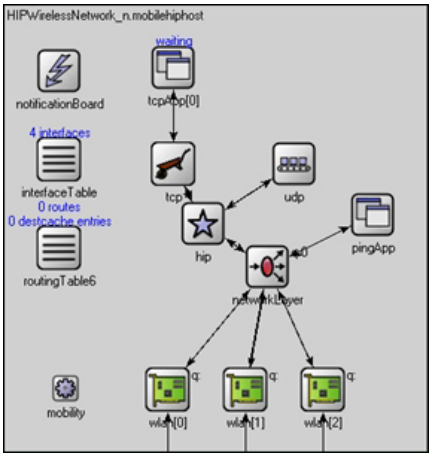

Figure 8. Wireless HIP Initiator/Responder with multiple interfaces (*WirelessMultihomeHipHosts6*)

3.4.4 DNS Server (StandardHost6 with DNS server application)

A DNS Server node in HIPSim++ is responsible to provide name resolution for HIP hosts by implementing the basic functions described in [4]. DNS Server node is basically a *StandardHost6* compound INET module comprising also our DNS implementation called *DnsBase* which runs appropriate DNS mechanisms. At least one DNS Server node is needed for every simulation scenario built in HIPSim++ framework, because (both wired and wireless) HIP hosts are using DNS queries in order to get their partners' initial locators (i.e. directly Responder or indirectly RVS IPv6 addresses). The DNS database used by our *DnsBase* module is an `.xml` file containing resource records of every node in the simulation topology. DNS queries are handled by the Host Identity Layer: the first transport packet initiates the query process based on the destination HIT (and the pre-set DNS IP address), and the Basic Exchange starts right after the response provides with the locator belonging to that destination HIT.

3.4.5 HIP Rendezvous Server (RvsHost6)

RvsHost6 nodes implementing HIP rendezvous functions in our simulation framework are also derived from the *StandardHost6* compound module by interposing the modified HIP module prepared to handle RVS tasks (i.e. the *RvsHIP module*). *RvsHost6* node forwards I1 messages originated by (wired or wireless) HIP Initiators to the appropriate (wired or wireless) HIP Responder signed in the RVS. Therefore potential Responders must register themselves in the RVS and in place of their own IP address, Responders must use their RVS's IP address in the Domain Name System. Wireless HIP nodes must continuously inform their RVSs about events of locator changes.

3.5 HIPSim++ Messages

In this section we introduce the most important message constructions of our HIPSim++ HIP simulation framework.

3.5.1 HIP signaling messages

In accordance to [2], different HIP messages start with a fixed header. The HIP header is logically an IPv6 extension header such in HIPSim++ all HIP messages are implemented as additions to the INET's *Ipv6ExtensionHeader*. Almost all the already standardized HIP message types and parameters are defined in our framework, including also the Locator parameter which is realized as an array of *HIPLocator* structures. An important exception is the ESP_INFO parameter which is missing due to the simplified management of IPSec SPIs in our simulation model.

3.5.2 HIP data messages

In HIPSim++ we currently use the Encapsulated Security Payload (ESP) based mechanism for transmission of user data packets [3]. As proper implementation of all the cryptographic mechanisms in HIP is outside of the scope of our researches, we use only simplified Encapsulating Security Payload Header [19] mechanisms for distinguish HIP data packets based on SPIs. Every HIP data message travels in ESP: packets coming from the transport layer will be encapsulated in an *ESPHeaderMessage* labeled with the appropriate SPI value. Every *ESPHeaderMessage* has a special object (called *IPv6EncapsulatingSecurityPayloadHeader*) per header to carry the SPI value as parameter. This object is derived from the *IPv6ExtensionHeader* class of INET in order to overcome some inflexibility issues of the existing IPv6 implementation and making the ESP packets to pass through the networking layer towards the HIP module.

3.5.3 DNS messages

The basic HIP namespace resolution functions are implemented using a simple query/response message pair called *DnsQuery* and *DnsResponse*.

3.6 Modifications to INETwithMIPv6

In order to implement HIP protocol mechanisms realized by the above introduced HIPSim++ model and to integrate HIPSim++ with the INET framework (INETwithMIPv6 version 20081128), several extensions and modifications have had to be introduced in the existing modules/classes of INET. However these modifications are not transparent to the current INET building blocks, it has to be emphasized that the revision we made is basically a set of additional supplements and extensions without breaking or changing the original functionality. In this chapter we are not intended to present all the details of the corresponding modifications but to give a list with a short overview about the changes and their nature.

- */Network/IPv6/IPv6ExtensionHeaders.msg*: insertion of HIP header and parameter structure; ESP header extended with SPI.
- */Network/IPv6/IPv6Datagram.cc*: integration of mechanisms for HIP header management.
- */Transport/UDP/UDP.cc*: correction of a bug preventing proper UDP communication over IPv6/HIP.
- */Network/IPv6/IPv6.cc*: correction of a bug causing memory leaks during packet transmission towards upper layers.
- */Network/ICMPv6/IPv6NeighbourDiscovery.cc*: introduction a new NotificationBoard message designed to inform the HIP layer about address changes after finishing Duplicate Address Detection procedures of IPv6.
- */NetworkInterfaces/Ieee80211/Mac/Ieee80211Mac.cc/h*: introduction of a simple Radio module identifier for NotificationBoard messages.

- */NetworkInterfaces/Ieee80211/Mgmt/Ieee80211MgmtSTA.cc/h*: extension of NotificationBoard messages with a new object for proper identification.
- */NetworkInterfaces/Ieee80211/Mgmt/Ieee80211AgentSTA.cc/h*: extension of NotificationBoard messages with a new object for proper identification; introduction of a new function for updating InterfaceTable information; introduction of new NotificationBoard messages for distinguishing new and old WLAN Access Points.
- */NetworkInterfaces/Radio/AbstractRadio.cc/h*: introduction of a new function for Radio module identification and ID setup.
- */World/ChannelControl.cc*: extension with support of multiple radio channels.
- */World/ChannelAccess.cc:* extension with support of multiple radio channels.
- */Network/IPv6/Contact/InterfaceEntry.cc/h*: extension with a toolset for managing connection states.

4. HIP TESTBED

In order to validate HIPSim++ in real-life scenarios and certainly to analyze Host Identity Protocol behavior in real-life mobile and multihomed environments, we designed and configured a HIP testbed architecture based on the most widespread HIP software implementation called InfraHIP [20].

In this testbed the HIP hardware infrastructure consists of one HIP rendezvous server, several HIP initiators and responders, a WLAN 802.11b/g access network, and native IPv6 gateway towards public HIP test servers connected to the European multi-gigabit research computer network called GÉANT [21]. The WLAN connectivity comprises IEEE 802.11b/g compatible Linksys WRT54GL Access Routers. This exclusive, local wireless access enables wireless HIP initiators and responders to maintain HIP connections with data rates up to 28Mbit/sec and RTTs around 10ms. For accessing the WLAN access architecture, wireless HIP nodes – based on Fujitsu-Siemens Lifebook C1110D laptops – have been equipped with 3Com 3CRPAG175 PCCARD WLAN interfaces featuring AtherosR chipset. The HIP rendezvous server and wired HIP initiators and responders are standard PCs with 4GHz Pentium 4 processors and 2 GB RAM modules.

As it was already pointed out, the software architecture of our HIP testbed is based on InfraHIP [20] which is currently considered to be the most complete and standard compliant HIP implementation. The project developing InfraHIP focuses on test deployment of Host Identity Protocol infrastructure in order to discover all the effects of HIP's locator/identifier splitting approach in practice, thus InfraHIP supports very wide-scale features including RVS, DNS and even DHT, HI3, and NAT/firewall traversal mechanisms. InfraHIP is a Linux-based, hybrid userspace/kernel implementation of the latest HIP IETF RFCs. All HIP network elements in the testbed run InfraHIP components on Ubuntu 8.10 "Intrepid" operating system with Linux kernel 2.6.27.

The integrated hardware-software architecture of our HIP testbed is shown in Figure 9.

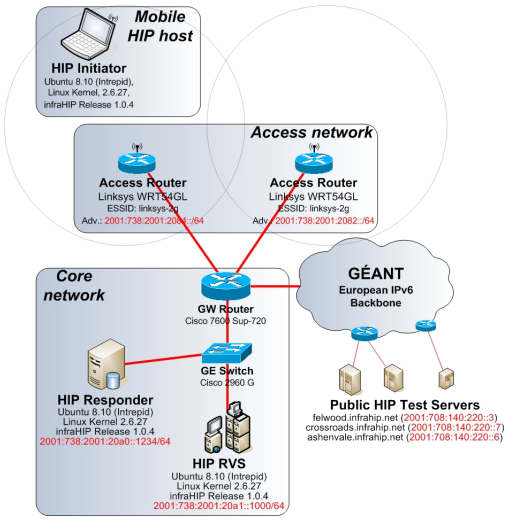

Figure 9. HIP testbed architecture

During the measurements, handovers in the testbed were initiated manually by shell scripts controlling the WLAN connection between Access Points and the wireless HIP node. Tshark [22] was used to capture relevant traffic and shell scripts were used to process log files and collect measurement results to be compared with HIPSim++ simulations.

5. EVALUATION

This section presents the results of our evaluation process based on the following scenario. The mobile HIP host (both the real and the simulated one) changes its network point of attachment by connecting to another WLAN access point. As the APs are in different IP networks advertising different IPv6 prefixes, the IP address of the mobile must be changed. HIP handles the situation by running the UPDATE process described above. In the real world measurements we used a network traffic generator application called Iperf [23] to exchange user data between the endpoints, while built-in INET applications were used in HIPSim++ for the same reason.

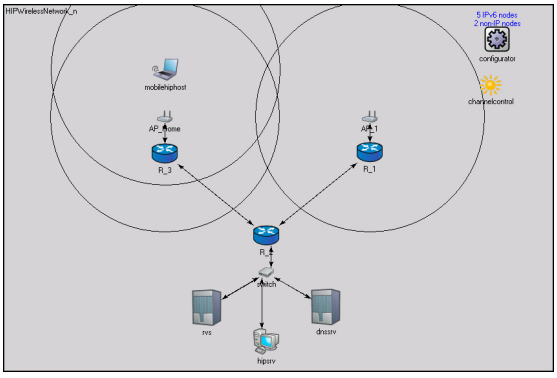

Figure 10. Network topology for HIPSim++ evaluation

The network topology built for HIPSim++ evaluation (Fig. 10) can be considered as an exact copy of our real-life HIP testbed architecture (Figure 9). There are only two differences. The first is the lack of DNS server in the testbed (here the `/etc/hosts` file is used for name-HIT-IP resolution) but it has no relevance in

the comparison: DNS procedures are initiated only before connection establishment (i.e. Base Exchange). The second one is that the testbed comprises IPv6 connection to the GÉANT network while the simulation topology lacks of such feature. This difference is also irrelevant, because GÉANT connection was not used during the evaluation.

Three main scenarios were analyzed during our evaluation. First we measured the handover latency experienced when using HIP. The latency was defined here as the time elapsed between loosing the connection at the old AP and the mobile sending out the third UPDATE packet while connected to the new AP (see Figure 3.). This latency consists of two main parts. First the node has to configure its new IPv6 address by means of stateless auto-configuration using RAs sent by the routers. Second the HIP implementation has to handle the IP address change. This definition of handover latency does not require the nodes' clocks to be synchronized, which makes time measurement much easier. On the other hand the definition is accurate as the HIP association is ready to send data at the point of time as the third UPDATE packet is processed. We made eleven series of measures to produce a more detailed picture on handover latency behavior. The series were differing in the Router Advertisement (RA) interval configured in the access routers. Both real-life and simulated networks were set up to trigger 100 handovers in every series. Nodes were configured to use IPv6 stateless auto-configuration mechanism to obtain new IP address on their interfaces. In such an environment the RA interval is the most relevant network parameter that influences the handover latency. We started the series with min/max RA interval of 0.03/0.07s and finished with min/max RA interval of 1/3s. We applied equal interval increment steps between the two margins. Figure 11 shows the results. RA intervals are represented as the average of their minimum and maximum values applied in the given series. Handover latency is expressed as the average of the 100 handover series. Statistical accuracy is represented by a 95% confidence interval indicated around each point. Results can be considered very close to each other. The maximum difference (i.e. around 0.3 sec) can be observed at the 1.81s average RA interval point.

We also measured how many UDP packets are lost during a handover in a HIP system. To get a detailed picture we repeated the experiment by different data rates offered by the HIP responder. A point on the plot represents the average UDP packet loss of 100 handovers. A 95% confidence interval gives a view on the statistical properties of the results. Looking at Figure 12 we can conclude that the simulated and real world measurements are very close to each other.

As our third evaluation scenario we originated TCP traffic between the HIP initiator and responder nodes and measured the throughput experienced at different handover frequencies. Figure 13 shows the results. Every point represents the average throughput of a one hour long period applying the same value for the number of handovers per every minute of that hour. Between the series we increased the number of handovers suffered per minute form 0 to 10. Results are expressed as a percentage of the throughput of the no-handover scenario, which is the first result on the left (i.e. 100%). We can say that the behavior is quite similar. However the increasing number of handovers has stronger effects on the simulated environment as those results are usually under the real world numbers and its gradient is higher as well. This observation is true until there are more than seven handovers per minute. Reaching this limit we can see a drastic drop down of the throughput in both cases. This is caused by the fact that the TCP connection has not enough time to recover. In this case when TCP send some data and does not get any answer it calculates a certain amount of time during which it won't send any additional data. This time calculation is based on exponential back-off and can cause large gaps in communication if frequent handovers causes TCP to believe congestion occurrence in the network.

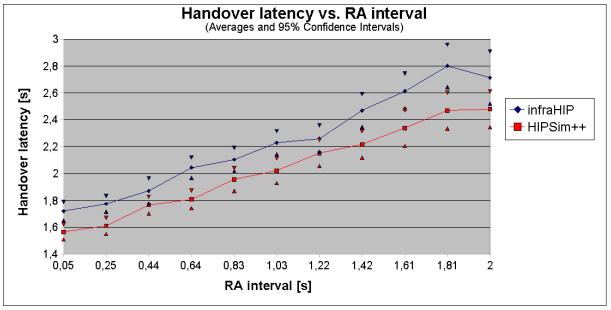

Figure 11. Handover latency vs. RA interval

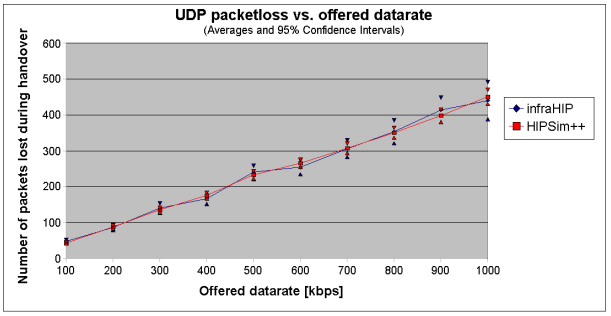

Figure 12. UDP packetloss vs. offered datarate

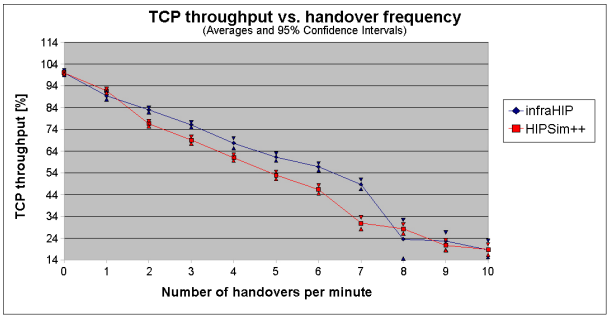

Figure 13. TCP throughput vs. handover frequency

6. CONCLUSION AND FUTURE WORK

In this paper we presented the main design choices and the implementation details of HIPSim++ which is a Host Identity Protocol simulation model integrated into the INET/OMNeT++ simulation environment. In order to assess our model and to evaluate the accuracy and preciseness of the implementation, we built and configured a real-life HIP testing environment based on InfraHIP, and compared the outcomes of our simulation with the reference results obtained from the testbed. Our analysis show

apparent accuracy and consistent operation of HIPSim++ in terms of handover metrics (latency, packet loss, throughput) and behavior when compared to the experiences gathered in the real-life HIP testbed. This accuracy has been provided by modeling HIP messages, nodes and mechanisms based on the actual recommendations of current IETF RFCs, and by re-using the existing detailed IPv6, mobility, channel, etc. models of the INET Framework. We hope that the proved accuracy and degree of reliability of HIPSim++ will make the model ideal for ongoing and future HIP research works.

As a part of our future activities we will further extend HIPSim++ with HIP signaling delegation and service discovery mechanisms, examine HIP multihoming issues, and prepare the model with advanced HIP-based mobility protocols such as micro-mobility, network mobility and per-application mobility.

7. ACKNOWLEDGMENTS

This work is supported by the OPTIMIX project which is partly funded by the 7th Framework Programme (FP7) of the European Union's Information and Communication Technologies (ICT). The authors would like to thank all participants and contributors who take part in the studies, especially for Levente Mihályi who played essential role in the initial phase of this research.

8. REFERENCES

[1] R. Moskowitz, P. Nikander: "Identity Protocol (HIP) Architecture", IETF RFC 4423, May 2006.

[2] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson: "Host Identity Protocol", IETF RFC 5201 (http://www.ietf.org/rfc/rfc5201.txt), April 2008.

[3] P. Jokela, R. Moskowitz, P. Nikander: "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)", IETF RFC 5202 (http://www.ietf.org/rfc/rfc5202.txt), April 2008.

[4] P. Nikander, J. Laganier: "Host Identity Protocol (HIP) Domain Name System (DNS) Extension", IETF RFC 5205 (http://www.ietf.org/rfc/rfc5205.txt), April 2008.

[5] J. Laganier, T. Koponen, L. Eggert: "Host Identity Protocol (HIP) Registration Extension", IETF RFC 5203 (http://www.ietf.org/rfc/rfc5203.txt), April 2008.

[6] P. Nikander, T. Henderson, C. Vogt, J. Arkko: "End-Host Mobility and Multihoming with the Host Identity Protocol", IETF RFC 5206 (http://www.ietf.org/rfc/rfc5206.txt), April 2008.

[7] J. Laganier, L. Eggert: "Host Identity Protocol (HIP) Rendezvous Extension", IETF RFC 5204 (http://www.ietf.org/rfc/rfc5204.txt), April 2008.

[8] Andras Varga, Rudolf Hornig: "An Overview of the OMNeT++ Simulation Environment", in the Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & orkshops (SIMUTools2008), ISBN:978-963-9799-20-2, Marseille, France, 2008.

[9] OMNeT++: A public-source, component-based, modular and open-architecture discrete event simulation environment. Official homepage: http://www.omnetpp.org/ [accessed on April, 2009.]

[10] The Network Simulator – ns-2, Official homepage: http://nsnam.isi.edu/nsnam/index.php/Main_Page [accessed on April, 2009.]

[11] OPNET Technologies, Inc., Official homepage: http://www.opnet.com [accessed on April, 2009.]

[12] The INET Framework for OMNeT++, Official homepage: http://www.omnetpp.org/doc/INET/neddoc/index.html [accessed on April, 2009.]

[13] The Mobility Framework for OMNeT++, Official homepage: http://mobility-fw.sourceforge.net/hp/index.html [accessed on April, 2009.]

[14] I. Baumgart, B. Heep, S. Krause: "OverSim: A Flexible Overlay Network Simulation Framework". In the Proceedings of the 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, pp.79-84., DOI: 10.1109/GI.2007.4301435, Anchorage, AK, USA, May 2007.

[15] Alfonso Ariza Quintana: "INET framework with Manet routing protocols", Personal homepage: http://webpersonal.uma.es/~AARIZAQ/ [accessed on April, 2009.]

[16] F. Z. Yousaf; C. Bauer; C. Wietfeld: "An Accurate and Extensible Mobile IPv6 (xMIPV6) Simulation Model for OMNeT++", in the Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (SIMUTools2008), ISBN:978-963-9799-20-2, Marseille, France, 2008

[17] Mobile IPv6 Implementation for the Omnet++ INET framework, official webpage: http://www.dlr.de/kn/en/desktopdefault.aspx/tabid-4979/8336_read-14161/ [accessed on April, 2009.]

[18] HIPSim++: A Host Identity Protocol (HIP) Simulation Framework for INET/OMNeT++, Official homepage: http://www.ict-optimix.eu/index.php/HIPSim

[19] S. Kent, R. Atkinson: "IP Encapsulating Security Payload (ESP)", IETF RFC 2406, November 1998.

[20] Infrastructure for HIP (InfraHIP): Project focusing on developing the missing infrastructure pieces of HIP, Official homepage: http://infrahip.hiit.fi/ [accessed on April, 2009.].

[21] GÉANT: High-bandwidth, academic Internet serving Europe's research and education community, Official homepage: http://www.geant2.net/ [accessed on April, 2009.]

[22] Tshark: The terminal terminal oriented version of Wireshark network protocol analyzer, Official homepage: http://www.wireshark.org/ [accessed on April, 2009.]

[23] Iperf: Tool for measuring maximum TCP and UDP bandwidth performance, Official homepage: http://www.noc.ucf.edu/Tools/Iperf/ [accessed on April, 2009.]