Large-Scale Integrated Network System Simulation with DEVS-Suite

Ahmet Zengin
Faculty of Technology, Sakarya University
Sakarya, 54187 - Turkey
[e-mail: azengin@sakarya.edu.tr]

Received March 9, 2010; revised May 20, 2010; accepted June 28, 2010; published August 27, 2010

Abstract

Formidable growth of Internet technologies has revealed challenging issues about its scale and performance evaluation. Modeling and simulation play a central role in the evaluation of the behavior and performance of the large-scale network systems. Large numbers of nodes affect simulation performance, simulation execution time and scalability in a weighty manner. Most of the existing simulators have numerous problems such as size, lack of system theoretic approach and complexity of modeled network. In this work, a scalable discrete-event modeling approach is described for studying networks' scalability and performance traits. Key fundamental attributes of Internet and its protocols are incorporated into a set of simulation models developed using the Discrete Event System Specification (DEVS) approach. Large-scale network models are simulated and evaluated to show the benefits of the developed network models and approaches.

Keywords: DEVS formalism, modeling and simulation of networks, routing, scalability

DOI: 10.3837/tiis.2010.08.001

1. Introduction

The primary aim of this work is to design and implement an OSPF simulator for Internet systems based on DEVS-Suite [1] using DEVS [2] concepts for high performance and scalability. In order to evaluate the behavior and performance of the large-scale network systems, modeling and simulation play central role. An important method for design and analysis of such systems and their routing protocols is simulation modeling, especially when analytical methods are known to be inapplicable [3][4]. Basically, network simulators try to model real world network systems and they are often essential since experiments may not be possible with actual computer networks. However, real network systems can be modeled by means of abstraction mechanism which renders possible to model complex systems into the limited resource computers. A network simulator should enable users to represent a network topology, prepare different scenarios, specify the nodes and links, and analyze the results. Graphical user interfaces allow the simulation users to visualize and track working of simulated environment. Simulation tools such as ns-2 [5], ns-3 [6], OPNET [7], SSFNet [8], GloMoSim [9] and OMNeT++ [10] have extensively been used for computer network research. But these tools have some disadvantages such as managing complexity, performance, scalability, visualization and lack of system theoretic design. For example, while ns-2 has a weak visualization, OPNET has a poor scalability even if it has good visualization tools. OMNeT++ is difficult to extend and highly need for specialization and lacks of system theoretic underlying approaches.

In this work, a generic network model composed of nodes and links is first built, later Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) routing protocols are designed and implemented for sake of high performance, modular and hierarchical structured network research tool. The beginning stage was to create a model and simulator software that enables to make decisions on network design. All common components of a packet switched network with atomic node and data link models of various capacity assignments are defined using DEVS formalism [2] with selected level of abstraction. DEVS modeling approach supports hierarchical modular model construction, distributed execution, and therefore offers a basis to characterize complex, large-scale systems with atomic and coupled models [11]. Recently, DEVS formalism has found many application areas such as swarm routing [12] and processor architectures [13].

On definition of basic components, a link state routing protocol OSPF and its behavior were modeled. OSPF routing protocol was chosen due for its suitability for large-scale applications. Currently, in particular for Internet, while distance vector protocols are used for inter-gateway interactions, link state protocols are used for intranet case [14]. OSPF as one of the famous link state routing protocol is an open standards routing protocol and a particularly efficient interior gateway (IGP) routing protocol that is faster than routing information protocol (RIP) which is one of the most known kinds of the distance vector protocols family. It employs the Dijkstra algorithm when estimating the shortest paths [15]. BGP routing protocol is also implemented to support routing between autonomous systems (AS).

To establish verification and validation of the designed tool, the network simulator ns-2 is selected to compare with DEVS-Suite OSPF implementation. In order to show the accuracy of model execution a set of experimental validation tests are performed and results are compared with state of the art simulator ns-2. Scalability, flexibility, portability, design for learning, affordability features of the designed simulator make itself an essential tool for network protocol research.

DEVS-Suite is the discrete event general purpose simulation environment based on DEVS formalism and also is a new version of the DEVSJAVA simulator [16][11] with more visualization and tracking capabilities. DEVS-Suite has a graphic interface called simview which eliminates the need to learn a simulation language while still maintaining a consistent, high performance and user-friendly product. The OSPF models on DEVS-Suite was a result of the application of both networking theory and software engineering principles with particular attention being paid to reliability and maintainability by comparing with ns-2. With the developed OSPF simulator, the student can create any network topology, experimenting on it, tracking the things happened in network, in particular, related to routing and finally can master the network concepts interactively. DEVS-Suite simulator can be run on a personal computer as well as run on online via DEVS-Suite Web Start [17] application which enables e-learning using Java Web Start technology.

The remainder of this paper is organized as follows: Section 2 provides a description of DEVS formalism, DEVS-Suite simulator, OSPF fundamentals, comparison of the network simulators and generators, and information about ns-2 simulator. Section 3 defines the developed OSPF simulation framework giving details of each components and summarizes the OSPF protocol implementation. Section 4 includes validation experiments with ns-2 comparison, covers performance results and evaluation of the large-scale experiments. Section 5 presents the conclusions from this research.

2. Background and Related Work

2.1 DEVS Formalism

To use modeling and simulation as a problem solving technique, there is need for a modeling formalism. As a formal system definition, formalism renders possible to create virtual worlds in our limited computation frameworks and tools. Limitations of the computational environments demand new high performance modeling formalisms and approaches. Large-scale network systems exhibit very high level complex, dynamic and parallel characteristics. Therefore, complex and distributed behaviors of the large-scale systems make network modeling effort difficult. However, discrete event modeling formalisms bringing abstraction and simplification mechanisms to modeling and simulation discipline facilitate modeling and simulation study of the systems such as computer networks demonstrating complex, dynamic, distributed and unpredicted behavior. The dynamics of network systems can be described using discrete event modeling. This is because the dynamics of network systems can be characterized in terms of components that can process and generate events. Among discrete event modeling approaches, the Discrete Event Systems Specification (DEVS) [2] is well suited for formally describing concurrent processing and the event-driven nature of arbitrary configuration of nodes and links forming network systems. This modeling approach supports hierarchical modular model construction, distributed execution, and therefore characterizing complex, large-scale systems with atomic and coupled models. Atomic models represent the structure and behavior of individual components via inputs (X), outputs (Y), states (S), and functions. Parallel DEVS, which extends the classical DEVS, is capable of processing multiple input events and concurrent occurrences of internal and external transition functions. Parallel DEVS atomic model supports local control on the handling of simultaneous internal and external events. A parallel atomic model can be described with

$$Parallel\ atomic\ model = (X, S, Y, \delta ext, \delta int, \delta conf, \lambda, ta). \quad (1)$$

The external (δext), internal (δint), confluent (δconf), output (λ), and time advance functions (ta) define a component's behavior over time. Internal and external transition functions describe autonomous behavior and response to external stimuli, respectively. The time advance function represents the passage of time. The output function is used to generate outputs. The Parallel DEVS confluent transition function provides local control by handling simultaneous internal and external transition functions.

Atomic models can be coupled together in a strict hierarchy to form more complex models. A coupled model can be constructed by composing models into hierarchical tree structures. A coupled model is defined in terms of its constituent atomic and/or coupled models. A DEVS coupled model specifies the connection of systems coupled together and interaction with each other. Given atomic models, DEVS coupled models are formed in a straightforward manner. Two major activities involved in coupled models are specifying its component models, and defining the coupling that represents desired interactions. A Parallel DEVS coupled model [18] is formalized as:

$$Coupled\ model = < X,Y,D,Mi,Ii,Zi,j > \quad (2)$$

In this symbolic representation, X is a set of input values and Y is a set of output values. D is a set of the DEVS components and Mi is a DEVS component model. Ii is the set of influencees for I and Zi,j is the i-to-j output translation function.

2.2 DEVS-Suite Simulation Environment

DEVS formalism can be executed using simulation engines such as DEVS-Suite [1] and DEVSJAVA [16]. DEVS-Suite and DEVSJAVA are object oriented realization of Parallel DEVS and its associated simulators. They support describing complex structures and behaviors of network systems using object-oriented modeling techniques and advanced features of the Java programming language. The formal foundation of DEVS, its efficient execution, and the availability of sequential, parallel, or distributed simulation engines using alternative computational environments such as CORBA, HLA, and Web-services are important considerations. Furthermore, the DEVS models are extended with other kinds of models such as fuzzy logic [19].

DEVS-Suite is an open source, discrete event, and general-purpose simulation environment [1]. It is a new generation extended from the DEVSJAVA simulator and DEVS Tracking Environment. The main modules of the DEVS-Suite are simview [16], DEVS tracking Environment [20], and timeview [1]. DEVS-Suite can simulate models specified using the DEVS formalism [2]. The architecture of the DEVS-Suite simulator environment is Model Facade View Control (MFVC) [20] by which simulation data can be displayed with its animation and viewing of time trajectories generated by the parallel DEVS abstract simulator. Soft synchronization among timeviews and animation is supported based on the simulator's logical (or real-time) execution speed [21].

In DEVS-Suite, execution of the models can be animated in terms of the input/output messages for coupled models and the state changes for the atomic models. Every atomic and coupled model component can have its own time-based trajectories and log files for inputs and outputs as well as the common phase and sigma state variables for atomic models. The simulation experiments can be triggered with test inputs — predefined inputs for every model can be selected from its dialogue box at the beginning of the simulation. At the end of the simulation, user-defined statistical metrics for any of the model components can be obtained with the so-called transducer models.

One of the main advantages of the simulator is an option window that provides selection between visualization and other capabilities. When loading the model, modeler can be toggle between simview and tracking options. Simview supports component views of DEVS source code in the form of block components. Visualization decreases the performance of a software system due to the heavy use of limited hardware resources [21]. Users can choose the visualization and/or tracking capability and if needed, none of them, for large-scale simulations. The DEVS-Suite and Ptolemy II [22] show similar performance when no visualization is used [23]. In comparison, they outperform the simEvents simulator [24].

2.3 OSPF Fundamentals and Control Packets

In this study, the OSPF routing protocol is modeled due for large-scale characteristics of it. Routing protocols in the network systems can be split into two main categories: link state routing and distance vector routing. Currently, in particular for Internet, while distance vector protocols are used for inter-gateway interactions, link state protocols are used for intranet case [14]. Open Shortest Path First (OSPF) as one of the famous link state routing protocol is an open standard routing protocol and a particularly efficient interior gateway (IGP) routing protocol that is faster than routing information protocol (RIP) which is one of the most known kinds of the distance vector protocols family. It employs the Dijkstra algorithm when estimating the shortest paths [15].

The OSPF routing protocol was developed to provide an alternative to RIP, based on Shortest Path First algorithms instead of the Bellman-Ford algorithm which is the basis for RIP. It uses a tree that describes the network topology to define the shortest path from each router to each destination address. In many places, RIP is still used in current Internet TCP/IP networks that have not been upgraded to OSPF. It is also used on OSPF networks as an end-station-to-router protocol. OSPF addresses all the deficiencies of RIP, without affecting connectivity to RIP based networks. Fast growing and large-scale networks must be designed properly if the capabilities of OSPF are to be fully exploited. Because of its ability to handle variable networking masks, OSPF also helps to reduce waste of today’s precious IP addresses. OSPF will enable networks to scale to very large topologies, while maintaining high levels of availability and performance. The main difference between OSPF and RIP is that RIP only keeps track of the closest router for each destination address, while OSPF keeps track of a complete topological database of all connections in the local network.

The OSPF algorithm works on three phases. They are important for describing how the algorithm behaves in discrete event fashion. In the startup phase, as soon as a router connects to the network, it sends Hello packets to all of its neighbors, receives their Hello packets in return, and establishes routing connections by synchronizing databases with neighbor routers that agree to synchronize. In the update phase, each router sends an update message at regular intervals. This message is called “link state” describing its routing database to all the other routers, so that all routers have the same description for the local network topology. Finally, in the shortest path tree phase, each router then estimates a mathematical data structure called a ”shortest path tree” that describes the shortest path to every destination address indicating the closest router for communication.

2.4 Large-scale Network Simulation Tools and Topology Generators

The basic limitations of analytical approaches have led to a variety of modeling and simulation approaches and tools. Tools such as ns-2 [5], ns-3 [6], OPNET [7], OMNeT++ [10], GloMoSim [9] and SSFNet [8] are used to reveal the inner workings of computer networks in virtual settings. A key emphasis has been on enabling design and testing of routing algorithms, MAC layers, and end-to-end queuing. Although the capabilities of these simulation tools support describing (wired and wireless) computer and device network protocols and communications in great detail, their underlying foundation lacks support for developing models in system theoretic manner. The conceptual models of these tools are derived from computer network hardware and software abstractions. These models are mostly implemented in object-oriented programming languages and simulated in virtual and/or emulated in physical testbeds. But these tools have some disadvantages in terms of underlying methodology, implementation and scalability [4]. These reasons are limited or lack object-oriented concepts for designing and implementing simulators. They lack support for casting real world entities to non-object simulation components. Furthermore, scalable and efficient execution requires support for not only concurrent simulation execution methods but also simple, yet sound modeling concepts. That is despite tools such as pdns [25] and SSFNet [8] being implemented in object oriented programming languages, they lack formal modeling theories. Furthermore, other tools such as OPNET is inherently not well suited for parallel and distributed execution. Since ns-2 requires expertise in C++, Tcl and also software engineering skills, the visual modeling support provided by OPNET is considered superior for educational purposes [26]. Other concerns about these simulators are limited or weak support for visualization as well as difficulties in tool installation and ease of use. Detailed comparisons of the network simulators are presented in [27] and [28]. Summary of the network simulators comparison is presented in **Table 1**.

Table 1. A comparison of network simulators.

Aspect	Ns-2	pdns	OPNET	OMNeT++	J-Sim	SSFNet	GloMoSim	DEVS-Suite
Object-orientation	medium	medium	strong	medium	very strong	very strong	medium	very strong
Models Library	strong	strong	strong	strong	medium	weak	medium	weak
Analysis	medium	medium	very strong	weak	weak	weak	strong	very strong
Extendibility	medium	medium	strong	very strong	very strong	very strong	very strong	very strong
Expertise need	very strong	very strong	weak	strong	weak	strong	weak	medium
Deployment	weak	weak	strong	medium	very strong	strong	strong	very strong
Documentation	medium	medium	very strong	strong	weak	weak	medium	medium
Availability	very strong	strong	weak	very strong	very strong	very strong	weak	very strong
Visualization	weak	weak	strong	strong	medium	very strong	very strong	very strong
User base	very strong	weak	strong	strong	medium	weak	weak	weak
Scalability	weak	very strong	medium	medium	strong	very strong	very strong	very strong
Performance	strong	very strong	medium	medium	strong	very strong	medium	very strong
Randomness	very strong	very strong	weak	strong	weak	weak	weak	very strong
Failure modeling	very strong	very strong	very strong	medium	weak	medium	weak	very strong
Web access	no	no	no	no	strong	no	no	very strong

2.5 BRITE Topology Generator

In order to study on large-scale network systems, its topologies and protocols, numerous simulation tools are developed and implemented. In **Table 1**, several state of the art simulators used in network research are listed and compared. Some of these tools have own automatic software components by which network topology and traffic models are generated. Network topology and traffic generators are employed in generation of realistic structured networks for simulation purposes and performance characteristics [28]. In large-scale simulation experiments, topology generation is important due for following reasons: (1) to build the topology of the large real networks manually is impossible, (2) in order to develop algorithms for Internet, there is need for Internet-scale models and (3) topology generation allows for efficient planning and long-term network design [29].

There are great deals of researches on reviewing the topology generators. Most of these tools are well summarized in [30]. From these models, The Boston University Representative Internet Topology Generator (BRITE) is adopted. BRITE topology generator is developed based on the AS power laws and incorporates with skewed network placement and locality in network connections [31]. BRITE is Java based universal topology generator (i.e. it is possible to generate topologies that can be processed by widely used simulators such as ns-2, OMNeT++ and SSF) and supports many topology models such as Waxman and Barabasi. In our implementation, BRITE interface is integrated with DEVS-Suite simulation package together with its visualization tool [32].

2.6 The Network Simulator 2

The network simulator ns-2 [5] is developed based on REAL network simulator project. It is designed for research for local and wide-area network simulations and network education. Ns-2 is an object-oriented, open source, discrete event network simulator, which is written in C++ and uses OTcl as a command and configuration interface [33]. It is based on a seven-layer network synthesis and designed as packet-based, which means that all packet interactions are in focus during simulation. It implements network transmission protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as DropTail, RED and CBQ, routing algorithms such as Dijkstra [15], and other algorithms [5]. Network simulator 2 provides an important support for modeling and simulation of TCP, routing, and multicast protocols over wired and wireless networks and is primarily useful for simulating local and wide area networks. Although ns-2 is fairly easy to use once you get to know the simulator, it is quite difficult for a first-time user, because there are few user-friendly manuals and it is difficult to install. Various extensions of parallel and distributed variations are developed to achieve execution scalability (e.g., pdns [25]).

Many researches including design, test and comparison of new network algorithms, protocols, and technologies are done with ns-2. Some deficiencies of ns-2 include limited support for visualization and complex simulator design [34]. Since ns-2 is dependent on different technologies, it can be very difficult to make changes to the existing models. Furthermore, from the modeling methodology vantage point, ns-2 can be considered as domain-specific simulator which is intimately tied to the computer network concepts [35].

3. Framework for Large-Scale Network Simulation

3.1 DEVS Model Specifications

In this section, basic model components of the developed large-scale simulation framework are elaborated by giving their DEVS specifications. Detailed information of the developed simulator can be found in [36]. As already mentioned in background section, DEVS formalism is selected for modeling such a distributed system since DEVS enables the modeler to specify systems in system theoretic manner and yields hierarchical and modular developments. These properties of DEVS facilitate to model distributed systems with smart components and overcome the complexity. Since a typical networked system can only be characterized as nodes and links, it is started to develop network model by specifying these basic components using parallel DEVS atomic model.

The conceptual model of developed OSPF network system together with its experimental frame is presented in **Fig. 1**. As shown in the Fig., a typical node running with DEVS kernel has a routing table in its routing module and runs OSPF protocol. Since routing module reflects whole intelligence in the network, main effort was given to develop it and it is also most detailed part of the node model. In the routing module, OSPF version 2 and Border Gateway Protocol (BGP) are implemented [37]. OSPF protocol supports intranet routing and BGP protocol supports internet routing. Routing module has several protocol stacks such as LSA history which stores versions of the link state advertisements and topology databases which stores whole network information. These stacks support protocol management and organization. Nodes also have network interface cards for every neighbor. A typical interface has a queue for incoming and outgoing packets which is simply a drop-tail queue as detailed in the next sections. Very simple MAC protocol is implemented, since main concern was to test routing protocols on large-scale experimental conditions. Nodes can originate control packets such as hello, LSA and acknowledgement packets. Nodes are modeled as parallel DEVS atomic models and its model description can be seen in **Fig. 2** and state diagrams of the nodes and links are depicted in **Fig. 3**.

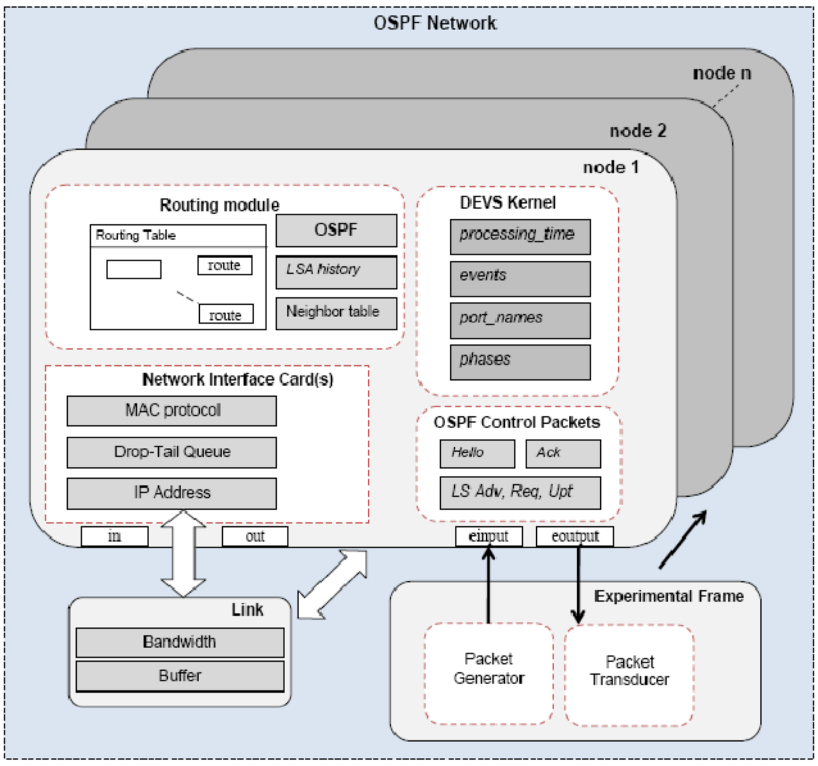


Fig. 1. Conceptual OSPF Network Model

M_{ospf_node} = < X , Y , S, δext, δint, δcon, λ, ta >

where

// Input ports and values

X = inport × invalues

invalues : {packet, packet_DATA, packet_HELLO, packet_LSA },

inports : {NIC1_in, inEvent} // one network interface is assumed

// Output ports and values

Y = outport × outvalues

outvalues : {packet, packet_DATA, packet_HELLO, packet_LSA },

outports : {NIC1_out, outEvent },

// State sets

S= phase × σ × Q

*phase :{**"idle", "startup", "queuing", "congested", "flooding", "addingNbor", "newLSAadded", "gettingRoute", "subNetting", "forwardingLSA"**}*

$\sigma = \Re^{+}_{0,\infty}$

Q = Qqueue × Qneighbour_table × Qtopology_database

where

Qqueue is a queue for incoming and outgoing packets,

Qneighbour_table is a data structure to store neighbour's name and link states and, Qtopology_database is to store topology information obtained by LSA packets.

// External transition function

$$\delta ext\,((phase, \sigma, Q), e, X)) = \begin{cases} if\ packet = LSA,\ its\ source = address\ and\ it\ is\ included\ in\ LSA\ history, \\ \textbf{discard the packet} \\ else\ if\ packet = DATA\ and\ queueu\ has\ enough\ space\ for\ the\ packet, \\ \textbf{enqueue the packet and } \boldsymbol{s} \leftarrow (\text{"}\textbf{queuing}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ else\ \boldsymbol{s} \leftarrow (\text{"}congested\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ where\ s \in S \end{cases}$$

// Internal transition function

$$\delta int\,(phase, \sigma, Q) = \begin{cases} if\ queue = \emptyset,\ \textbf{dequeue packet and } \boldsymbol{s} \leftarrow (\text{"}\textbf{idle}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ if\ packet \neq\ null\ and\ packet = HELLO, \\ \textbf{add neighbour and } \boldsymbol{s} \leftarrow (\text{"}\textbf{addingNbor}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ else\ if\ neighbour\ table\ size = number\ of\ NIC's,\ \boldsymbol{s} \leftarrow (\text{"}\boldsymbol{flooding}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ else\ if\ packet = LSA, \\ \textbf{add to topology database and } \boldsymbol{s} \leftarrow (\text{"}\textbf{newLSAadded}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ else\ if\ packet = DATA\ and\ its\ destination \neq address, \\ \boldsymbol{s} \leftarrow (\text{"}\boldsymbol{gettingRoute}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ else\ if\ packet\ is\ DATA\ and\ its\ destination = address\ \boldsymbol{s} \leftarrow (\text{"}\boldsymbol{subnetting}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ where\ s \in S \end{cases}$$

// Confluent transition function

δcon ((phase, σ, Q), e, X)) = δext (δint (phase, σ, Q), 0, X))

// Output function

$$\lambda(phase, \sigma, Q) = \begin{cases} when\ phase = \text{"}flooding\text{"},\ \textbf{create LSA packet and output } \boldsymbol{y} \leftarrow (\boldsymbol{NIC1\_out}, \boldsymbol{LSA}) \\ when\ phase = \text{"}startup\text{"}, \textbf{create HELLO packet and output } \boldsymbol{y} \leftarrow (\boldsymbol{NIC1\_out}, \boldsymbol{hello}) \\ when\ phase = \text{"}\ newLSAadded\text{"},\ \boldsymbol{y} \leftarrow (\boldsymbol{NIC1_{out}}, \boldsymbol{LSA})\ \textbf{and} \\ \boldsymbol{s} \leftarrow (\text{"}\boldsymbol{forwardingLSA}\text{"}, \boldsymbol{\sigma'}, \boldsymbol{x}) \\ when\ phase = \text{"}gettingRoute\text{"},\ \boldsymbol{get\ outgoing\ interface}, \boldsymbol{NIC\ and} \\ \boldsymbol{y} \leftarrow (\boldsymbol{NIC\_out}, \boldsymbol{packet}) \\ when\ phase = \text{"}\textbf{subNetting}\text{"},\ \boldsymbol{y} \leftarrow (\boldsymbol{outEvent}, \boldsymbol{packet}) \\ where\ y\ \in Y \end{cases}$$

//time advance function

ta(s) = σ

Fig. 2. OSPF Node Model Specification

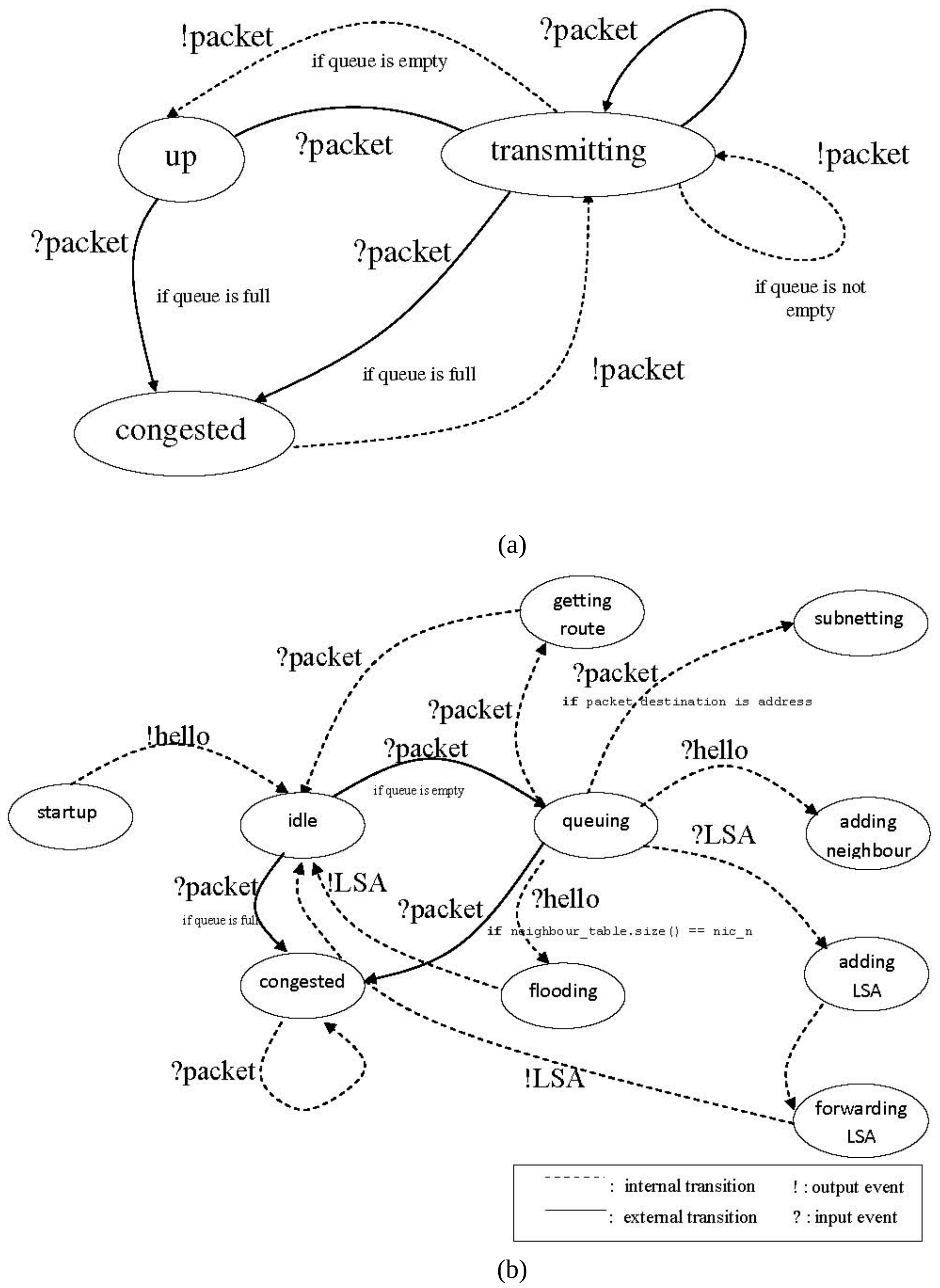


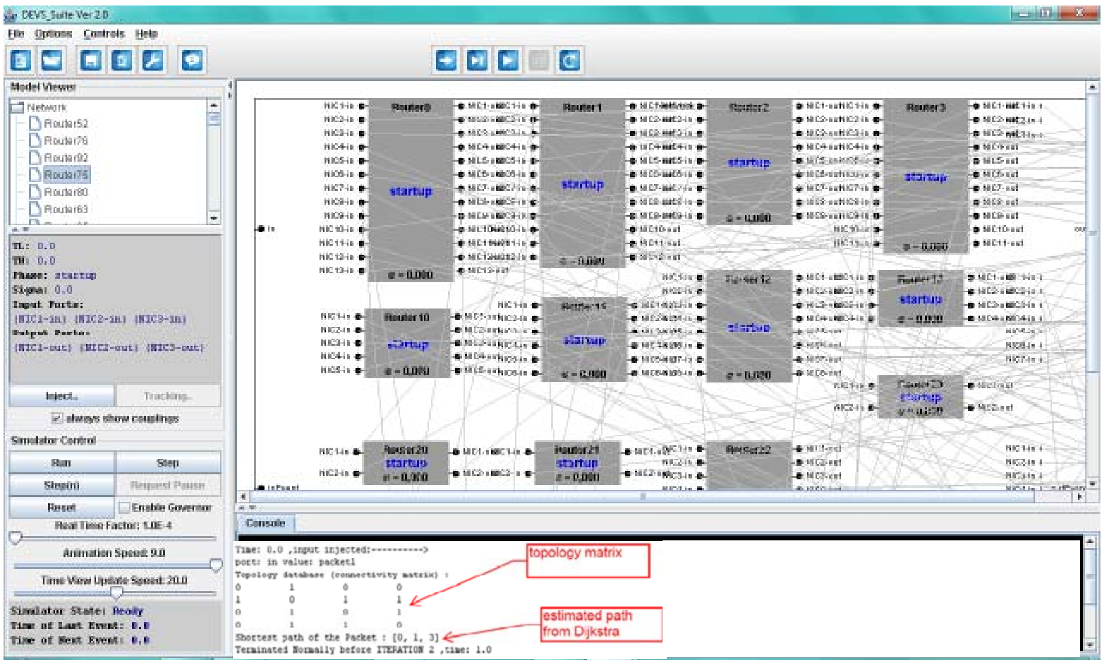
Fig. 3. Node and Link Models State Diagrams.

Fig. 3 (a) depicts a link's state chart in which states are only chanced via internal and external transitions. As seen in the **Fig. 3 (b)**, an OSPF node's behavior is abstracted to ten states to mimic protocol behavior. For example, when a node is in "idle" state and if it receives a packet and its queue is full, then state changes to "congested".

Generic nodes can be connected via duplex links allowing the traffic to flow in both directions (see **Fig. 1**). The link atomic models represent the communication channels and are characterized with bandwidth (bits/sec) and transmission delay specified in milliseconds. Each link has a corresponding buffer with finite capacity. The packets that arrive are placed in the buffer and are transmitted to the next node using mixed version of first-in first-out (FIFO) and priority queue management strategy, i.e. control packets have more precedence than data packets. Links are able to carry traffic of a certain bandwidth up to the total capacity of the link.

By connecting nodes and links, a coupled model of the network with OSPF routing capability is developed in DEVS-Suite. Developed DEVS-Suite OSPF framework provides visualization, advanced tracking capability, reusability and component-based model design. **Fig. 4** shows DEVS-Suite modeling framework and OSPF model. In the top left side of the DEVS-Suite simulator, a model viewer lists all models with coupled models and bottom left side has simulation controls. Bottom of the window is simulation console output to track messages from code.

Due to a node designed as a generic, with little changes, it can be implemented to other network elements/devices such as hub, switch, gate, a border node etc. To determine the behavior of a node, two parameters are used: (1) process speed which directly influences processing time of a node, and (2) queue in which incoming and outgoing packets are stored. In **Fig. 4**, router models can be seen with its name, state and next event time visually.



```
TL: 0.0
TN: 0.0
Phase: startup
Sigma: 0.0
Input Ports:
{NIC1-in} {NIC2-in} {NIC3-in}
Output Ports:
{NIC1-out} {NIC2-out} {NIC3-out}
```

```
Simulator State: Ready
Time of Last Event: 0.0
Time of Next Event: 0.0
```

```
Time: 0.0 ,input injected:---------->
port: in value: packet1
Topology database (connectivity matrix) :
0         1         0         0
1         0         1         1
0         1         0         1
0         1         1         0
Shortest path of the Packet : [0, 1, 3]
Terminated Normally before ITERATION 2 ,time: 1.0
```

Fig. 4. DEVS-Suite Simulation Viewer with Large-Scale Network Model

3.2 OSPF Implementation

After modeling basic components of the network system such as nodes and links, OSPF behavior is implemented to give network routing capability and intelligence. As already mentioned in Section 2, OSPF protocol is selected since current Internet systems work with it due for its scalability properties. Routing protocols in the network systems can be split into two main categories: link state routing and distance vector routing. Currently, in particular for Internet, while distance vector protocols are used for inter-gateway interactions, link state protocols are used for intranet case [14]. Open Shortest Path First (OSPF) as one of the famous link state routing protocol is an open standards routing protocol and a particularly efficient interior gateway (IGP) routing protocol that is faster than routing information protocol (RIP) which is most known member of the distance vector protocols family. It employs the Dijkstra algorithm when estimating the shortest paths [15]. In the next sections, DEVS OSPF modeling phases are summarized.

1) Network discovery phase: The main idea behind discovering the set of routers in the network domain is to repeatedly find the neighboring routers from the currently known routers, until no new routers are discovered. OSPF forms adjacencies between neighboring routers operating in the same Autonomous System (AS). OSPF neighboring routers use the Hello protocol to discover each other. Hello protocol packets are sent periodically to establish and maintain neighbor relationships between OSPF neighbors.

The Hello protocol is used by OSPF to verify the eligibility of potential OSPF neighbors, and to confirm the correct area ID, timer values, and OSPF priority. The Hello protocol also maintains adjacencies after neighbors are up by multicasting Hello packets every 10 seconds.

2) Flooding phase: On receiving Hello messages, a node updates its neighbor table and sends link state advertisement packets to all neighbors. Routers have a mechanism for flooding link state advertisements. If a topological change occurs in the network (due to changes in link status or routes), an immediate update is sent from that neighbor, alerting other OSPF-speaking routers to the change. Only the route prefix that is affected by the change is modified, and only that LSA is sent with the updated change. In our model, DEVS entities are used to carry link states. Flooding procedure starts when a link state update occurs (event triggered) or it can start per 30 seconds. Since flooding adds significantly overhead to the simulation, event triggered approach is selected instead of fixed time updates. Each router then updates its database with the change, and network convergence occurs. If incoming link state advertisement of a particular link is fewer versions from that in stored topology database, the LSA packet should be dropped from the queue without further processing.

The database of the Autonomous System (AS) topology in developed OSPF framework is represented both neighbor table and topology database. The routing table storing path information for all possible destinations in the AS is calculated by supplying these databases to Dijkstra class of the router model. Dijkstra class then generates topology matrix and shortest path tree and therefore data is routed.

Topology database stores LSA packets from all other routers and neighbor table stores neighbor id and associated interface names. Neighbor table and topology databases implemented as Linked Lists while routing table stores route objects.

3) Shortest path calculation phase: Computation of shortest paths is important phase in routing data packets. There are many approaches to calculate shortest paths depending on the type of network and problem specification. Most known and implemented algorithm is Dijkstra [15]. According to the Dijkstra logic, a router calculates the shortest-path tree considering itself on top of the hierarchy.

In DEVS-Suite OSPF framework, a network coupled model's link state database is represented as a directed graph (see **Fig. 4**). The graphs' vertices are routers or other networks. Each link has an associated link state advertisement. The formation of the shortest path tree is done using the Dijkstra algorithm, a tree is formed from this set of the link state database called topology database.

4) Data message forwarding phase: As already mentioned, packets that are exchanged among components in the form of DEVS messages can be distinguished as data and control packets. Data packets are basic IP packets which carry information such as id and precedence. Control packets allow the node to obtain whole network view and to measure the traffic.

OSPF protocol message exchange is depicted in **Fig. 5**. Communication is started with a hello packet and completed with an acknowledgement packet which means that message is received. The OSPF packet types are as follows:

- Hello packet: These packets are sent periodically or event triggered on all interfaces. Neighborhood is formed by exchanging of Hello packets. Neighbor table is populated with Hello packet information.
- Link State Advertisement (LSA): Every LSA packet contains description of routing domain. In other words, LSA packets carry neighbor information about the originating router. LSA packets have a version field to avoid duplication of the ones stored in topology database. The implemented topology database is composed of every router’s link state advertisements packets.
- Link State Request (LSR): LSR packet is used to request the neighbors’ database that is new version.
- Link State Update (LSU): LSU packets implement the flooding of link state advertisements. Every LSU carries a collection of link state advertisements one hop further from its origin.
- Acknowledgement: A message is depicting receiving of the packet.
- Data: An IP packet contains information.

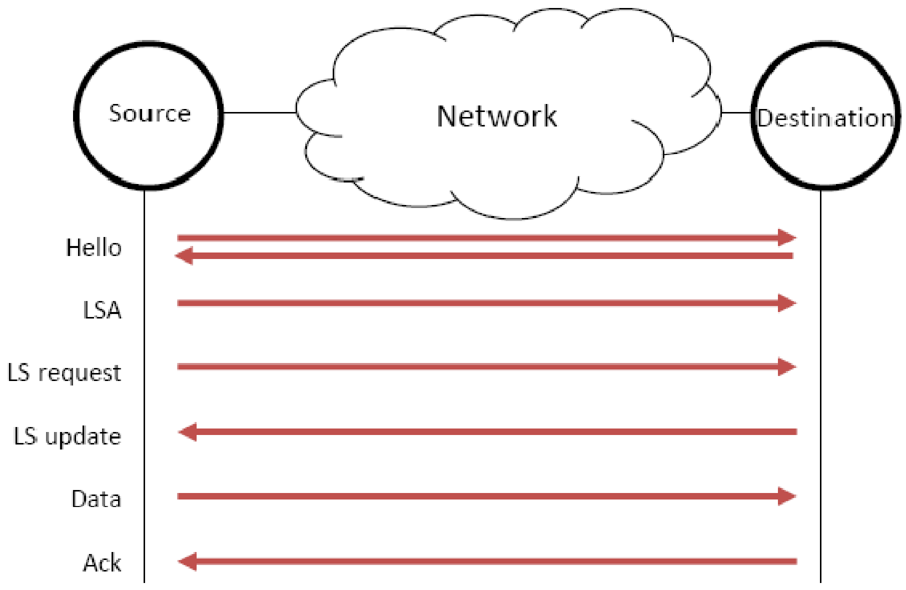


Fig. 5. OSPF Protocol Messaging

5) Autonomous systems (AS): It is important to model hierarchical layers of today’s Internet. A network can be decomposed to domains and clusters. As already mentioned before, network model is formed by coupling nodes via links and is coupled model. Coupled model can be used in a higher level hierarchy as an atomic model. This is called closure under coupling by which increasingly larger networks can be systematically developed and experimented with. OSPF protocol also allows collections of networks and hosts to be coupled together. OSPF coupled model is called Autonomous System (AS) as a single SPF domain that this approach reduces routing traffic and amount as compared to treating the entire network. In this implementation, all routers in the AS have not an identical topological database. A router named border router has a different topological database for every AS connecting to it.

6) Queuing model: Queue is modeled as a Linked List data structure which maintains the FIFO (First In First Out) order of the objects inserted into it. Some new objects can be inserted and others removed, but the orders of the objects in both states are maintained. Packets have different priorities which means insertion order of newly incoming packets is determined by priority value (0 lowest, 7 highest). The queue implements a very simple drop-tail queue with the following behavior.

- If a packet's priority value is highest value, it is inserted at the front of the queue.
- If a packet is to be enqueued and is bigger than the maximum queue size, it is rejected.
- If a packet to be enqueued is smaller than the maximum queue size, but there is not enough space for it, the packets at the end of the queue are dropped until enough space available.
- If there is enough space for a packet to be enqueued, it is inserted at the end of the queue.

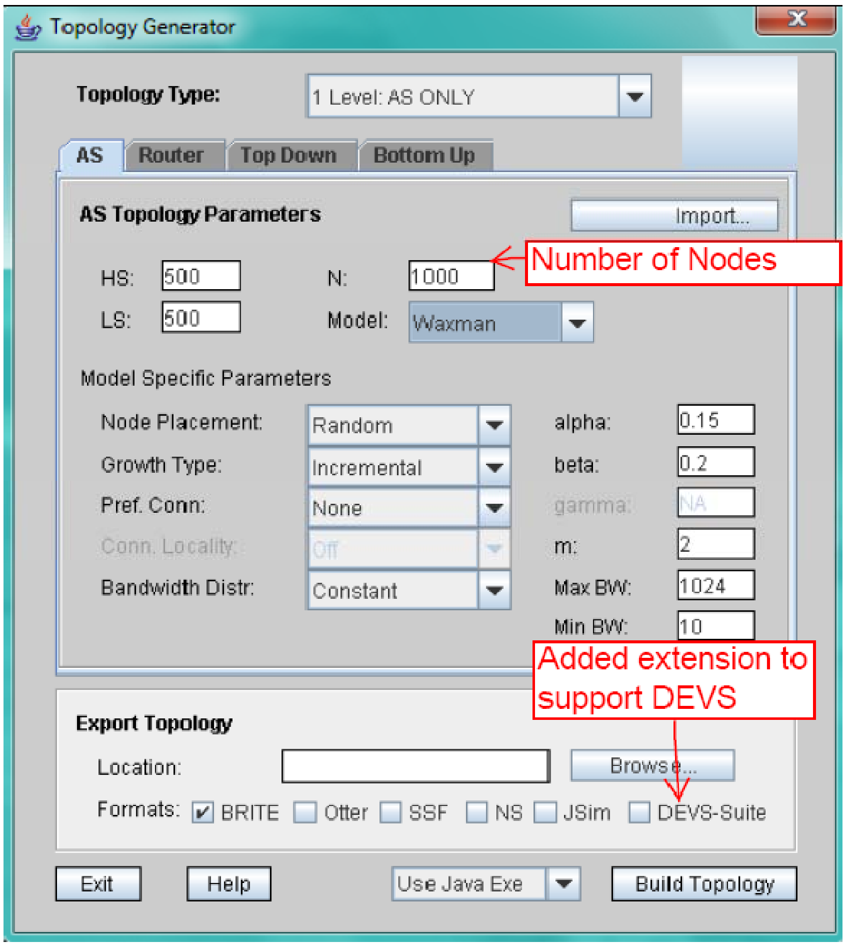
Packets can be discarded upon arriving at a node because of lack of queue space or expired time to live which limits hop count. In addition, when a packet traverses across a link, if there is no available bandwidth on the link, the packet is lost or dropped.

7) Traffic model: In order to experiment with network model, it is necessary to model user traffic and behavior. To make realizations of network traffic and examine specific scenarios, the experimental frame concept and its DEVS-Suite realization are employed. An experimental frame model is separate system from the network model and it is collection of specifications of the conditions by which model is experiment with it [2]. In our implementation, a typical experimental frame consists of an event generator and event transducer. The generator can generate packets with fixed time intervals by randomly choosing source and destination addresses. Uniformly random traffic generation is modeled but exponential and constant bit rate traffic patterns can be implemented easily using Java infrastructure. Generator also creates and schedules specific events in the network such as link down and node congestions so that to program errors. The transducer observes and analyzes the network outputs, and stores these results in trace files. Transducer simply converts data to information which is meaningful for modeler. Together with DEVS-Suite tracking capability, exact trace is applied on developed model.

3.3 DEVS-Suite Topology Generator

Large-scale models over tens to hundreds routers can only be created via tools named topology generator. BRITE [30] topology generator is extended for supporting DEVS coupled models (see **Fig. 6**). Open source and no longer supported BRITE topology generator is developed using both C++ and Java object-oriented languages and allows modelers to import from and export to specific topology files such as ns-2, OMNeT++, JavaSim and SSFNet. Added extension includes an export capability from Brite format to Java source file which is formed DEVS coupling model specifications. As can be seen in **Fig. 6**, BRITE provides to create multiple generation models including flat AS, flat Router and hierarchical topologies. Models can be conFig.d using GUI by changing links attributes such as bandwidth, delay and topography for wireless models.

Though DEVS-Suite has a visualizing tool for its models (see **Fig. 4**), it is important to demonstrate nodes as circles and links with lines. To do so, BRITE-visualizer [31] is implemented as seen in the Fig. 7.



Topology Generator

Topology Type: 1 Level: AS ONLY

AS | Router | Top Down | Bottom Up

AS Topology Parameters — Import...

HS: 500 N: 1000
LS: 500 Model: Waxman

Model Specific Parameters

Node Placement: Random — alpha: 0.15
Growth Type: Incremental — beta: 0.2
Pref. Conn: None — gamma: NA
Conn. Locality: Off — m: 2
Bandwidth Distr: Constant — Max BW: 1024
Min BW: 10

Export Topology

Location:

Formats: [✓ BRITE] Otter SSF NS JSim DEVS-Suite

Fig. 6. BRITE Topology Generator Extended for DEVS-Suite

4. Large-scale Simulation Experiments and Simulation Results

4.1 Validation

Model validation constitutes a very important step in network simulator development. For validation of the developed OSPF model, outcomes from ns-2 simulator are used instead of real network data. To do this, first it is designed to experiment on a small scale topology as seen in Fig. 8 (a). There are four routers numbered from 1 to 4 and four links. Ns-2 is a widely used simulator by network community and it is validated. In the experiments, same models with same configurations are employed to highlight the key structural differences between modeling DEVS-Suite and ns-2. In Fig. 8 (b), a nam animator screenshot of the sample network is shown. Simulation experiments were performed both in ns-2 version 2.32 and DEVS-Suite version 2.0. All simulation configurations and parameters are uniformly selected for comparison purposes. Of course, it is impossible to have the same exact conditions for DEVS-Suite and ns-2 since they have different levels of detail. Selected parameters are listed in **Table 2**. According to the configuration parameters listed in **Table 2**, simulation experiments with both ns-2 and DEVS-Suite simulators are performed for ten seconds. The throughput results as a function of time are shown in **Fig. 9**. As depicted in **Fig. 9**, throughput curves converge to nearly the same average values, 822.4 KB/sec. for ns-2 and 821.9 KB/sec. for DEVS-Suite after a stabilization phase time (2 seconds). Measured throughputs are almost same and the difference comes from selected abstractions and assumptions. It is also observed that the routing tables for the four router nodes are consistent – the creation of tables is validated step by step. It is observed that routing tables are theoretically correct and no packet is discarded due for lack of any route.

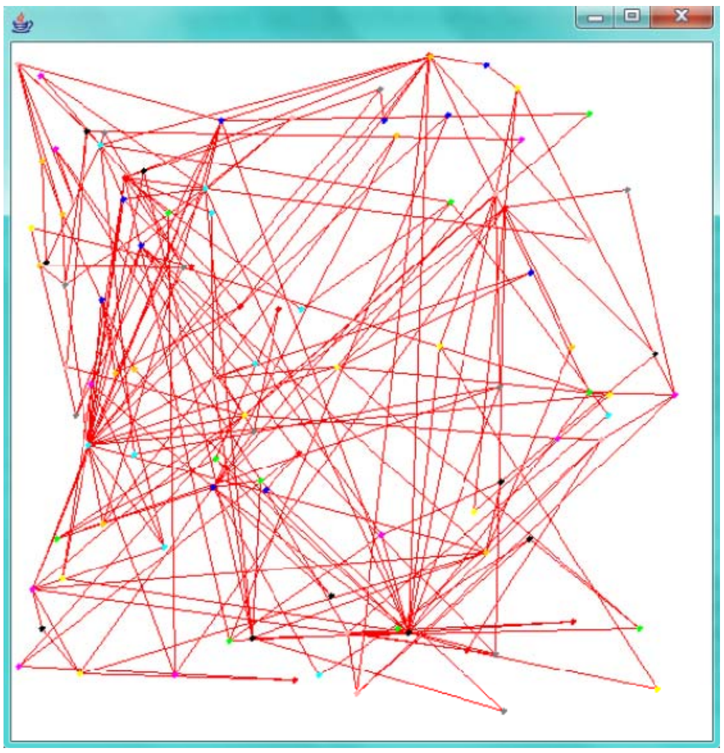
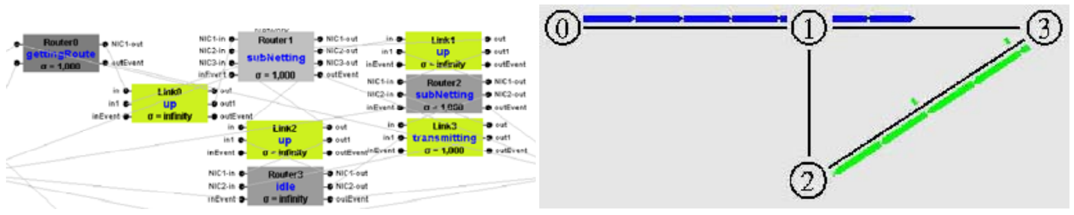


Fig. 7. BRITE Topology Visualizer Showing 100 Nodes of Created Topology

4.2 Performance evaluation

In this study, a series of simulation experiments are done using the developed model as described in previous sections in order to investigate performance of the simulator in large-scale models and Internet. Experiments are conducted in Core 2 Duo machine running at 2.1 GHz with 4 GB RAM and Ubuntu 9.10 64 bit operating system. Performed experiments can be categorized into two phase: (1) simulator performance experiments and (2) large-scale model experiments. In the first case, several simulator experiments are done to measure DEVS-Suite simulator with developed OSPF model. Later, results are compared to well-known network simulator such as ns-2 and JSim.

On the other hand, large-scale DEVS coupled models up to thousands nodes are generated using BRITE topology generator, integrated to DEVS-Suite simulator environment and then measured the simulator outcomes. In the following, these results are given in detail to show developed model's and simulator's performance. Over ten thousands nodes, Java Virtual machine is reported out of memory error, therefore experiments were done within that scale.



(a) (b)

Fig. 8. DEVS-Suite ad NAM Screenshot of The Simple Network Consisting of Router Nodes and Duplex Links

Table 2. Simulation Model Parameters of Ns-2 and DEVS-Suite

Simulation Model Parameters		
	DEVS-Suite	ns-2
Packet sizes	552 bytes	
Link bandwidth	2 Mbps	
Link delay	1 msec.	
Simulation time	10 sec.	
Traffic type	Uniformly random	FTP over TCP
Queue Type	FIFO-Priority	Drop-Tail
Queue Limit	200 KB	20 Packets
Protocol	OSPF	Link State(LS)
Processing speed	1 msec./event	N/A

In **Table 3**, lines of code needed by DEVS-Suite and ns-2 simulators are listed. Lines of code are measured from validation experiments as detailed before. When modeling and simulating four nodes network (see **Fig. 8**), ns-2 requires 84 lines of Tcl code and DEVS-Suite requires relatively less pure Java codes in 68. When using automated tools in DEVS-Suite such as BRITE topology generator, model development can be done without typing any code in automatic manner. Model can be built up on only one class named Simulator in ns-2; however DEVS-Suite models are in two class files: one for network model and other for experimental frame. DEVS-Suite OSPF model and ns-2 generated almost same simulation behavior (see **Fig. 9**). When modeling OSPF behavior, DEVS-Suite generated a thousand events and ns-2 generated approximately thirty-times events more than DEVS. This is important for performance due for every event allocates a significant space on memory and event context changing is expensive. The difference between event resolutions results in large-scale performance.

Table 3. Lines of code needed of ns-2 and DEVS-Suite

	Ns-2	DEVS-Suite	DEVS-Suite + topology generator
Lines of code	84	68	-
Number of classes	1	2	2
Number of events (events/sec)	28388	1000	1000

DEVS-Suite OSPF memory footprint is also compared to the memory demands of several simulators after building the model but before advancing simulation time. These extreme points give us some sense of how much of the memory is being used to describe the architecture. By comparison, Nicol [38] reports that ns-2 demands 52.2 KB per connection, and JavaSim demands 17.3 KB per connection. **Table 4** summarizes the per-connection core costs of all the simulators. While DEVS-Suite OSPF model with simview visualization consumes 39.5 KB per connection, model without visualization consumes 31.5 KB memory per connection. According to these values, Java implementation of DEVS shows better memory management than ns-2 and consumes two times more memory cells than JavaSim. Most of the memory demands come from routing table implementation and later visualization. In our implementation, data structures such as Vector and Hashmap are exploited for storing routing information. Together with more preferable routing table representations such as Nlx-vector approach [39], it is possible to lessen memory demands. The difference between results from with and without DEVS visualization is surprisingly low. The reason is that DEVS-Suite creates most GUI elements even if visualization is not selected.

Table 4. Per-connection core memory demands of the simulators [38].

Tools	Per connection Memory Footprint
JavaSim	17,3Kb
Ns-2	52,2Kb
DEVS-Suite(with simview)	39,5Kb
DEVS-Suite(without simview)	31,6Kb

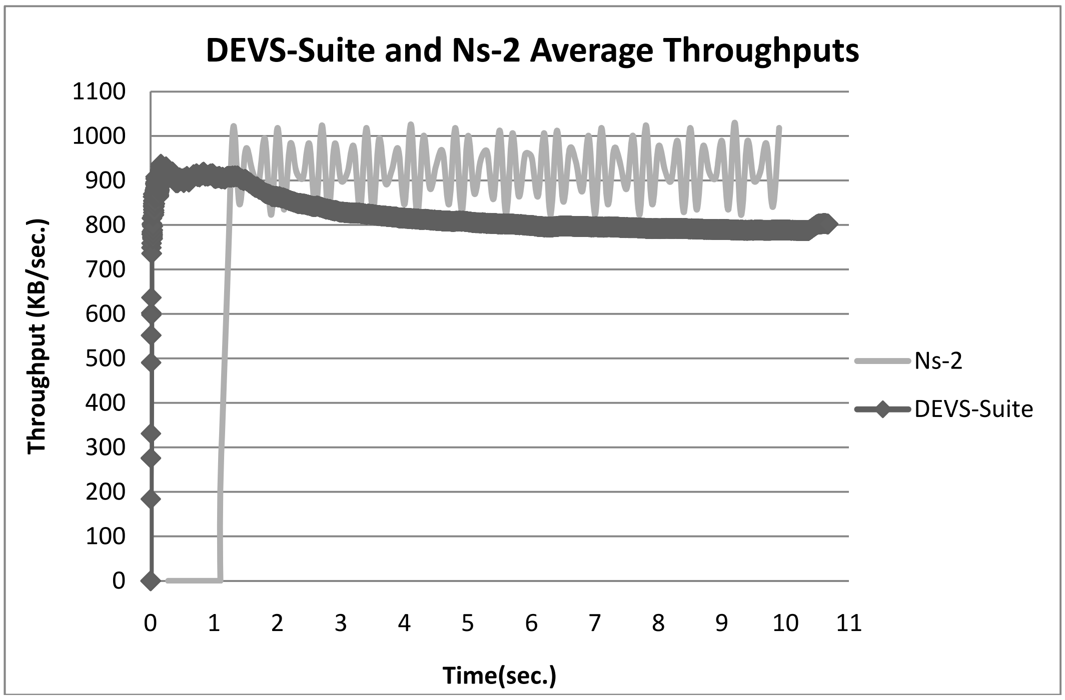


Fig. 9. Simulated Performance Measurements of Ns-2 and DEVS-Suite for Throughput

As already mentioned above, primary goal of the DEVS-Suite OSPF simulator is to achieve a high-speed simulation for large-scale network systems. **Fig. 10** depicts the wall clock execution times of the developed simulator across number of nodes. The Fig. clearly shows the performance superiority of the DEVS-Suite. When network scales up to ten thousands nodes, execution time increases almost linearly (660 seconds for 10.000 nodes).

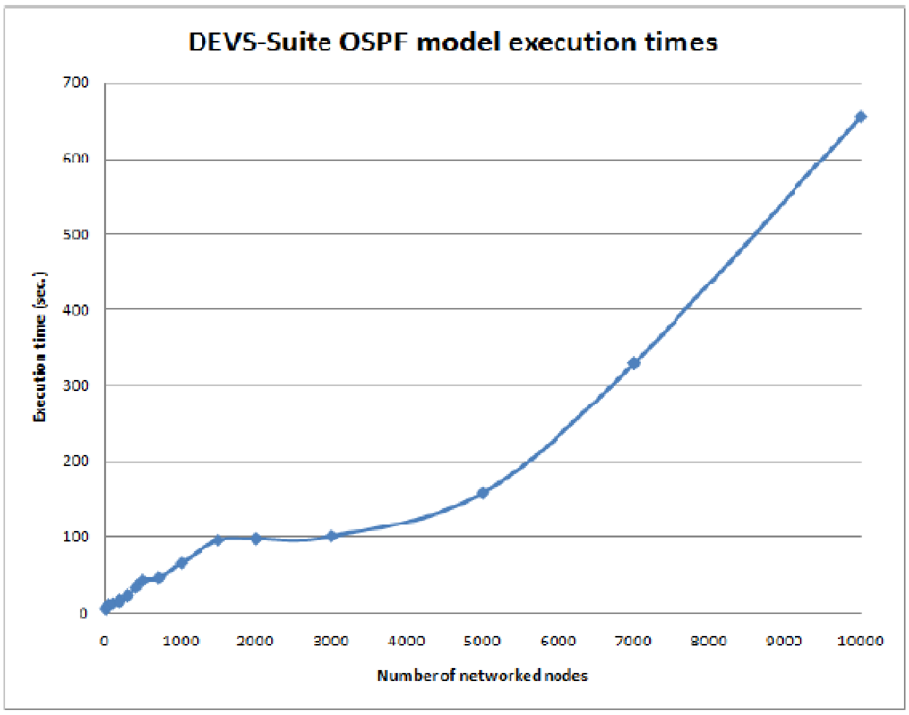


Fig. 10. Execution Times of The DEVS-Suite OSPF Simulator

4.3 Scalability evaluation

Table 5 shows the performance results for all synthetic topology scenarios across varying topology parameters. Models composed of varying number of nodes and autonomous systems are generated using BRITE topology generator. All networks are modeled as Waxman topology model [40] and connectivity parameters (links per nodes - m) are set to 2 but selected as 1 in some models as shown in the table.

First of all, convergence value of the models are measured and listed. Convergence time is a simulation set up time that all routers install their routing databases. Low convergence means speedy routing process and it is desired for large-scale approaches. For all models' convergence time vary linearly with the number of nodes. It has also linear relation with connectivity (m value). As shown in the **Table 5**, convergence is still acceptable even if scaled up to ten thousands nodes.

Besides convergence value, efficiency is measured for all network models. Efficiency can be formulated as a ratio of packets delivered successfully. For all configurations, it is obvious that the developed simulator runs with an extremely high degree of efficiency, which is estimated as packet delivery ratio and lowest efficiency is 99.547% that means 99.547 percent of the total packets are delivered to their destinations safely.

Throughputs and turnaround times as main network performance criteria are also observed from experiments. These values allow evaluating the network performance. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. In the large-scale models, as average bandwidth consumption increases, throughput inherently gives small value, for example 2.3 Kbps for largest network size. Average delay was measured in seconds for large networks, for example, it exceeds to minutes for large-scale networks while it is less than ten seconds for small-scale networks (see **Table 5**). In the table, logical simulation time is also listed in simulation cycles or steps.

From the experiments, one important outcome is the effect of connectivity on network performance (see **Fig. 11**). For example, in 1000 nodes scenarios, several different models are built in terms of number of autonomous systems and number of nodes per autonomous system. When the number of autonomous system increases, convergence is decreasing and performance, efficiency and, turnaround time are also increasing (see **Fig. 11**). Not for the first two, increasing of turnaround time is not desired. When connectivity is set to 1 (m=1), network routing speed is increased approximately 4.5 times. This is because OSPF runs with links states and less number of links means less amount of database but high delay.

Table 5. Performance Results of Synthetic Topology Networks

Number of nodes	*Number of AS's*	*Number of nodes per AS*	*Topology Model*	*Logical convergence time(steps)*	*Efficiency*	*Average throughput (Kbps)*	*Average turnaround time (s)*	*Simulation logical time*
10	1	10	Waxman(m=2)	46	100	4,40	5,076	1051
20	1	20	Waxman(m=2)	109	100	4,39	6,317	1116
30	1	30	Waxman(m=2)	191	100	4,39	6,818	1199
50	1	50	Waxman(m=2)	392	99,978	4,28	7,64	1401
50	5	10	Waxman(m=2)	104	100	4,36	10,231	1118
70	1	70	Waxman(m=2)	647	99,919	4,01	8,23	1599
100	1	100	Waxman(m=2)	747	99,924	4,05	8,62	1755
100	5	20	Waxman(m=2)	207	99,999	4,34	12,88	1225
100	10	10	Waxman(m=2)	132	100	4,36	12,347	1146
150	5	30	Waxman(m=2)	355	99,899	3,90	13,69	1374
200	10	20	Waxman(m=2)	259	99,995	4,30	15,17	1281
250	5	50	Waxman(m=2)	517	99,965	4,19	15,19	1535
300	10	30	Waxman(m=2)	450	99,947	4,05	18,2	1484
350	5	70	Waxman(m=2)	739	99,968	4,13	18,99	1774
500	5	100	Waxman(m=2)	1304	99,829	3,57	18,63	2330
500	10	50	Waxman(m=2)	706	99,85	3,65	21,9	1735
500	50	10	Waxman(m=2)	654	100	4,27	21,091	1688
700	10	70	Waxman(m=2)	1061	99,889	3,82	23,86	2103
1000	10	100	Waxman(m=2)	1385	99,855	3,65	24,11	2419
1000	50	20	Waxman(m=2)	935	99,968	4,13	26,17	1971
1000	100	10	Waxman(m=1)	246	100	4,11	45,243	1321
1000	100	10	Waxman(m=2)	1154	99,991	4,24	23,95	2186
1500	50	30	Waxman(m=2)	1092	99,879	3,71	31,78	2140
2000	100	20	Waxman(m=2)	1638	99,943	4,00	30,095	2680
2500	50	50	Waxman(m=2)	238	99,817	3,44	39,46	1289
3000	100	30	Waxman(m=2)	2184	99,86	3,63	34,807	3230
3500	50	70	Waxman(m=2)	245	99,869	3,67	35,24	2707
5000	50	100	Waxman(m=2)	2496	99,678	2,86	37,06	3544
5000	100	50	Waxman(m=2)	2445	100	3,74	95,431	1523
7000	100	70	Waxman(m=1)	407	99,9	3,51	90,47	1542
10000	100	100	Waxman(m=2)	3321	99,547	2,30	39,906	4372

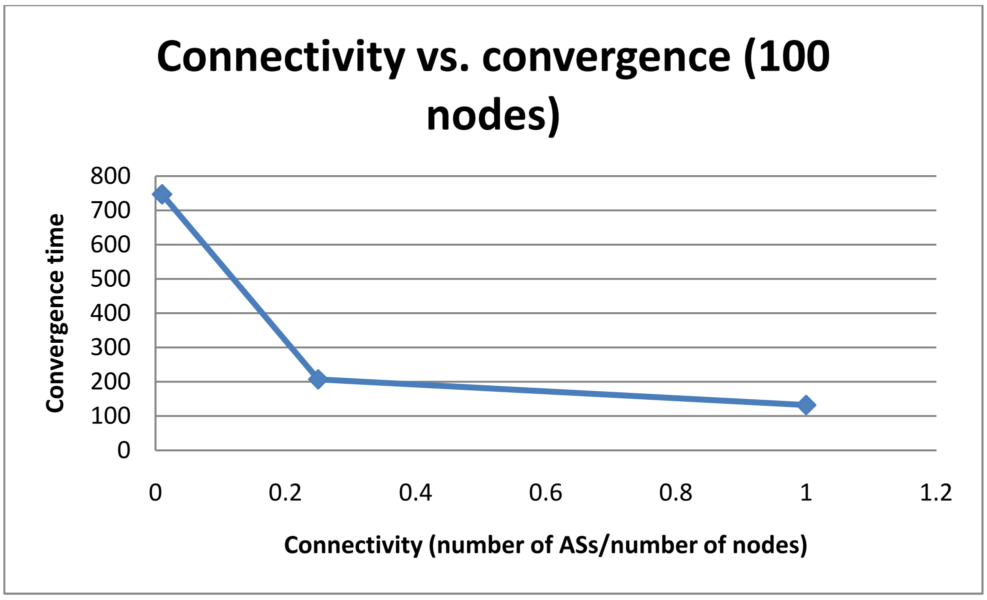


Fig. 11. Performance Effect of Connectivity

5. Conclusions and Discussions

This paper presented the design of a new simulation environment for research design alternatives of the large-scale simulation of the computer networks. Despite the continuous effort made to support high-fidelity models in network routings, it is in vain if scalability of the models in not considered. DEVS system theoretic approach and automated tools have an important impact on the performance of the simulated system as presented in **Table 5**. DEVS-Suite OSPF simulator overcomes the limitations of ns-2 concerning the performance and viewing and tracking the execution of routing protocols. The developed simulator can be scaled up to large sizes since it has underlying high performance DEVS formalism. Also using variable structure DEVS [41] may be better to mimic dynamic nature of the distributed systems such as computer networks. Although models are scaled up to ten thousands nodes, simulation experiments can be scaled up to hundred thousand with DEVS/HLA [42] interface.

Acknowledgment

This work has been funded by the Sakarya University Scientific Research Projects Agency under contract 2007-05-02-001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Sakarya University.

References

[1] S. Kim, H. Sarjoughian, and V.Elamvazhuthi, “DEVS-Suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring,” in *Proc. of the Spring Simulation Conf.*, San Diego, CA, pp. 29–36, Mar. 2009.
[2] B. P. Zeigler, H. Praehofer, and T. G. Kim, “Theory of Modeling and Simulation.” New York: Academic Press, 2000.

[3] S. Floyd and V. Paxson, “Difficulties in Simulating the Internet,” *IEEE-ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, Aug. 2001, http://www.icir.org/.oyd/papers.html.
[4] R. M. Fujimoto, W. Lunceford, E. H. Page, and A. Uhrmacher, “Grand Challenges for Modeling and Simulation,” http://www.dagstuhl.de/Reports/02351.pdf, 2002.
[5] Network Simulator 2 (ns-2), http://www.isis.edu/nsnam/ns/, 2010.
[6] Network Simulator 3 (ns-3), http://www.nsnam.org/, 2010.
[7] OPNET Simulator, http://www.opnet.com/, 2010.
[8] J. Cowie, A. Ogielski, and D. Nicol, “The SSFNet Network Simulator,” http://www.ssfnet /homePage.html, Renesys Corporation, 2002.
[9] X. Zeng, R. Bagrodia, and M. Gerla, “GLOMOSIM: A Library for the Parallel Simulation of Large Scale Wireless Networks,” in *Proc. of Parallel and Distributed Simulation Conf.*, pp. 154, 1998.
[10] A. Varga, “The OMNeT++ Discrete Event Simulation System,” http://www.omnetpp.org/, 2010.
[11] H. Sarjoughian and B. Zeigler, “DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment,” in *Proc. of SCS Western Multi-Conference*, vol. 5, San Diego, CA, pp. 29–36, 1998.
[12] A. Zengin, H. S. Sarjoughian, and H. Ekiz, “Honeybee Inspired Discrete Event Network Modeling,” in *Proc. of 16th European Simulation Symposium*, Budapest, Hungary, pp. 176–182, 2004.
[13] H. Sarjoughian, Y. Chen, and K. Burger, “A Component-based Visual Simulator for MIPS32 Processors,” in *Proc. of Frontiers in Education*, Saratoga Spring, New York, Oct. 2008.
[14] M. Steenstrup, Routing in Communications Network. Prentice-Hall, 1995.
[15] E. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numerische Mathematik*, vol. 1, 1959.
[16] ACIMS, “DEVSJAVA Modeling and Simulation Tool,” http://www.acims.arizona.edu/ SOFTWARE, 2010.
[17] H. Sarjoughian, “DEVS-Suite WebStart,” http://acims1.eas.asu.edu/WebStarts/, 2010.
[18] A. Chow, “Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and Its Distributed Simulator,” *International Transactions of the Society for Computer Simulation*, vol. 13, no. 2, pp. 55–67, 1996.
[19] H. Sarjoughian and F. Cellier, “Discrete Event Modeling & Simulation Technologies: A Tapestry of Systems and AI-based Theories and Methodologies for Modeling and Simulation,” Springer Verlag, 2001.
[20] H. S. Sarjoughian and R. Singh, “Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles,” in *Proc. of the Advanced Simulation Technology Conf.*, Washington DC, pp. 99–104, Apr 2004.
[21] E. Helser, “Design and Analysis of View Synchronization in DEVS-Suite,” *Master’s Thesis*, Computer Science and Engineering Department, Arizona State University, 2009.
[22] The Ptolemy Project, http://ptolemy.eecs.berkeley.edu/ptolemyII/, 2010.
[23] H. S. Sarjoughian and V. Elamvazhuthi, “COSMOS: A Visual Environment for Component-based Modeling, Experimental Design, and Simulation,” in *Proc. of the 2nd International Conf. on Simulation Tools and Techniques*, pp. 1–9, 2009.
[24] Mathworks Simevents, http://www.mathworks.com/products/simevents/, 2010.
[25] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, “A Generic Framework for Parallelization of Network Simulations,” in *MASCOTS*, pp. 128–135, 1999.
[26] J. Guo, W. Xiang, and S. Wang, “Reinforce Networking Theory with OPNET Simulation,” *JITE*, vol. 6, pp. 215–226, 2007.
[27] L. Begg, W. Liu, K. Pawlikowski, S. Perera, and H. Sirisena, “Survey of Simulators of Next Generation Networks for Studying Service Availability and Resilience,” Department of Computer Science and Software Engineering University of Canterbury, Christchurch, New Zealand, Tech. Rep., Feb. 2006.

[28] J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus, “Comparative Study of Wireless Network Simulators,” in *Proc. of the Seventh International Conf. on Networking, Washington, DC, USA: IEEE Computer Society*, pp. 517–523, 2008.
[29] S.-H. Yook, H. Jeong, and A.-L. Barabasi, “Modeling the Internet’s Large-scale Topology,” doi:10.1073/pnas.172501399, 2001.
[30] M. A. Rahman, A. Pakstas, and F. Z. Wang, “Network Modelling and Simulation Tools,” *Simulation Modelling Practice and Theory*, vol. 17, no. 6, pp. 1011–1031, 2009.
[31] A. Medina, A. Lakhina, I. Matta, and J. Byers, “BRITE: An Approach to Universal Topology Generation,” in *Proc. of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems,* Cincinnati, Ohio, USA, 2001.
[32] S. D. Webb, W. Lau, and S. Soh, “NGS: An Application Layer Network Game Simulator,” in *Proc. of the third Australasian conf. on Interactive entertainment*, pp. 15–22, 2006.
[33] S. McCanne and S. Floyd, “ns-LBL network simulator,” http://www-nrg.ee.lbnl.gov/ns/, 1997.
[34] F. Baumgartner, M. Scheidegger, and T. Braun, “Enhancing Discrete Event Network Simulators with Analytical Network Cloud Models,” in *Proc. of International Workshop Inter-domain Performance and Simulation*, pp. 2130, 2003.
[35] H. Sarjoughian and K. Shaukat, “A Comparative Study of DEVS and ns-2 Modeling Approaches,” *International Transactions of the Society for Modeling and Simulation*, 2009.
[36] A. Zengin and H. Sarjoughian, “Teaching and Training of Network Protocols with DEVS-Suite,” in *Proc. of International Symposium on Performance Evaluation of Computer & Telecommunication Systems*, vol. 41, pp. 104–111, Jul. 2009.
[37] I. E. T. Force, “OSPF version 2 (rfc 2328) Internet Standards Track Protocol,” http://www.ietf.org/rfc/rfc2328.txt, 2010.
[38] D. M. Nicol, “Scalability of Network Simulators Revisited,” in *Proc. of the Communications Networking and Distributed Systems Modeling and Simulation Conf.*, 2003.
[39] G. F. Riley, M. H. Ammar, and R. Fujimoto, “Stateless Routing in Network Simulations,” in *Proc. of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 524–531, 2000.
[40] B. M. Waxman, “Routing of Multipoint Connections,” “*IEEE Journal on Selected Areas in Communications*,” vol. 6, no. 9, pp. 1617–1622. http://dx.doi.org/10.1109/49.12889, Aug. 2002.
[41] X. Hu, B. P. Zeigler, and S. Mittal, “Variable Structure in DEVS Component-based Modeling and Simulation,” *SIMULATION: International Transactions of The Society for Modeling and Simulation*, vol. 81, no. 2, pp. 91–102, 2005.
[42] H. S. Sarjoughian and B. P. Zeigler, “DEVS and HLA: Complementary Paradigms for Modeling and Simulation,” *International Transactions of the Society for Modeling and Simulation*, vol. 17, no. 2, pp. 187–197, 2000.

Ahmet Zengin is Assistant Professor at Sakarya University, Turkey. His experience with modeling and simulation includes a one-year-stay in ACIMS Lab at the Arizona State University. His research topics include DEVS theory, multi-formalism modeling, parallel and distributed simulation, modeling and simulation of large-scale networks, distributed systems management, biologically-inspired optimization schemes. His main research interest lies in parallel and distributed simulation and the High Level Architecture.