An Environment for Implementing and Testing Routing Protocols in CARMNET Architecture

Adam Kaliszan
Chair of Communication and Computer Networks
Poznan University of Technology
Poznań, Poland
e-mail: adam.kaliszan@gmail.com

Mariusz Glabowski
Chair of Communication and Computer Networks
Poznan University of Technology
Poznań, Poland
e-mail: mariusz.glabowski@put.poznan.pl

***Abstract*—The paper proposes a new development environment for implementing and testing multi-criteria routing protocols in wireless mesh networks considered under the CARrier-grade delay-aware resource management for wireless multi-hop/Mesh NETworks (CARMNET) project. The paper presents the justification for choosing Optimized Link-State Routing (OLSR) protocol as the reference CARMNET routing protocol for further extension from a single-criterion to the multi-criteria protocol. The existing software router architectures (Quagga, eXtensible Open Router Platform – XORP, and BIRD) are evaluated. A software and hardware architecture of a single wireless node as well as a network testbed are described. The particular phases of the proposed implementation of the modified multi-criteria OLSR protocol in the nodes of testbed environment are presented. Two types of nodes that differ in processor architecture and energy efficiency are proposed for the testbed. The high performance nodes will be used during the development of the routing protocol, while the low performance nodes (energy efficient nodes) will be used during testing the effectiveness of the protocol elaborated. The developed architecture of the testbed network provides the ability to separate the resources exploited by control functions (routing) and data plane functions (forwarding).**

Keywords-routing protocols; mobile ad hoc network; optimized link state routing protocol; software router.

I. INTRODUCTION

Modern wireless mesh networks use multihop transmission in order to provide communication between the nodes that are not in direct transmission range. Currently, the mesh networks are used to extend the range of telecom operators' wireless networks. According to this approach, assumed also in CARMNET project [1], the nodes without direct access to the Internet can use neighboring wireless nodes' (mobile or stationary) help in order to get access to the operators' networks. However, in order to make mesh networks an important solution for telecom operators, development of cross-layer resource management framework is required. Current research on mesh networks focuses mostly on: traffic stream classification, packet scheduling, buffer memory management, routing, and mobility management. One of the key problem in multihop wireless networks, especially in networks that serve heterogeneous traffic, is the problem of optimal routing.

Within the activities related to CARMNET project, a new routing protocol that allows for multi-criteria path selection will be proposed [2]. According to the project's assumptions, the new protocol will be based on OLSR protocol. The new protocol will be implemented in testbeds located in Poznan University of Technology and Scuola Universitaria Professionale SV Italiana (SUPSI) in Lugano, in order to verify algorithmic assumption taken in the proposed protocol, and to perform complex tests of the protocol's efficiency.

The main aim of the paper is to present the proposed implementation platform for elaborating multi-criteria routing protocol. The further part of the paper is organized as follows. In Section II, the choice of OLSR protocol as the reference protocol is justified. This protocol will be modified, in order to allow for multi-criteria path selection. In Section III, the existing software router architectures are described. The decision of reusing the existing implementation of OLSRd [3] and Quagga software router architecture [4] is justified. In Section IV, the software development environment and the hardware testbed, used for implementing and testing multi-criteria routing protocol, are presented. Section V concludes the paper.

II. CHOICE OF ROUTING PROTOCOL

In CARMNET project it is assumed that there is a necessity of implementing a routing protocol, which is capable of building the routing tables including not only the best path but a set of k best paths that lead to the known IP networks/nodes in wireless mesh networks [2]. The paths in the set will be selected as the subsequent shortest paths to the specified destination, based on one of the k-shortest paths algorithms [2]. The paths are determined according to main criterion, e.g., delay, and they include additional criteria (metrics). The additional metrics will be useful in order to choose the best path, that fulfills the criteria for a given traffic stream. The criteria will correspond to the Quality of Service (QoS) requirements for all traffic classes offered in the CARMNET network. An example of the criteria can be delay, a number of hops, link reliability or link load. It is assumed in the CARMNET project that – due to a large number of already elaborated routing protocols

 ISBN: 978-1-61208-279-0

for wireless networks – the specified requirements for the routing protocol can be satisfied by appropriate modification of one of the existing routing protocol (there is no need for design of a completely new routing protocol). The following evaluation criteria were assumed during searching the protocol that can form the basis for further modification:

- The protocol has to ensure the possibility of changing the path determination algorithm,
- The protocol has to ensure the backward compatibility after extension of its functionality,
- The protocol has to ensure the possibility of additional information flooding,
- The protocol should be based on a simple finite state machine,
- The protocol should demonstrate high resistance to packet loss. The loss of a packet as well as receiving an out-of-order packet cannot force the state machine to re-initialize,
- The protocol should provide the possibility of skipping certain nodes during the process of path determination.

As the result of analysing the existing routing protocols for wireless network, the OLSR routing protocol was chosen. Only this protocol fulfills all the criteria presented above.

In OLSR protocol, the Dijkstra's algorithm is used for shortest paths tree determination. This algorithm can be replaced by one of the algorithms for set of k shortest paths determination [8]. It is possible thanks to the OLSR protocol architecture, which allows to extend algorithm functionality in an easy way. The protocol consists of the core part, which is responsible for the main functionality of the protocol, and additional modules, which enable the protocol to be extended. Additionally, this approach enables cooperation between the nodes with and without extensions. According to the OLSR specification, the nodes that do not support protocol extensions are taking part in forwarding additional information (not understandable for them) in a transparent way. In CARMNET project, the OLSR extensions will be used to flooding additional information about links' and nodes' parameters as well as to elaborate a new shortest path algorithm. The specified extensions will result in a new *multi-criteria OLSR* protocol (the flooding mechanism was used in Phosphorus project [9] for broadcasting information about grid resources in Open Shortest Path First version 2 (OSPFv2) link state advertisements). Additionally, the knowledge about links' and nodes' parameters can be used by admission control algorithms.

In the OLSR protocol specification we can observe significant state machine simplification in comparison to other link state protocols. There are only three states defined (NOT_NEIGH, SYM_NEIGH, MPR_NEIGH) [8] in the process of forming OLSR adjacencies (neighbor relationship). This simplification makes easier the implementation, testing and extending of OLSR protocol. In OLSR, the sequence numbers were used to protect the protocol against numerous packet loss and out-of-order packets delivery, occurring in wireless networks. However, in OLSR protocol, in contrast to OSPF protocol, there is no handshake process (sequence numbers synchronization) between two nodes before the routing information exchange begins [10]. In OSPF protocol, there is a special bit that specifies whether neighboring nodes are synchronized or not. After receiving a packet with a wrong value of the synchronization bit, the synchronization procedure re-starts. In the case of OLSR protocol, the last synchronization number (*Packet Sequence Number*) is remembered for 30 seconds. After exceeding this time period, the packet with any value of Packet Sequence Number is accepted. Further, the node accepts only the packets with Packet Sequence Number values greater than the value included in the last message received.

The OLSR protocol allows also to disable forwarding feature in certain of the nodes: these nodes will not forward the packets that are not addressed directly to them. This feature is very useful for the nodes with limited power resources. In order to inform about activation/deactivation of the forwarding feature in the OLSR node, the node sets an appropriate value of the *willingness* field in hello messages sent to its neighbors.

The OLSR protocol implements also an effective algorithm for distributing the routing messages among the nodes. According to this algorithm, each node determines the so-called Multi Point Relays (MPRs) among their neighbors. Subsequently, the nodes send the routing messages only to the selected MPRs, not to all neighbors. The OLSR protocol limits the number of MPR nodes, giving also a possibility of using redundant nodes in order to increase network reliability.

III. SOFTWARE ROUTER ARCHITECTURE

Nowadays, the most popular software router implementations are Quagga [4] and eXtensible Open source Routing Platform (XORP) [11], [12]. Quagga is an unofficial successor of Zebra [5]. The official successor of Zebra project is the commercial product called ZebOS [6]. Because of the General Public Licence (GPL) on which Zebra was based and nowadays Quagga is developed, the Quagga is the most popular software. Quagga architecture has been applied among others in commercial product Vyatta (since version 4.0) [7].

Both solutions (Zebra and XORP projects) have modular architecture. Each module (routing module), is responsible for all functions related to a specific routing protocol implemented in it. Additionally, in both solution, there is a special module responsible for communication between the routing modules and an operating system's Application Programming Interface (API). A new architecture of a software router is proposed within the project BIRD [13]. However, the BIRD project is in initial phase of implementation.

 ISBN: 978-1-61208-279-0

Unfortunately, there is no OLSR protocol implementation in any of the three existing software router projects. There is only one free open source code OLSR implementation, called OLSRd [3]. The OLSRd implementation allows to add new plugins including OLSR protocol extensions, in accordance with OLSR protocol philosophy. Among the implemented hitherto plugins, there is also the plugin for communication with Quagga software router.

Please note that in OLSRd implementation, the program uses directly operating system's API in order to get information about the node's (router's) interfaces and in order to add new paths to the node's routing table. Additionally, this implementation of OLSR protocol uses directly the router's interfaces to send/receive OLSR messages. Such an approach means that the routing protocol (control plane) has to be run on each node performing packets forwarding (data plane). This means, that in OLSRd implementation, the routing protocol functions cannot be transferred to a separate machine, in accordance with the concept proposed in [14].

The OLSRd implementation of the OLSR protocol has no command line interpreter. Therefore, it is impossible to change the parameters of the protocol (the router configuration) during its operation. The only method of setting the parameters of the protocol is to load them directly from a file, during the protocol start-up. Each change in the router's configuration requires modification of the configuration file, and, subsequently, restart of the protocol's software. In the configuration file, in addition to the parameters of the protocol, a list of the required plugins (extending the OLSR protocol functionality) is also included.

Possibility of creating plugins eliminates the need for modification of the core part of the OLSR protocol. Unfortunately, the existing plugins' API documentation [15] is outdated and therefore, in order to create a new plugin it is necessary to study the source code of the OLSRd software. Analysis of the entire OLSRd project's code will be required in one of four phases of the works related to both the implementation of the OLSR protocol and to its modifications.

In the first phase, the OLSRd source code will be used directly. In order to adapt the OLSR protocol to the needs of CARMNET project, a special OLSRd plugin, extending the functionality of the OLSR protocol, will be implemented. The software architecture that will be built in the first phase of protocol development is shown in Figure 1. In Figure 1, the plugin extending the OLSRd functionality is depicted as a dotted line rectangle.

The second phase of the implementation of the OLSR protocol will include its co-operation with other routing protocols. This goal will be achieved due to the application and possible revision of the plugin allowing OLSRd software to communicate directly with the Quagga software. The Quagga software router architecture is shown in Figure 2. The Quagga software consists of many modules, each of

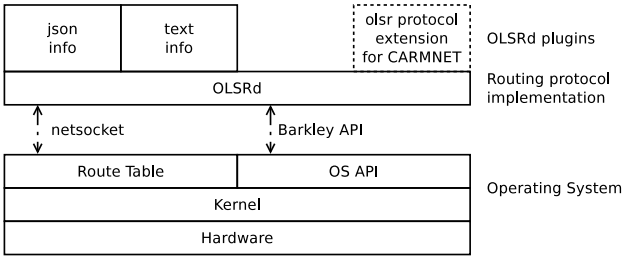

Figure 1. Extended OLSRd software architecture in the first phase of implementation

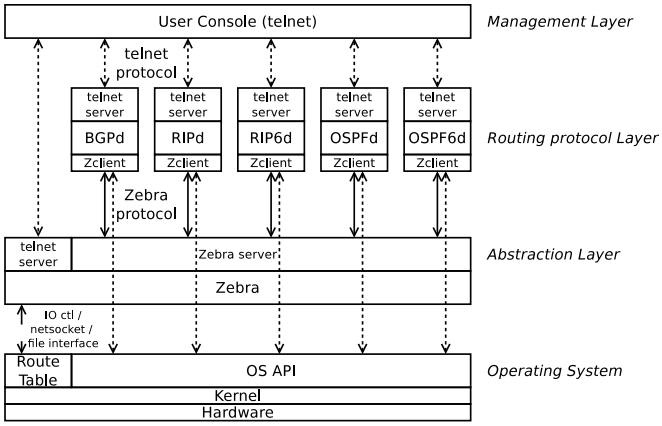

Figure 2. Quagga software router architecture

them is responsible for handling different routing protocol (BGPd daemon is responsible for Border Gateway Protocol BGP, RIPd and RIP6d daemons are responsible for Routing Information Protocol for Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6), etc.). In contrast to the architecture proposed in OLSRd, the modules in Quagga do not refer directly to the operating system's API in order to retrieve information about interfaces and their addresses or in order to add new entries to the routing table. The activities listed above are executed by Zebra module that provides a bidirectional abstraction for the other modules. Communication between the modules is provided by Zebra protocol. Figure 3 shows the architecture of the system, obtained in the second phase of OLSR protocol implementation, after combining Quagga software with OLSRd software.

The software developed in the Quagga project was optimized for sake of its performance, imposing a specific approach to programming as well as the C programming language. For the typical network layer tasks (sending and receiving packets), Berkeley's socket API [16] was used in the Quagga. Consequently, also in this solution the routing protocol software (control plane) has to be running on the same machine the interfaces of which are used. The direct access of OLSRd module to the network interfaces is indicated in Figures 2 and 3 using dotted arrows.

Another approach, which will be used in phase 3 of OLSR

 ISBN: 978-1-61208-279-0

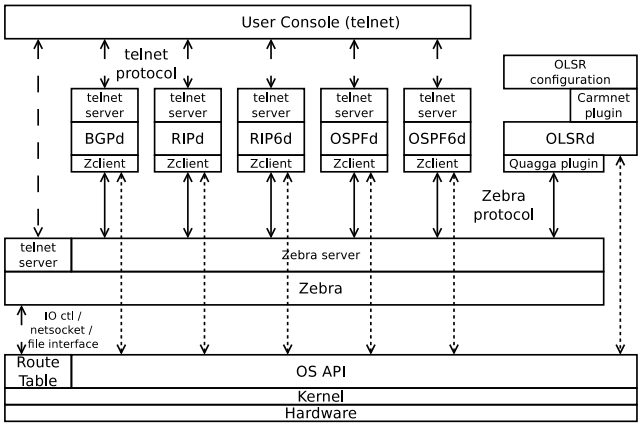

Figure 3. Software architecture in the second phase of OLSR protocol implementation

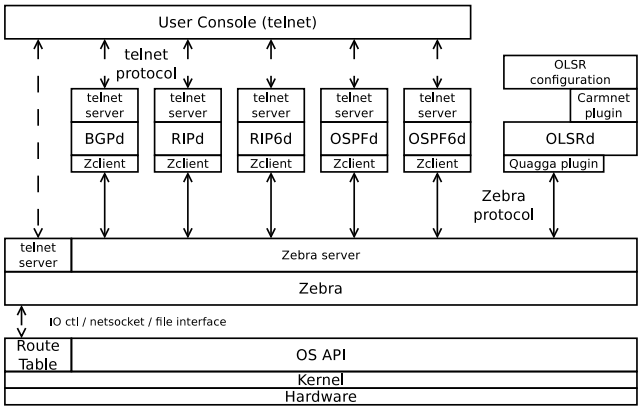

Figure 4. Software architecture in the third phase of OLSR protocol implementation: Zebra module supports sending and receiving packages for other modules

protocol's implementation and modification, was proposed in work [14]. In [14], the concept of modification of Zebra protocol and Zebra module is described. The modification is based on extending the Zebra's functionality by sending and receiving routing protocol's messages. This modification makes it possible to transfer the modules responsible for the routing from the routers to the separate machine. The transfer of the routing protocol to the separate high performance machine allows to perform many interesting experiments. Since we are not limited by (frequently) low computing power of the nodes performing data plane, it is possible to study the influence of complex routing algorithms on network performance. In addition, the transfer of the routing protocol to the separate machine does not enforce the efficient implementation of the protocol when being in experimental stage.

In order to fully integrate the routing protocol software with Quagga architecture, in phase 4 of the implementation a new module for Quagga project will be written. This module will be responsible for supporting both, the reference OLSR protocol and the proposed multi-criteria OLSR protocol. The module will use the same libraries as the other Quagga's modules and will offer a command line interpreter. The use of libraries from Quagga project will significantly reduce the size of the code. In this way it will be possible to install this software on a simple router, supported by the Open WRT system (the concept of a network consisting of devices that support OpenWRT will be presented in the next section).

IV. TESTBED NETWORK

The concept of a testbed network is shown in Figure 5. This network consists of nodes and an internal server that will facilitate the process of software development. All the nodes and the internal server are connected via a network, denoted in Figure 5 as an internal Local Area Network (LAN). This network is invisible to the routing protocols running on the nodes. The nodes are not able to use the internal LAN to exchange packets between each other.

As proposed in the previous section, the software architecture of the routing protocol will be implemented in two types of nodes: the high performance nodes, built using x86 processors, and the energy efficient nodes, built using Reduced Instruction Set Computer (RISC) processors. These nodes will then be used to build the network, enabling the testing and performance examination of protocols, implemented within the CARMNET project. The use of the nodes that were built using x86 processors (hereinafter referred to as x86 nodes) facilitates the implementation of the software, however, such nodes are more expensive and consumes more energy with respect to the nodes using RISC processors.

The main advantage of x86 nodes is that the developing software for this kind of the nodes does not require cross-compilation. The source code can be compiled directly on the developer's workstation or within the node. The simplest solution for x86 nodes is the native compilation of the software at the nodes. In the case of native compilation it is only required to provide the node with a source code. In the proposed architecture, providing software code to individual nodes will take place via an internal development server that grants an access to the repository server. An additional requirement for native compilation is the need for installing the necessary compilers and software libraries at each of the nodes in the network. Consequently, the large memory resources are required in all the nodes during the software compilation. In the case of x86 nodes, each of them is equipped with: a Solid State Drive (SSD) with the capacity of 24 GB, 4 GB of Random Access Memory (RAM) and double core ATOM Central Processing Unit (CPU). Currently, these CPUs offer the best ratio of performance to energy consumption among all x86 CPUs. In addition, the ability of installing a secondary hard disk using Serial Advanced Technology Attachment (SATA) interface in x86 nodes is provided. The interface mini Peripheral Component Interconnect (mPCIE) is used for connecting the primary

 ISBN: 978-1-61208-279-0

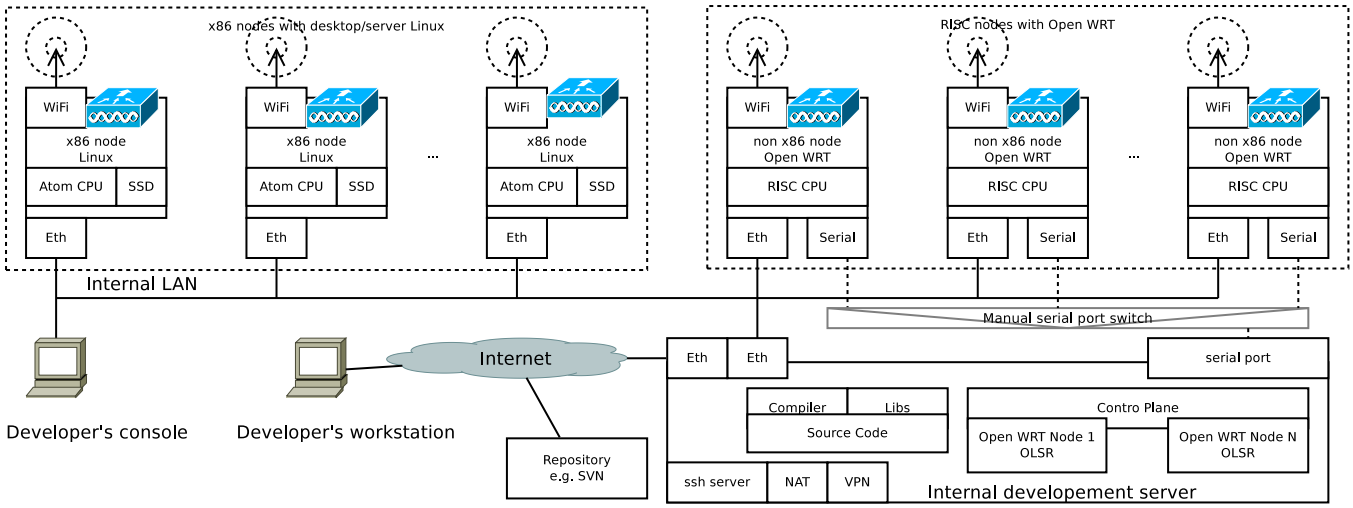

Figure 5. A testbed for development and testing of routing protocols

SSD and for connecting the Wireless Fidelity (WiFi) card. The presented node's configuration allows for installing Linux in both, "server" and "desktop" version. It is assumed that in order to speed up starting the Linux system, a desktop environment will not be installed.

The presented architecture of the nodes based on x86 processors simplifies the process of software development and testing the routing protocols. Due to the high energy requirements and the price of a single node this architecture will not be used in practical deployments. For large networks it is intended to use low cost nodes based on RISC CPUs. It was assumed that these nodes will support Open WRT [17]. Open WRT is a Linux distribution dedicated for routers. It is characterized by small hardware requirements, since a firmware image (including kernel but without WiFi support) often does not occupy more than 2 MB of memory. This is very important because most of the wireless routers available on the market are equipped with a flash memory (for storing operating system's firmware) with the capacity of 8 MB. One of the main advantages of the Open WRT is its ability to run on wide variety of processor architectures. This is possible owing to the fact that the software has an open code, which can be compiled into binary code designed for a specific platform. The compilation is done using a set of tools and libraries called buildchain. A buildchain, used in the Open WRT, is based on buildroot project [18].

In addition, the Open WRT supports ipkg package system, which allows for subsequent installation of software packages without the need for uploading the entire firmware image to a node. Despite the possibility of installing ipkg in flash or RAM memory, such a solution is often unfeasible due to the limited resources of these memories. Therefore, the Open WRT nodes will not be used for testing the routing protocol software being in the second phase of its development.

The nodes with Open WRT may be applied to test the software that is in the third phase of the development. In this phase, the control plane (responsible for routing) and the data plane (responsible for forwarding) are separated [14]. However, the transfer of the control plane from the nodes to the separate machine has some disadvantages. The nodes without the control plane are useless outside the testbed network.

The full autonomy of the nodes is achieved only in the fourth phase of the software development of OLSR routing protocol. In addition, an integration with Quagga software, completed in this phase, provides the minimum size of the module that support OLSR protocol as well as the possibility of launching it in simple routers supported by the Open WRT. The software obtained in the fourth phase will be characterized by the highest performance.

Let us notice that the process of upgrading the operating system at the nodes managed by OpenWRT may be quite complicated. In the case of certain of the nodes it may be necessary to use an additional connection to the node via a serial port. The reason for this is the application of a bootloader that works only with the serial port. The bootloader is used as the easiest way to upload a new operating system image to the node. Unfortunately, uploading the operating system via the serial port is not comfortable: the serial port offers low bit rate and, in addition, requires direct physical access to the device.

The described architecture for the implementation and testing of routing protocols foresees two working modes of the developers. In the first mode, the developers work directly on the internal server that is connected to the same LAN as the nodes (the developer's computer is just a console). The source code can be compiled on the internal development server or on each network node (see Figure 5). The machine, which is building the software, has to have

 ISBN: 978-1-61208-279-0

an access to the source code. One of the simplest yet most effective ways to provide access to the source code is to use the repository system, that is located on a server attached to the Internet. For this purpose, the internal development server performs the function of network address translation (NAT). In the case of teleworking, the access to the internal LAN is secured by a Virtual Private Networks (VPN) tunel. The mode, in which the developer has physical access to the internal LAN, has been denoted in Figure 5 by placing the computer labeled as "developer's console".

In the second mode, the developers are working without any access to the internal LAN. They only have access to the source code located on a repository server, which is connected to a public network. The compilation of the source code can be done either on the dedicated internal development server or on the programmer's workstation. The second mode is denoted in Figure 5 by the computer icon labeled as a "developer's workstation". The main limitation of this mode is that the software developers do not have any access to the network nodes, and, they are not able to perform any tests.

The final component of the proposed architecture for the implementation and testing of routing protocols are repositories that store i.a. the source code of the protocol software. According to the testbed network's architecture presented in Figure 5, the repository server is located in a separate location (accessible via public IP address). The separation of the repository and the internal development server guarantees an access to the source code even after switching the internal development server off (or in the case of its failure). Additionally, according to this solution many testbed networks may use the common repository.

V. Conclusion and Future Work

This paper proposes a testbed network architecture for implementation and testing of multi-criteria routing protocol for wireless mesh networks. The proposed testbed network architecture is programmer-friendly. It provides three ways of compiling the network nodes' source code: the native compilation on the individual network nodes, the compilation on the separate internal development server and the compilation on a developer's workstation.

Two types of network nodes are proposed for the testbed network: high-performance x86-based nodes and energy-efficient RISC nodes. The use of two types of the nodes allows to adapt the testbed network to the evolution of the routing protocol software. It is assumed that the software created in the first phase, due to its sub-optimal use of resources, will require high computing power that can be only provided by the nodes based on x86 architecture. Further optimization of the software will results in lower requirements for computing power in the nodes. Consequently, the final tests will be conducted in a large network, made up of the nodes with lower performance and greater energy efficiency (RISC architecture).

Acknowledgement

This work was supported by a grant CARMNET financed under the Polish-Swiss Research Programme by Switzerland through the Swiss Contribution to the enlarged European Union.

References

[1] "CARrier-grade delay-aware resource management for wireless multi-hop/Mesh NETworks," <retrieved: April, 2013>. [Online]. Available: http://www.carmnet.eu/

[2] J. C. Climaco, M. M. Pascoal, J. M. Craveirinha, and M. E. V. Captivo, "Internet packet routing: Application of a k-quickest path algorithm," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1045–1054, September 2007

[3] "Olsrd homepage," <retrieved: April, 2013>. [Online]. Available: www.olsr.org/

[4] "Quagga homepage," <retrieved: April, 2013>. [Online]. Available: www.quagga.net/

[5] "Information about Zebra" <retrieved: April, 2013>. [Online]. Available: http://www.gnu.org/software/zebra/

[6] "ZebOS homepage," <retrieved: April, 2013>. [Online]. Available: http://www.ipinfusion.com/about

[7] "Vyatta homepage," <retrieved: April, 2013>. [Online]. Available: http://www.ipinfusion.com/about

[8] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, Oct. 2003, <retrieved: April, 2013>. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[9] "Phosphorus homepage," <retrieved: April, 2013>. [Online]. Available: http://www.ist-phosphorus.eu/

[10] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340 (Proposed Standard), Internet Engineering Task Force, Jul. 2008, <retrieved: April, 2013>. [Online]. Available: http://www.ietf.org/rfc/rfc5340.txt

[11] "Xorp homepage," <retrieved: April, 2013>. [Online]. Available: www.xorp.org/

[12] "Xorp architecture," <retrieved: April, 2013>. [Online]. Available: http://xorp.run.montefiore.ulg.ac.be/latex2wiki/design_overview

[13] "Bird homepage," <retrieved: April, 2013>. [Online]. Available: http://bird.network.cz/

[14] A. Kaliszan, M. Głąbowski, and S. Hanczewski, "A didactic platform for testing and developing routing protocols," in *Proceedings of The Third Advanced International Conference on Telecommunications*, Stuttgart, Germany, May 2012.

[15] A. Tonnesen, "Unik olsrd plugin implementation howto," 2004, <retrieved: April, 2013>. [Online]. Available: http://www.olsr.org/docs/olsrd-plugin-howto.html/

[16] B. Hall, *Beej's guide to network programming using internet sockets*, july 2012, <retrieved: April, 2013>. [Online]. Available: http://beej.us/guide/bgnet/

[17] "Open wrt homepage," <retrieved: April, 2013>. [Online]. Available: https://openwrt.org/

[18] "Build root homepage," <retrieved: April, 2013>. [Online]. Available: http://buildroot.uclibc.org/

 ISBN: 978-1-61208-279-0