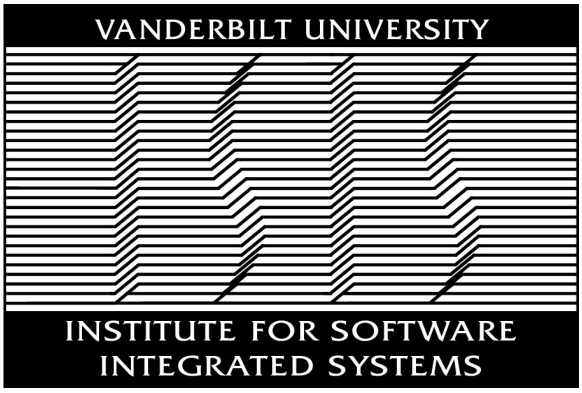

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee 37203

A REPORT ON SIMULATING EXTERNAL APPLICATIONS WITH SOAMANET IN THE LOOP

Himanshu Neema, Abhishek Dubey, Gabor Karsai

TECHNICAL REPORT

ISIS-10-108

Aug, 2010

A REPORT ON SIMULATING EXTERNAL APPLICATIONS WITH SOAMANET IN THE LOOP

Himanshu Neema, Abhishek Dubey, and Gabor Karsai

Institute for Software-Integrated Systems
Vanderbilt University
Nashville, TN 37203, USA

1 Introduction

Service-Oriented Architectures (SOAs) are increasingly being used for designing and building large-scale networked and distributed systems. These systems are realized by dynamically composing a variety of network-available services. SOAMANET [15] provides a tool-suite for evaluation of such systems that involve large-scale SOAs on dynamic network platforms, such as Mobile Ad-hoc Networks (MANETs). For mission critical systems, it is often desired to increase the fidelity of tests to gain increased confidence in the SOA and/or MANET designs. One approach towards this is to use real applications running in a realistic network environment. However, performing field testing of large-scale distributed applications is difficult and expensive, especially when the nodes are mobile and use ad-hoc networking for connectivity. In lab solutions which can provide estimation of how application performance (specifically timeliness of information dissemination and throughput) varies with change in settings such as routing protocols, radio settings, geographical location are helpful. In this report, we describe a novel solution for this problem using a socket-based integration approach that allows running SOAMANET simulator in-loop with the real application.

The simulator runs in real-time and estimates network performance based on the size of the messages sent by real-applications on the physical network. The message recipients hold the received messages in a queue till they receive a confirmation from the simulator. This way, the network delays computed by the simulator translates into an actual delay experienced by the message handlers on the receiving platform. During our experiments we were able to show how the change in bandwidth and increased hop distances between physical nodes affect the application performance. This allows making judicious decisions about network and application settings before deploying them on the field.

This report is organized as follows: Sections 2 and 3 describe the SOAMANET tool-suite. Section 4 describes the architecture of a representative application. Section 5 describes the integration architecture. Section 6 presents an experimental case study. Finally, section 7 concludes the report.

2 SOAMANET Background

With the advances in networking technology and the availability of cheap and efficient mobile hardware, there is a growing trend of the use of networked and distributed mobile devices - which have now become a backbone for various strategic and tactical complex applications. These complex military applications are typically composed of several different and collaborating smaller applications that are accessed as services by authorized entities on an on-demand basis. We call such an approach as Service-Oriented Architectures (SOAs) - an architectural style of building large-scale business and military applications that run on networked and distributed platforms and are constructed by composing various constituent network-available and distributed services. Evaluation of such large-scale SOAs, particularly on dynamic network platforms (such as MANETs) is a non-trivial problem that requires not only a correct modeling of SOAs and the network platforms, but also the relationship between the two.

Most of the tools generally found provide simulation support for either service-oriented architectures (SOA research) [21], [25], [17], [18], [2] or mobile ad-hoc networks (networking research) [20], [28], but not SOA and MANET in combination. Further, [33] and [35] develop strategies to migrate existing applications to MANET environments, [23] and [36] discuss discovery protocols useful in this domain, [27] applies SOAs on MANETs on real network and devices, and [26] provides a preliminary prototype in vehicular ad-hoc networks. In

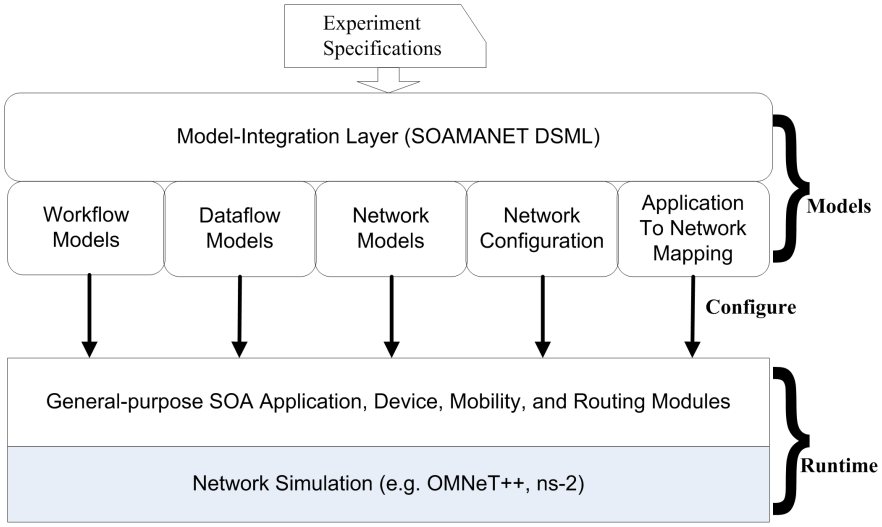

Fig. 1. SOAMANET Architecture.

contrast to current efforts, SOAMANET is a dedicated tool for seamless evaluation of SOA and/or MANET designs in various ways such as complete simulation within a network simulator, or simulation using externally running real applications. It uses Model-Integrated Computing (MIC) technology [30], [13], [32], [31], [10] for building models and synthesizing simulation artifacts.

Figure 1 shows the high-level of architecture of the SOAMANET tool. In the figure, the non-shaded parts represent the key components of the SOAMANET tool, whereas the lightly shaded box represents the external simulation at run time. The experiment specification provides a description of the system to be evaluated. This is used to create a domain-specific SOA modeling language (DSML) [10], [32] that is used to build models for SOA modeling and simulation. These models are used next to automatically configure the simulation using UDM [16] at run-time. The run-time simulation platform consists of general-purpose SOA modules - for applications, devices, mobility patterns, and routing protocols - that support a variety of workflow patterns [19] as well as several dataflow models. These modules are run on an off-the-shelf network simulator that has support for mobile ad-hoc routing such as OMNeT++ [14] or ns-2 [5]. The application to network mapping provides a means for automatically configuring these modules on network nodes.

SOAMANET currently supports OMNeT++ [14] for network modeling and simulation. Using OMNeT++'s flexible architecture, SOAMANET augments it for the development of generic SOA software models for applications, devices, mobility patterns, and routing protocols. The SOA applications are built in C++ in a modular and reusable manner and are currently configured as UDP applications in the OMNeT++ language. SOAMANET uses a library called INETMANET [12] that supplies models for the complete network stack including various MANET routing protocols [3] such as AODV [34], DSR [29], OLSR [24], and DYMO [22], which are also supported by SOAMANET for the SOA application configuration. Currently the terrain (3D) modeling of objects, mobility, and communications is not supported in SOAMANET.

3 SOAMANET Extensions

SOAMANET has been extended in several ways beyond its main application as a simulation tool for SOA over MANETs. This basic mode of SOAMANET's application is represented as Mode 1 in Figure 2. However, with its highly re-usable modeling techniques and analysis capabilities, it can be also used to support a variety of applications. For example, SOAMANET supports running simulations completely external to a network simulator, i.e. completely on real network nodes (Mode 2). Additionally, external/real applications can run externally while using OMNeT++ based network simulator as the network between them. This is accomplished by either establishing socket-based communication between corresponding proxy nodes inside

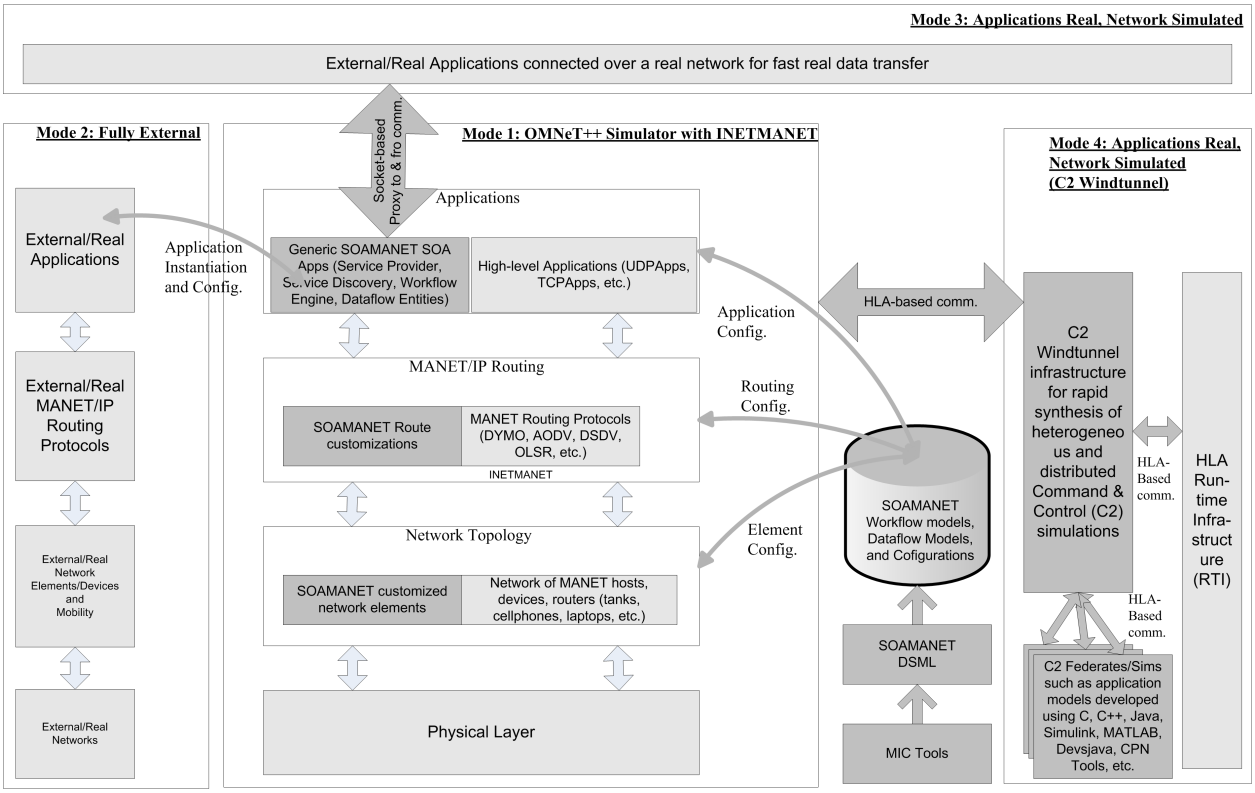

Fig. 2. Various modes of application of the SOAMANET tool.

the network simulator (Mode 3) or via the Command and Control (C2) Wind Tunnel[1] (C2WT) simulation integration framework [1] (Mode 4). SOAMANET's Mode 1 and Mode 4 are highly useful when it becomes practically infeasible to build a realistic network or run real applications due to cost, security, or safety concerns. Additionally, while Mode 2 represents the "most realistic" simulation using all real entities, it often is not practical in the SOAMANET evaluation context, particularly because the network nodes are mobile and dynamic networking platforms such as MANETs must be used. This is because space and cost limitations constrain the use of real mobile devices and also the real hosts such as tanks and aircrafts are usually unavailable for testing purposes. In such cases, Mode 3 and 4 represent good alternatives for providing a safe, secure, confined, and efficient solution, which also supports high-fidelity simulations with node mobility and networking using the SOAMANET simulator. This report focuses on the Mode 3.

4 Architecture of a Representative Application

In this study, we considered a class of applications that use asynchronous event-based communication. Figure 3 illustrates an example application configuration. Each application platform can have software components that either publish or consume the data. The communication between a publisher and a consumer can be implemented either as a unicast connection or a multicast channel. Moreover, both of these transport mechanisms could be implemented over any of the suported MANET protocols such as OLSR [24] or DYMO [22].

Typically, a middleware facilitates the connections between publishers and consumers, freeing them from lower level tasks such as discovering the target node, and marshaling and unmarshalling of the data. For

[1] C2WT [1] is a graphical environment for rapid design and deployment heterogeneous C2 simulation federations. It runs on an HLA [11] based Run-Time Infrastructure (RTI) called Portico [7], and supports rapid integration of federates written in a number of languages/simulators such as C, C++, Java, MATLAB/Simulink [4], DEVSJAVA [9], CPN Tools [8], and OMNeT++ [14].

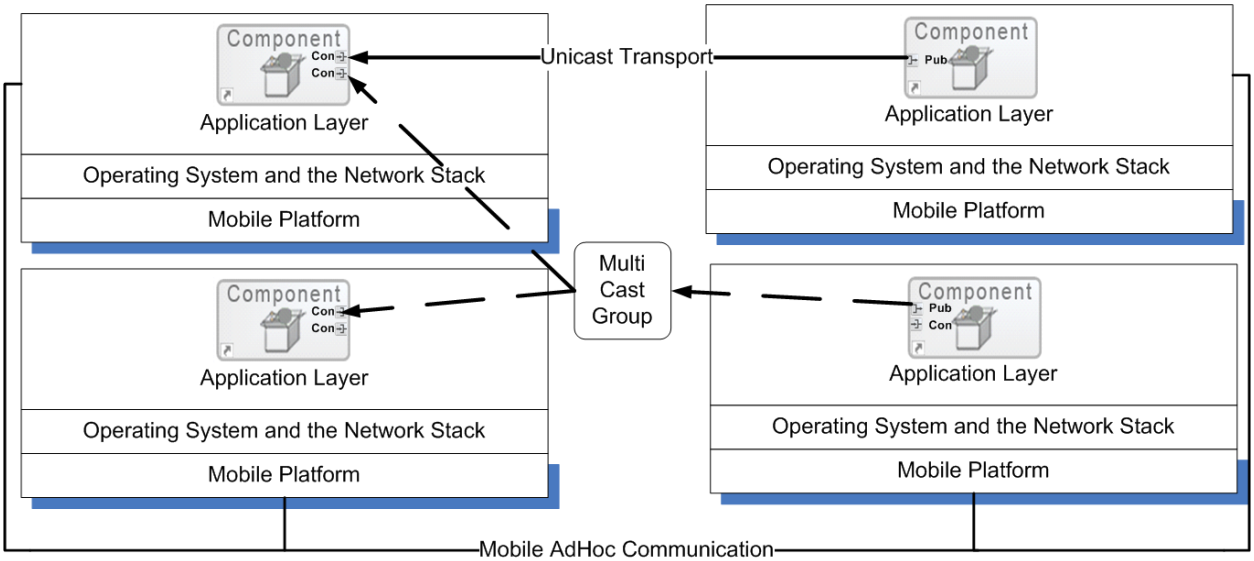

Fig. 3. Representative applications communicate asynchronously over mobile adhoc networks using either multicast or unicast channels.

example, the System of Systems Common Operating Environment (SOSCOE) is a middleware solution from Boeing[2] for connecting command and control (C2) systems to services, legacy systems, and operating system software.

Similarly, Opensplice [6] is a middleware that implements Data Distribution Services (DDS), a standard from the Object Management Group (OMG). DDS supports a publish-subscribe communication mechanism that ensures that the message sender and receiver remain decoupled. Messages are distributed by publishers without knowing who will receive the messages. Each message is associated with a special data type called topic. A subscriber registers to one or more topics of its interest. DDS guarantees that the subscriber will receive messages only from the topics that it had subscribed. A host can act both as a publisher of various topics, and simultaneously as a subscriber of other topics.

5 Integration Architecture

In the context of SOAMANET evaluation, it is often desired to run real applications over a simulated network, particularly when the network is highly dynamic in nature such as in MANETs. As shown in Figure 2, Mode 3 of SOAMANET application allows for such an evaluation method. In particular, here the external applications are run on real devices/computers and they talk to each-other over a simulated network. The communication between external applications and network is exercised using a TCP socket based communication layer that implements a mutually agreeable communication protocol. Although the real messages are sent to the recipient external applications directly over a fast connection (such as a Gigabit Ethernet over a local switch), they are used only when the corresponding messages from the simulated network have arrived.

Figure 4 shows the overall architecture for integrating external applications in SOAMANET. The external environment contains various applications running on several computing nodes/devices connected together through some middleware, e.g. Data Distribution Service (DDS). Each external application is connected to an application proxy within SOAMANET using a TCP socket. External applications are represented by application proxies within the simulation environment. Therefore, they are configured using a representative MANET network topology with a suitable MANET routing protocol, configured at the start of simulation. Application proxies can also be configured to move according to experiment specifications. In this experiment we used DYMO as the MANET routing protocol.

[2] http://www.boeing.com/bds/soscoe/index.html

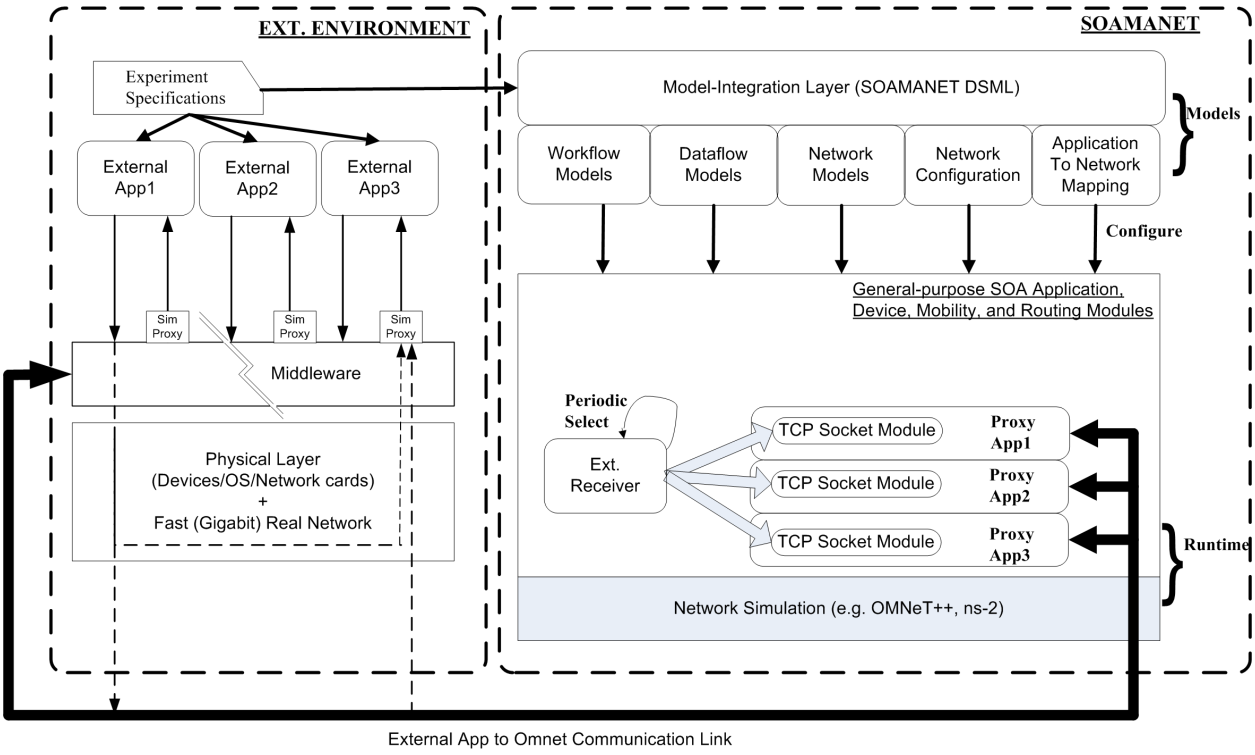

Fig. 4. Integration Architecture for External Applications

Every message sent from an external application to another external application results in the transmission of two messages. The first message is transferred by the middleware to a holding area called Sim Proxy. A Sim Proxy is created for each receiving endpoint of an application. The second message is sent to the application proxy running in SOAMANET using TCP sockets (thick dark arrows between external environment and SOAMANET). The network simulation in SOAMANET is synchronized to run with the real-time by extending the default scheduler of discrete-event network simulator (such as OMNeT++ or ns-2). In this experiment, we used OMNeT++ as the network simulator. Currently, the applications can be configured to run with peer-to-peer messaging (unicast) as well using a multicast network. For multicast networks, the simulation (and application proxies) can be appropriately configured to participate in various multicast groups.

Within SOAMANET, the references to the socket connections held by the application proxies are kept in a separate module called "External Receiver". This module periodically performs *select* operations on these TCP sockets to check for the arrival of messages. The periodicity is set to a very low value (0.005 seconds of simulation time) to minimize undue delays in messaging. At the end of the loop, the list of messages is sent with zero delay (logical) to appropriate application proxies.

When messages traveling in the simulated network arrive at destination application proxies, they are sent immediately to the Sim Proxy via TCP sockets. At that time, the Sim Proxy finds the corresponding real message (matched using a unique identifier) that it had held, and releases the real message to the appropriate external application. The time taken between the receipt of a message at application proxy in SOAMANET and receipt of the corresponding message in the Sim Proxy is equal to $T_1 + T_2 + \Delta$, where T_1 is the time taken to transmit the message from the sender Sim Proxy to the simulator, T_2 is the time taken to transmit the message from the simulator to the receiver Sim Proxy, and Δ is the simulated delay computed in SOAMANET. Typically $T_1, T_2 << \Delta$, due to T_1 and T_2 being transmission times over a fast 1Gbps network.

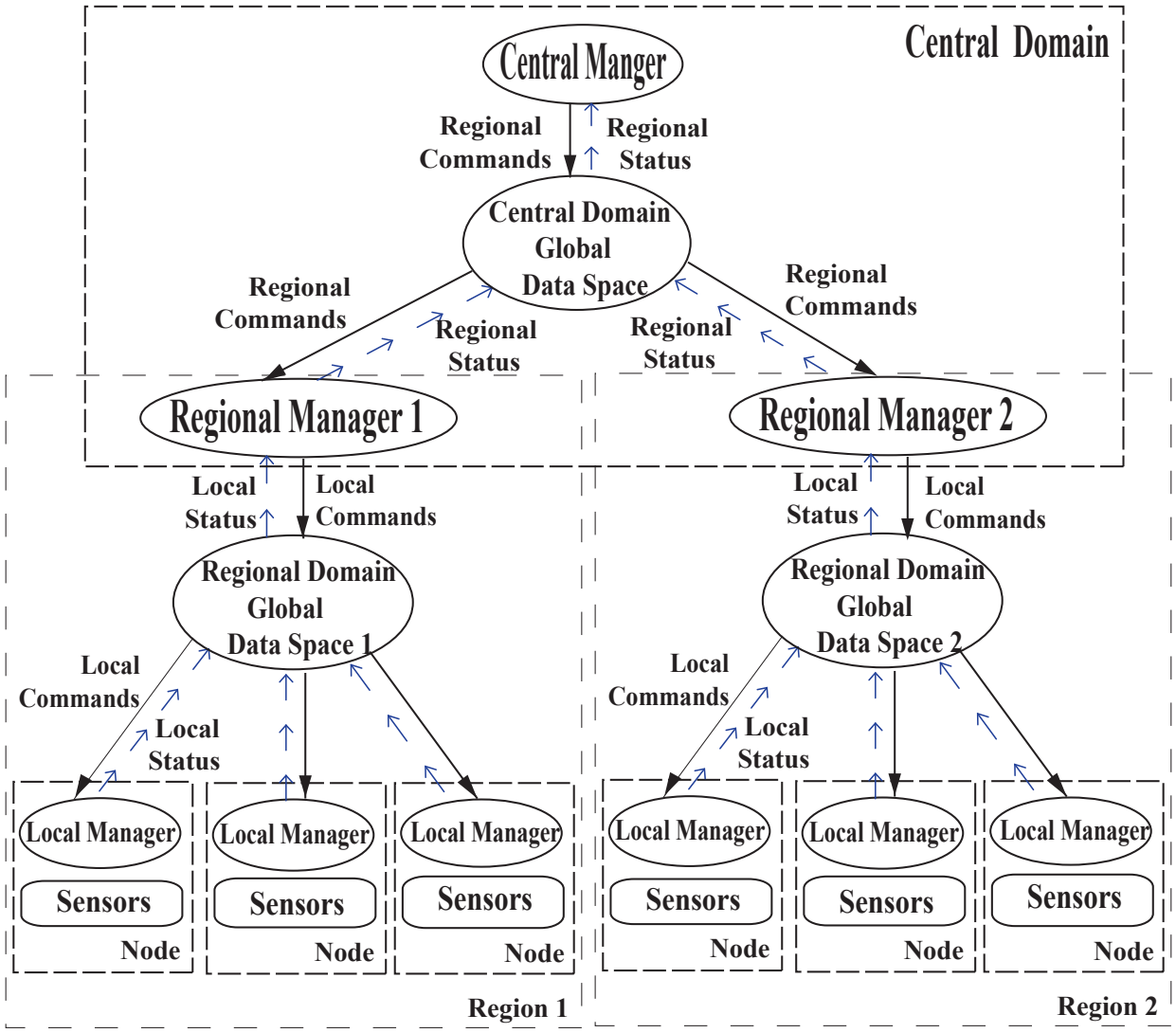

Fig. 5. Data Delivery Channels. The host status information flows from bottom to top direction and the commands flow in the opposite direction.

6 Case Study

We consider a distributed monitoring and control framework required to manage the health of mobile computing platforms. The hierarchical network structure of monitoring is described in Figure 5. A local manager runs on each node to collect and report the local status. It receives control commands from a regional manager to monitor the system health. A regional manager is a bridge between local managers and the central manager. A regional manager belongs to a multicast group. It supervises a set of local managers, collects status reports from local managers, filters out the necessary information, and transfers the information to the central manager (also known as the global manager). It also distributes commands to local managers that are under its supervision. The central manager also collects all status reports and processes them to create a global report.

It is possible to have more than one regional manager in a regional zone. The secondary regional manager only takes over if the primary instance fails. A local manager, running on a local platform, can change its multicast group if it moves out of communication range.

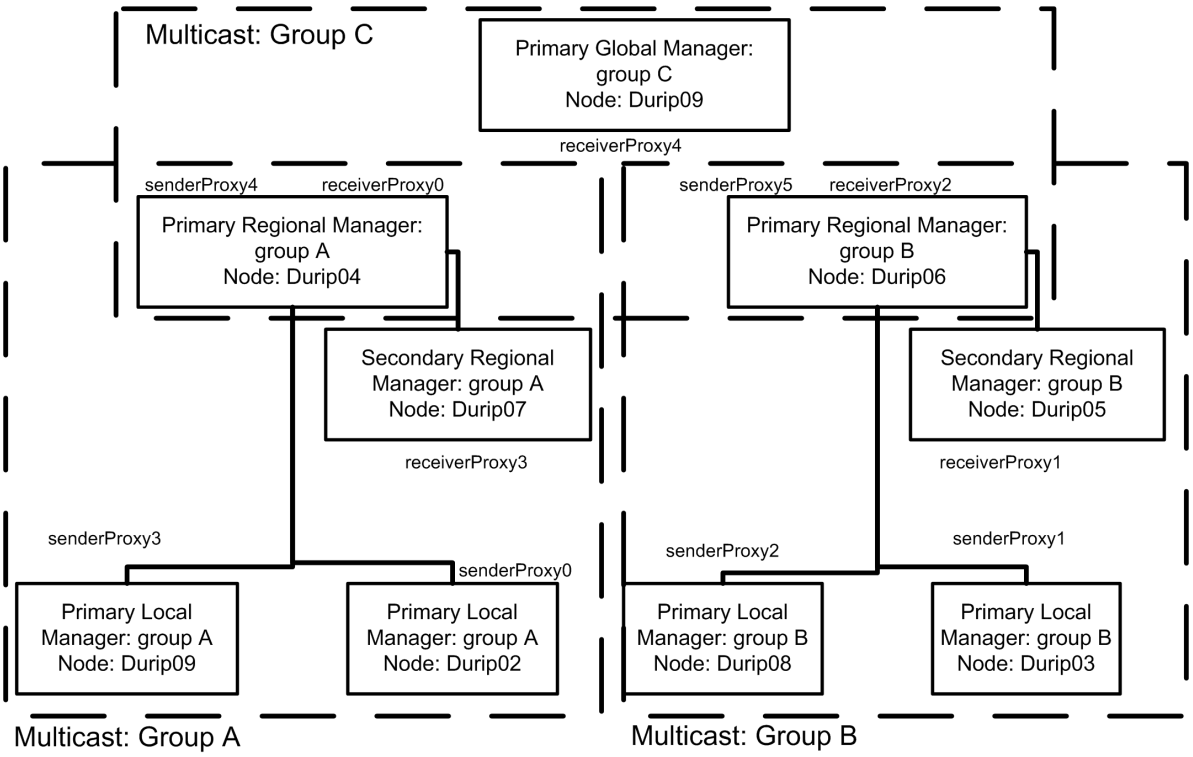

Fig. 6. Experiment setup

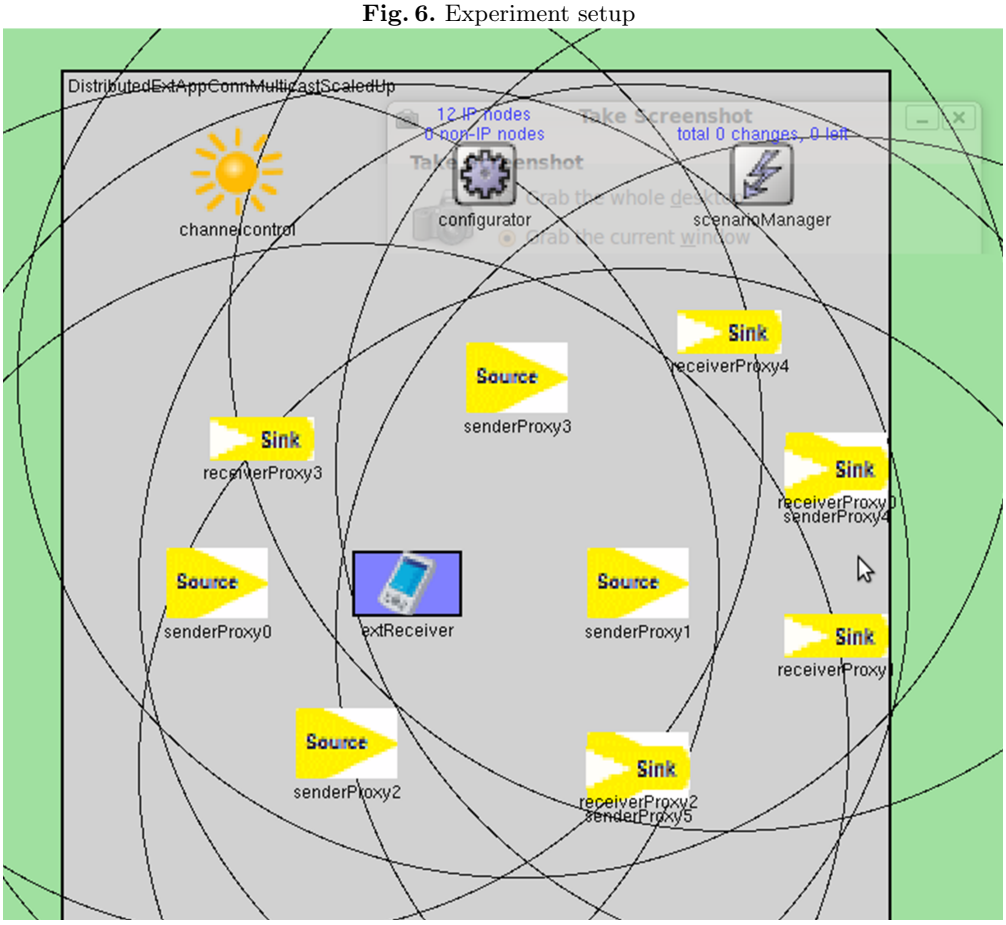

Fig. 7. Geographical location of all nodes.

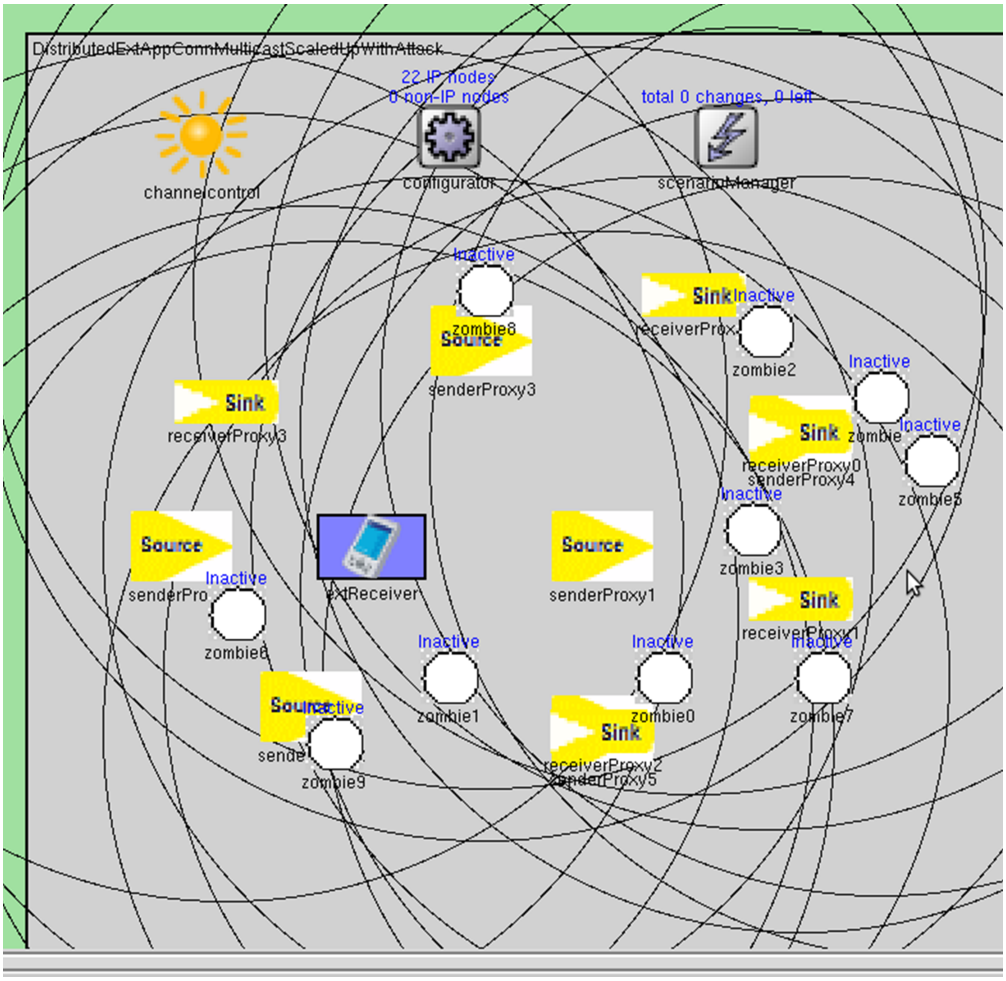

Fig. 8. Geographical location of all nodes and zombie nodes attacking the network.

We instantiated this application in our lab and integrated it with SOMANET as described in previous section and conducted several experiments. The aim of the experiment was to simulate the effect of radio bandwidth and network jamming attacks on the system. The experimental results and setup are described in following subsections.

Experiment Setup As shown in Figure 6, we used a Linux cluster to emulate the mobile platform nodes. All of the nodes were running an instance of an Opensplice daemon that facilitated application communication. Application code was altered to enable hooks such that it can interface with the simulator. As shown in Figure 5, the setup is arranged hierarchically to send data from lowest nodes, via the regional managers, to the global manager. In this experiment, we did not simulate the downward flow of commands from top to bottom.

Table 1 in the appendix shows the configuration file created in OMNET++ for this experiment. The MANET routing protocol used was DYMO. Each application instance shown in Figure 6 requires a corresponding application proxy to be created within the network simulator. While local managers are configured as sender proxies (i.e., they send information to other nodes), regional and global managers were configured as receiver proxies. Two regional managers, running on *durip*04 and *durip*06 were also configured as senders, because they also send information to global manager. Figure 7 describes the simulated geographical position and the radio range for all nodes.

Results: Figure 9 shows the median delay experienced at all receiver nodes for two different experiments, one with the radio bandwidth of 11 Mbps and the other with radio bandwidth of 54Mbps. The results clearly show that the performance degrades significantly with the reduction in the radio bandwidth. Notice that the

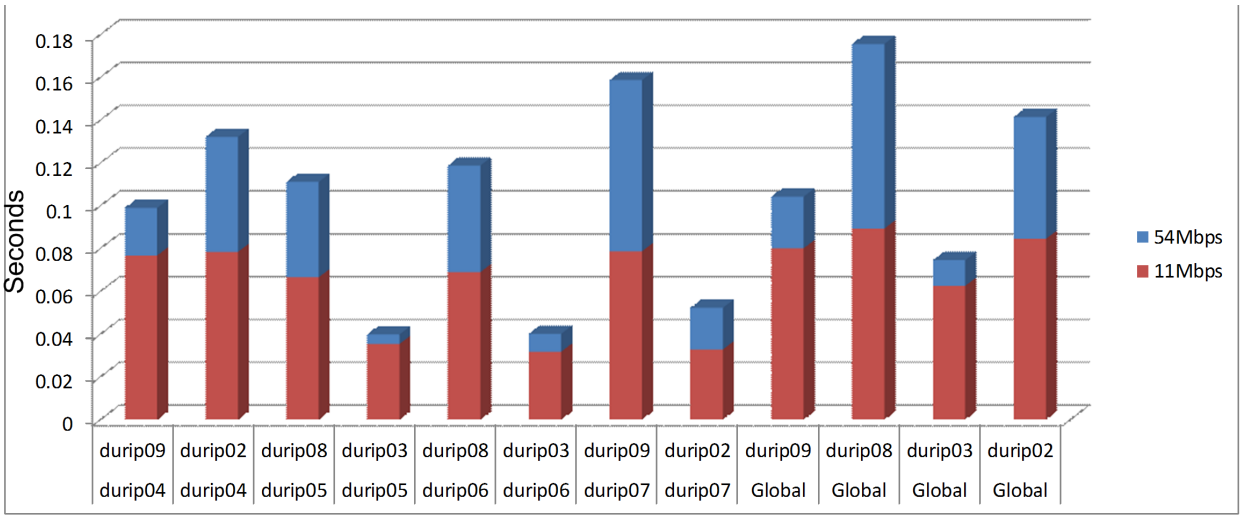

Fig. 9. Median Delays Between Sender and Receivers for two different radio bandwidths. For each tick on xaxis, the bottom label is the receiving node, the upper label is the sender node. Units of Y axis data is seconds.

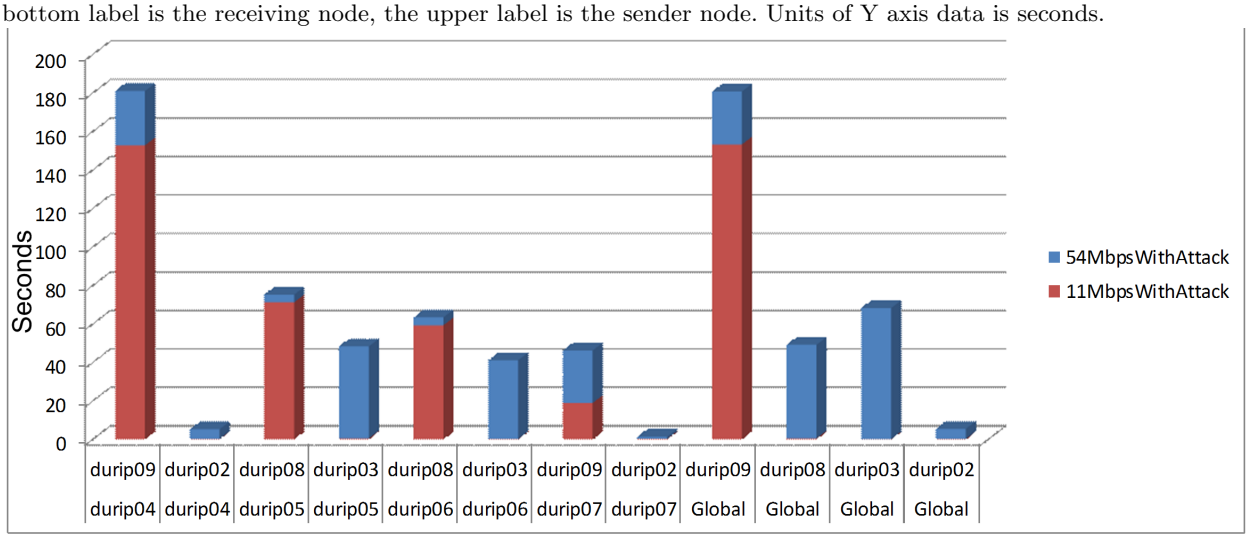

Fig. 10. Median Delays Between Sender and Receivers for two different radio bandwidths with a network attack. For each tick on xaxis, the bottom label is the receiving node, the upper label is the sender node. Units of Y axis data is seconds.

delay between *durip*02 and the global manager is high because of the distance between them. The values on y axis are in seconds. Total experiment time was 300 seconds.

Next, we conducted a simulation experiment in which we instantiated eight zombie nodes to produce the network attack data. Each zombie node flooded the network with random data, which clearly increased the network delay. The geographical locations of nodes and zombies are shown in Figure 8. The results for experiments for both 11 Mbps and 54 Mbps are shown in Figure 10. We noticed that the *number of packets dropped* increased heavily when network was under attack. Moreover, the median delays were substantially larger than in the previous experiment. Total experiment time was 300 seconds.

7 Conclusions

Evaluation of general-purpose service-oriented architectures on mobile ad-hoc networks involves a variety of challenges. The problem is further exacerbated when high fidelity simulations are desired requiring to use

real applications communicating over a simulated network. In this report, we present an extension of the SOAMANET tool that caters to this evaluation requirement. Using model-based techniques, we present a unique capability to rapidly synthesize such evaluation platforms.

We also presented several case studies for the experiments conducted using SOAMANET and showed that with its modeling techniques, analysis capabilities and an intuitive user interface, SOAMANET provides a powerful tool for designing, developing, and analyzing dynamic SOA and/or MANET designs and implementations as well as to connect external applications seamlessly with the network simulation. Moreover, we also highlighted SOAMANET's highly modular and flexible architecture that can be used to extend its application in several other modes for working with external applications.

We are currently working on enhancing SOAMANET's capabilities to support scenarios that include various Command and Control (C2) architectural elements, as well as on applying its generic application modules for in-the-field evaluation of SOAMANET designs - network protocols, anticipated workflows, dataflows, and mobility patterns. In the future, we also expect to extend SOAMANET to support integration with additional simulation engines to increase its functionality in terms of the supported SOAs, network topologies and protocols, and MANET routing protocols.

Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and was accomplished under Cooperative Agreement W911NF-07-2-0020. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation heron. The authors would also like to thank David Claypool, former research engineer at BAE Systems, for his guidance on drafting realistic experiment scenarios.

References

1. The command and control (C2) windtunnel, https://wiki.isis.vanderbilt.edu/OpenC2WT/index.php/Main_Page
2. eb services interoperability organization (ws-i), www.ws-i.org
3. Manet charter from internet engineering task force (IETF), http://www.ietf.org/proceedings/53/179.htm
4. Matlab/simulink, http://www.mathworks.com
5. The network simulator - ns2, http://www.isi.edu/nsnam/ns/
6. Opensplice high-performance DDS, http://www.opensplice.com/
7. Portico RTI, http://www.porticoproject.org
8. CPN Tools, http://wiki.daimi.au.dk/cpntools/cpntools.wiki
9. DEVSJAVA, www.acims.arizona.edu
10. Generic Modeling Environment (GME), http://www.escherinstitute.org/Plone/tools/suites/mic/gme
11. HLA standard: Ieee standard for modeling and simulation (m&s) high-level architecture (hla) framework and rules.
12. INETMANET, http://github.com/inetmanet/inetmanet
13. The ISIS Model Integrated Computing (MIC) Toolsuite, http://www.escherinstitute.org/Plone/tools/suites/mic
14. OMNeT++ Network Simulator, http://www.omnetpp.org/
15. SOAMANET, https://wiki.isis.vanderbilt.edu/SOAMANET/index.php/Main_Page
16. Universal data model (UDM), http://www.escherinstitute.org/Plone/tools/suites/mic/udm/
17. W3c web services architecture working group, http://www.w3.org/2002/ws/arch/
18. World wide web consortium (w3c), www.w3.org
19. van der Aalst, W., Hofstede, A.H.M.T.: Yawl: Yet another workflow language. Information Systems 30, 245–275 (2003)
20. Broch, J., Maltz, D.A., Johnson, D.B., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. pp. 85–97. ACM, New York, NY, USA (1998)

21. Bruni, R., Lafuente, A.L., Montanari, U., Tuosto, E.: Service oriented architectural design. In: TGC'07: Proceedings of the 3rd conference on Trustworthy global computing. pp. 186–203. Springer-Verlag, Berlin, Heidelberg (2008)
22. Chakeres, I., Perkins, C.: DYnamic MANET On-demand (DYMO) routing internet draft, http://datatracker.ietf.org/doc/draft-ietf-manet-dymo/
23. Cho, C., Lee, D.: Survey of service discovery architectures for mobile ad hoc networks. term paper. Department of Computer and Information Science and Engineering (CICE), University of Florida, Fall 5531 (2005)
24. Clausen, T., Dearlove, C., Jacquet, P.: The optimized link state routing (OLSR) protocol version 2 internet-draft, http://datatracker.ietf.org/doc/draft-ietf-manet-dsr/
25. CM, M., KJ, L., F, M., PF, B., R, M.: Oasis standard reference model for service oriented architecture 1.0 (2006), http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
26. Gonçalves, Jo a.F., Esteves, E.F., Rossetti, R.J., Oliveira, E.: Simulating communication in a service-oriented architecture for v2v networks. In: EPIA '09: Proceedings of the 14th Portuguese Conference on Artificial Intelligence. pp. 15–26. Springer-Verlag, Berlin, Heidelberg (2009)
27. Halonen, T., Ojala, T.: Cross-layer design for providing service oriented architecture in a mobile ad hoc network. In: MUM '06: Proceedings of the 5th international conference on Mobile and ubiquitous multimedia. p. 11. ACM, New York, NY, USA (2006)
28. Holland, G., Vaidya, N.: Analysis of tcp performance over mobile ad hoc networks. In: MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking. pp. 219–230. ACM, New York, NY, USA (1999)
29. Johnson, D.B.: Dynamic source routing (DSR) internet-draft, http://datatracker.ietf.org/doc/draft-ietf-manet-dsr/
30. Karsai, G., Ledeczi, A., Neema, S., Sztipanovits, J.: The model-integrated computing toolsuite: Metaprogrammable tools for embedded control system design. In: Proc. of the IEEE Joint Conference CCA, ISIC and CACSD, Munich, Germany. pp. 50–55 (October 2006), http://chess.eecs.berkeley.edu/pubs/285.html
31. Karsai, G., Sztipanovits, J., Lédeczi, Á., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE 91(1), 145–164 (2003)
32. Lédeczi, A., Bakay, A., Maróti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing domain-specific design environments. Computer 34(11), 44–51 (2001)
33. Natchetoi, Y., Wu, H., Zheng, Y.: Service-oriented mobile applications for ad-hoc networks. In: IEEE SCC (2). pp. 405–412 (2008)
34. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing (2003), http://datatracker.ietf.org/doc/draft-ietf-manet-aodv/
35. Suri, N., Marcon, M., Quitadamo, R., Rebeschini, M., Arguedas, M., Stabellini, S., Tortonesi, M., Stefanelli, C.: An adaptive and efficient peer-to-peer service-oriented architecture for manet environments with agile computing. In: Proc. IEEE Network Operations and Management Symp. Workshops NOMS Workshops 2008. pp. 364–371 (2008)
36. Tyan, J., Mahmoud, Q.H.: A comprehensive service discovery solution for mobile ad hoc networks. Mob. Netw. Appl. 10(4), 423–434 (2005)

```
[Config MulticastScaledUp]
# NOTE: MULTICAST SIMULATION ASSUMES ALL ENDPOINT NAMES ARE UNIQUE
# Static parameters
network = DistributedExtAppConnMulticastScaledUp
DistributedExtAppConnMulticast.playgroundSizeX = 650
DistributedExtAppConnMulticast.playgroundSizeY = 550
**.extReceiver.udpApp[*].select_period = 0.0001
# MULTICAST group defining parameters
**.extReceiver.udpApp[*].multicast_group_names_and_members = "GroupA/durip02,durip09,durip04,durip07
 GroupB/durip03,durip06,durip05,durip08 GroupC/durip04_1,durip06_1,durip09_1"
# SenderProxy parameters
*.numSenders = 6
**.senderProxy0.udpApp[*].server_app_node_name = "durip02"
**.senderProxy0.udpApp[*].server_app_hostname = "192.168.1.2"
**.senderProxy0.udpApp[*].server_app_port_number = 30001
**.senderProxy0.udpApp[*].server_app_endpoint_name = "durip02"
**.senderProxy1.udpApp[*].server_app_node_name = "durip03"
**.senderProxy1.udpApp[*].server_app_hostname = "192.168.1.3"
**.senderProxy1.udpApp[*].server_app_port_number = 30001
**.senderProxy1.udpApp[*].server_app_endpoint_name = "durip03"
**.senderProxy2.udpApp[*].server_app_node_name = "durip08"
**.senderProxy2.udpApp[*].server_app_hostname = "192.168.1.8"
**.senderProxy2.udpApp[*].server_app_port_number = 30001
**.senderProxy2.udpApp[*].server_app_endpoint_name = "durip08"
**.senderProxy3.udpApp[*].server_app_node_name = "durip09"
**.senderProxy3.udpApp[*].server_app_hostname = "192.168.1.9"
**.senderProxy3.udpApp[*].server_app_port_number = 30001
**.senderProxy3.udpApp[*].server_app_endpoint_name = "durip09"
**.senderProxy4.udpApp[*].server_app_node_name = "durip04"
**.senderProxy4.udpApp[*].server_app_hostname = "192.168.1.4"
**.senderProxy4.udpApp[*].server_app_port_number = 50001
**.senderProxy4.udpApp[*].server_app_endpoint_name = "durip04_1"
**.senderProxy5.udpApp[*].server_app_node_name = "durip06"
**.senderProxy5.udpApp[*].server_app_hostname = "192.168.1.6"
**.senderProxy5.udpApp[*].server_app_port_number = 50001
**.senderProxy5.udpApp[*].server_app_endpoint_name = "durip06_1"
# ReceiverProxy parameters
*.numReceivers = 5
**.receiverProxy0.udpApp[*].server_app_node_name = "durip04"
**.receiverProxy0.udpApp[*].server_app_hostname = "192.168.1.4"
**.receiverProxy0.udpApp[*].server_app_port_number = 40001
**.receiverProxy0.udpApp[*].server_app_endpoint_name = "durip04"
**.receiverProxy1.udpApp[*].server_app_node_name = "durip05"
**.receiverProxy1.udpApp[*].server_app_hostname = "192.168.1.5"
**.receiverProxy1.udpApp[*].server_app_port_number = 40001
**.receiverProxy1.udpApp[*].server_app_endpoint_name = "durip05"
**.receiverProxy2.udpApp[*].server_app_node_name = "durip06"
**.receiverProxy2.udpApp[*].server_app_hostname = "192.168.1.6"
**.receiverProxy2.udpApp[*].server_app_port_number = 40001
**.receiverProxy2.udpApp[*].server_app_endpoint_name = "durip06"
**.receiverProxy3.udpApp[*].server_app_node_name = "durip07"
**.receiverProxy3.udpApp[*].server_app_hostname = "192.168.1.7"
**.receiverProxy3.udpApp[*].server_app_port_number = 40001
**.receiverProxy3.udpApp[*].server_app_endpoint_name = "durip07"
**.receiverProxy4.udpApp[*].server_app_node_name = "durip09"
**.receiverProxy4.udpApp[*].server_app_hostname = "192.168.1.9"
**.receiverProxy4.udpApp[*].server_app_port_number = 50001
**.receiverProxy4.udpApp[*].server_app_endpoint_name = "durip09_1"
```

Table 1. Omnet Configuration file. Notice that each physical node in the experiment is mapped to a proxy in the omnet simulation.