Information Sciences 222 (2013) 81–98

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Discrete event modeling of swarm intelligence based routing in network systems

Ahmet Zengin [a,*], Hessam Sarjoughian [b,c], Huseyin Ekiz [d]

[a] *Department of Computer Engineering, Sakarya University, Turkey*
[b] *School of Computing Informatics, Decision Systems Engineering, Arizona State University, Tempe, AZ, USA*
[c] *School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran*
[d] *Department of Computer Engineering, Suleyman Sah University, Turkey*

ARTICLE INFO

Article history:
Available online 25 June 2011

Keywords:
DEVS
Beehive
RIP
Modeling and simulation
Network routing

ABSTRACT

Simulation remains attractive for performance and scalability analysis and/or design of networks. This paper presents a biologically inspired discrete-event modeling approach for simulating alternative computer network protocols. This approach identifies and incorporates the key attributes of honeybees and their societal properties into simulation models that are formalized according to the Discrete Event System Specification (DEVS) formalism. We describe our approach with particular emphasis on how to model the individual honeybees and their cooperation. These models, collectively referred to as SwarmNet, support routing algorithms akin to honeybees searching for and foraging on food. Adaptation and probabilistic specifications are introduced into honeybee (BEE) and Routing Information Protocol (RIP) routing algorithms. A set of simulation experiments are developed to show the biologically inspired network modeling with the BEE routing algorithm, as compared with the RIP routing algorithm, offers favorable throughput and delay performance and also exhibit superior survivability against network load surges. The paper concludes with some observations on the SwarmNet modeling approach and outlines some future research directions.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

Modeling systems that exhibit network characteristics remains an area of active research. Despite the presence of classical and contemporary methodologies, there continues to be a growing need for better models for simulating complex network systems ranging from computer systems to social societies. Existing approaches are developed using concepts and suitable model abstractions which are realized in computer programming languages. One class of simulation modeling frameworks is grounded in systems engineering and theoretical computer science whereas another class is born out of abstractions based on natural systems and their formulation in mathematics. In particular, since engineered systems must have precise and highly detailed specifications, it is important for these approaches to support formal structure and behavior specifications. For example, design of a high-speed computer network with guaranteed minimal communication latency must have blueprints that can be implemented and executed based on software and hardware technologies. In contrast, the structure and behavior of natural systems are often not fully known and do not require exact specification given their qualitative dynamics. Specification of the holistic behavior of honeybees and their colonies, for example, is formulated in terms of abstractions that are inherently inexact.

* Corresponding author.
E-mail addresses: azengin@sakarya.edu.tr (A. Zengin), hss@asu.edu, hssarjoughian@ece.ut.ac.ir (H. Sarjoughian), ekiz@ssu.edu.tr (H. Ekiz).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.06.012

The complexity and scalability traits of biological systems provide useful abstractions that can lead to novel modeling approaches suitable for engineered systems. Biologically inspired modeling frameworks are known to be robust, distributed, and fault tolerant [7,23]. Well-known examples are social insects such as ants, bees, and termites. The studies of these insects have been a source of inspiration for computer network researchers for many years [3,5,6,15,28,32,9,20]. However, most existing modeling and simulation frameworks that are used for simulating computer networks do not account for the social capabilities which insects possess in the presence of environmental transformation. While some recent research in modeling and simulation approaches incorporate adaptability and robustness traits of social insects, their underlying modeling constructs and simulation execution concepts are primarily conceived and devised in terms of mathematical algorithms and programming languages.

Model abstractions, and simulation techniques and tools targeted for social and biological systems are based on multi-agent modeling frameworks such as Swarm [31], Ascape [4], and Repast [24]. They typically support abstractions that lack formal modeling concepts and methods that are essential for describing complex, detailed dynamical behaviors. Lack of principles for defining detailed dynamics of computer network nodes and links and hierarchical composition of models along with separate centralized, parallel, or distributed logical or real-time simulation protocols undermines the use of these tools for simulating computer network protocols.

Another class of agent-based modeling approaches was developed based on the modularity and hierarchical concepts defined in discrete event system specification (DEVS) [37]. Agent-based DEVS modeling approaches support developing detailed network systems, such as biological and ecological systems (e.g., [35]). They are extensions of the DEVS formalism in that models can be added or removed during simulation. These approaches support different kinds of dynamic changes in a model's structure and its resulting behavior. In this paper, we developed an agent-based discrete-event modeling approach which incorporates abstractions derived from honeybees. This biologically inspired modeling approach is aimed at simulating computer network systems that can adapt successfully to unpredictable changes in the network nodes and links.

In Section 2, we briefly review DEVS formalism and some related existing modeling and simulation approaches. A description of honeybees, their social dynamics, and concepts that underlie their resiliency given changes in their own structures, those of their environment and several swarm based routing approaches are also discussed in Section 2. In Section 3, the modeling concepts from honeybees are mapped to a set of adaptable agent-based DEVS models. Section 4 presents network model specifications together with experimental models. Examples are developed in the `SwarmNet` environment in Section 5. An analysis of the simulation experiments and an assessment of the approach are also presented in this section. The paper concludes with conclusions and future directions in Section 6.

2. Background

2.1. DEVS modeling approach

Among discrete event modeling approaches, the discrete event systems specification (DEVS) [37] 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 and modular model construction and distributed execution, and therefore offers a basis to characterize complex, large-scale systems with atomic and coupled models. Atomic models characterize the structure and behavior of individual components via inputs, outputs, states, and functions. The internal, external, confluent, output, and time advance functions 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. Atomic models can be coupled together in a strict hierarchy to form more complex models. Parallel DEVS, which extends the classical DEVS, is capable of processing multiple input events and concurrent occurrences of internal and external transition functions. The Parallel DEVS confluent transition function provides local control by handling simultaneous internal and external transition functions.

DEVS atomic and coupled models have corresponding abstract simulators. These simulators dictate how the atomic/coupled models are to be executed according to the semantics of the Parallel DEVS formalism. Computational realizations of the DEVS formalism and its associated simulation protocols are executed using simulation engines such as DEVSJAVA [1] and DEVS-Suite [25]. DEVSJAVA is an object-oriented realization of the Parallel DEVS. It supports describing complex structures and behaviors of network systems using object-oriented modeling techniques and advanced features of the Java programming language. DEVS-Suite includes extensions to DEVSJAVA such as timeview and tracking options. Distributed multi-processor realizations of the DEVS formalism have been developed using technologies such as HLA (e.g., [36]), CORBA [11,10], and Web-services (e.g., [22]). Furthermore, the DEVS models are extended with other kinds of models such as fuzzy logic [21].

2.2. Swarm routing approaches

In this section, several routing protocol approaches developed by inspiration from nature are summarized. Main concerns in such routing protocols are optimality, simplicity, robustness, convergence, flexibility, scalability, multi-path routing, reachability, and quality of service [17]. The ant colony optimization (ACO) methods have been inspired by operating

Table 1
Swarm-based routing approaches [33].

Author	Algorithm	Year
Schoonderwoerd [26]	ABC	1996
Di Caro and Dorigo [8]	AntNet	1997
Subramanian et al. [30]	ABC Uniform Ants	1997
White et al. [34]	ASGA	1998
Bonabeau et al. [6]	ABC Smart	1998
Heusse et al. [18]	CAF	1998
Kassabalidis et al. [19]	Adaptive SDR	2002
Farooq [16]	BeeHive	2006

principles of ants [6], which empower a colony of ants to perform complex tasks such as nest building and foraging [14]. A detailed survey of the swarm routing approaches for network systems can be found in [33]. A list of some well-known swarm routing approaches is provided in Table 1.

2.2.1. Ant-based control (ABC)

Schoonderwoerd [26] first developed ACO based approach for routing in telecommunication networks [33]. In this approach, the basic principle is about the use of multi-agents interacting using stigmergy. Results show that developed algorithm is sufficiently adaptive and demonstrates robustness across difficult network conditions. Randomness in the motion of ants emerges as a novel method which increases the chance of discovery of new routes. Randomize ants traverse the network nodes probabilistically and select the highest probability path.

2.2.2. AntNet

AntNet is developed by Di Caro and Dorigo [8]. Having an enhancement to ABC, the algorithm is developed particularly for asymmetric packet-switched networks [14]. The algorithm implicitly achieves load balancing by probabilistically distributing packets on multiple paths in which routing is performed with interactions of forward and backward ants. Two kinds of agents called forward and backward ants are devised to gather routing information. Several variations for the AntNet have been developed by researchers in particular for wireless networks [33].

2.2.3. BeeHive

BeeHive is developed with inspiration of the foraging principles of honeybees [17]. Communication between real bees is modeled by designing intelligent bee agents, which are able to make routing decisions in large and complex topologies. The results obtained from extensive simulation experiments conclude that bee agents occupy smaller bandwidth and require significantly less processor time compared to the agents of existing state-of-the-art algorithms [17].

3. Biologically inspired network modeling approach and the developed BEE algorithm

A swarm routing approach derived from honeybees and their interactions was developed using the concept of the honeybee scout-recruit mechanism during foraging [27]. The starting point is based on the idea of “Real bees can find optimum solutions in their foraging activity” [27]. The concept has been effective in developing biologically inspired approaches for modeling and simulating intelligent network systems.

Foraging behavior in honeybee societies is a good example for investigating social insect metaphors such as self-organization [27]. Honeybees collectively decide selection of nectar and pollen resources and allocation of workers (foragers and scouts) to various tasks through self-organization. These selection and allocation processes among honeybees of a hive are performed in the absence of any central management authority. In a decentralized and concurrent way, each bee obeys a set of simple rules based on some metrics such as nectar concentration, as well as distance and travel time to food sources. These metrics, including parameters such as the number of bees responsible for storing food in the hive, determine profitability of a nectar source. If the colony encounters more than one nectar source, the most profitable one is preferred by foragers relative to other sources with less profitability. Foragers are distributed among nectar sources using profitability criterion during the course of nectar collecting process. If the amount of a nectar at a given location changes, then the importance of the nectar source is reduced for the whole colony. Furthermore, the colony deploys a relatively small number of its population called scouts to search for nectar. Scouts locate rich sources and monitor nectar availability in the environment [2]. The assignment of forager bees to food sources according to the profitability criterion is known as the **scout-recruit mechanism** in honeybees. One of the most well-known mathematical models of the honeybee’s scout-recruit system was developed by Seeley [27].

The basic activities of honeybees consist of exploration, exploitation, and abandonment of nectar sources. The collective behavior of honeybees can be partitioned into the following phases: traveling from the hive to food sources, scouting for nectar, collecting nectar from the food sources, returning to the hive, transmitting observed state of the environment (e.g., availability of nectar) to other bees, and responding to the environmental variations while adapting to the changes that are taking place in the hive.

Table 2 shows an analogy between honeybee colonies and computer networks. Computer networks correspond to the colonies of honeybees who seek rich nectar sources which requires finding paths with higher capacity to profitable nectar sources (see Fig. 1). Each network node is analogous to a beehive. The network links correspond to the foraging regions where honeybees leave their hives to gather food. Network nodes exchange control and data packets. Each network node deploys a number of control packets (scouts) to find the most profitable paths for a given destination. The data packets correspond to foragers carrying nectar to beehives. Data packets are dispatched from a source to a destination according to routing information gathered by the control packets. The router inside each node uses the information received from all the nodes in the network (obtained by its control packets) to calculate the shortest path to each destination in terms of a chosen metric. The network links are used to limit the number of control and data packets that can be sent and received between nodes. This abstraction corresponds to the amount of information scouts and foragers can carry while searching for and transporting nectar.

Control packets help to reduce congestion by making alterations to routing tables in order to route new traffic away from congested nodes. Control packets also play a crucial role in ensuring survivability of the network based on the same principle through which honeybee colonies maintain their existence by using scouts to search for nectar. Furthermore, the movement of control packets is used to balance network loads. Therefore, each node can have an ensemble of scouts controlling network congestion.

Self-organization of artificial bees is based on these relatively simple rules derived from individual insect's behavior. These artificial bees correspond to a special class of automata called **scout-recruit** that react to their local perception of the environment by stochastically adopting predefined behaviors. The BEE (scout-recruit) routing algorithm is defined by the following rules:

- Rule 1: Periodically or in event-triggered way, each hive dispatches scout bees for gathering information about food sources and choosing the best route (see Algorithm 1).
- Rule 2: During foraging, the goal of each scout is to collect nectar as much as possible.
- Rule 3: Scouts wonder around and gather information about the status of the food sources and routes (see Algorithm 2).
- Rule 4: Given each node and its routing table, scouts choose neighbor nodes using probabilities.
- Rule 5: In the network, the nectar quantity collected by flying along a certain route is inversely proportional to the route cost.
- Rule 6: Scouts are delayed at congested links.

Table 2
Analogy between network systems and honeybee colonies.

Computer networks	Honeybee colonies
Nodes	Beehive**s**
Links	Flying to/from nectar/hive
Control packets	Scouts
Data packets	Foragers
Routing information	Dances and cues

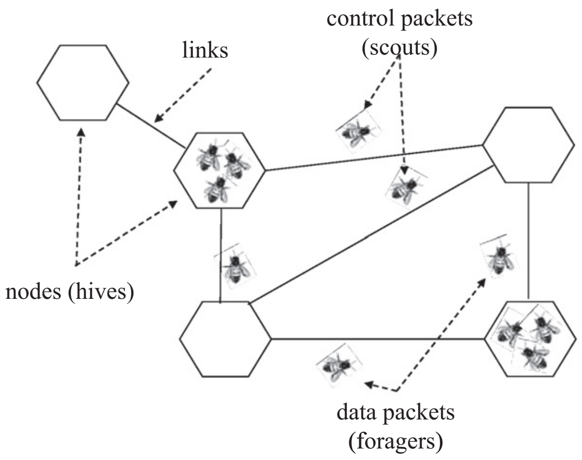


Fig. 1. Network structure of a typical hive.

- Rule 7: Once a scout has reached its destination, it goes back to its source. During return, it has high queuing precedence in order to avoid delaying its arrival to source.
- Rule 8: A scout never visits a node twice in one direction. To do so, scouts save a tabu list which includes the IPs of the visited node.
- Rule 9: A scout never uses a link that does not have enough bandwidth.
- Rule 10: A scout dies after it has reached its maximum number of hops.
- Rule 11: When all or some scouts have returned to their beehives, the costs of finding or recording paths are evaluated and entered to the routing table (see Algorithm 3).

Algorithm 1: Algorithm for launching scouts

```
procedure createScout()
for i = 0 number_of_neighbors do
  create new IP packet called scout
  packet_precedence ← 7(highest)
  packet_source ← address
  packet_id ← agent
  sendScout ()
end for
end procedure
```

Algorithm 2: Algorithm for processing scouts

```
procedure processScout()
if scout has already visited the router || its hop limit is reached then
  discard the scout
else
  if scout is searching for nectar then
    add IP address of that router to scout's tabu list
    retrieve a route from routing table for scout's destination
    hopCount++
    forward scout to ongoing interface
  else
    put scout in front of the queue
    hopCount++
    forward scout to ongoing interface in its tabu list
  end if
end if
end procedure
```

Algorithm 3: Algorithm for updating routing table

```
procedure updateRoutingTable(scouts)
while there are still scouts in the queue do
  routing_table's_time_stamp ← scout's_time_stamp
  current_scout ← next_scout
  if there is a route in the routing table for the destination of the scout then
    if matching route in the routing table is originally advertised by the interface from which the scout was received
  then
      outgoing_interface ← receiving_interface
      route_cost ← current_route's_cost + 1
    else
      if current route's cost + 1 < matching route's cost then
        next_hop ← IP_Address_of_interface_ from_which_scout_was_received
        outgoing_interface ← receiving_interface
        cost ← current_route's_cost + 1
        update the time variable unless cost is INFINITY
```

(continued on next page)

Algorithm 3 (*continued*)

Algorithm 3: Algorithm for updating routing table

```
            end if
        end if
    else
        if current route's cost < = INFINITY then
            next_hop ← IP_address_of_interface_from_which_scout_was_received
            outgoing_interface ← receiving_interface
            cost++
            update the time variable
            add route to the routing table
        end if
    end if
end while
end procedure
```

In Table 3, the developed BEE algorithm is compared with ABC, AntNet, RIP and BeeHive algorithms. BEE algorithm can disseminate data packets over multi paths such as in AntNet and BeeHive algorithms. Multipath routing is important for providing efficient load balancing. Developed approach is highly fault tolerant since several scouts are deployed at any time in the network. Network system is hierarchically clustered and composed of two layers: autonomous systems (AS) and backbones. Communication between scouts, foragers and drones can be realized directly or indirectly via routing tables. In initialization of the network and in case of an expired route, all scouts can work reactively to find new paths, while their normal operation is proactive, i.e., routes are estimated in fixed time intervals or in an event triggered manner. Probabilistic routing is applied since the probabilistic routing used in the BEE algorithm forwards the packets' at alternative routes and yields better load balancing.

4. Network model specification

In this section, we describe an adaptable agent-based network modeling approach using the discrete event system specification (DEVS) formalism. The general specifications for the parallel atomic and coupled DEVS models [12] are described in several textbooks and numerous articles. The discrete-event nature of the DEVS formalism fits well when the problem space has parallel and distributed characteristics [29]. Model development effort and time can be substantially reduced as the scale of model increases given the formal properties of hierarchical model construction and the fact that relatively simple atomic model dynamics result in complex emergent behavior. As part of this work, a network simulator called SwarmNet is developed using the DEVS-Suite simulator with the support to collect data and conduct performance of analysis. Network data measurement and collection efforts enhance the quantity and quality of datasets (e.g., topology, performance, workload, and routing). To build the SwarmNet framework, all components forming the whole network such as nodes (routers, hubs, and switches), links, and complex data elements, and their interactions and parameters are defined based on the atomic and coupled DEVS component model specifications.

Although SwarmNet is devised by supporting common attributes of network entities in the existing best-effort Internet, the approach can accommodate other network architectures (e.g., the mobile wireless network architecture and the sensor network architecture). Users can build a network scenario in a plug-and-play fashion by connecting components in their desired manner within SwarmNet. Users may also extend appropriate model components and redefine new attributes and methods to incorporate their own protocols/algorithms since the dynamics specified within the nodes and links can be used to determine the behavior (e.g., throughput time) of the network model.

In order to model network systems, we have defined a set of basic DEVS model components, including nodes which communicate with one another via links. Since it is assumed that only nodes and links of a network can cause bottleneck, they

Table 3
Routing algorithms comparison [16].

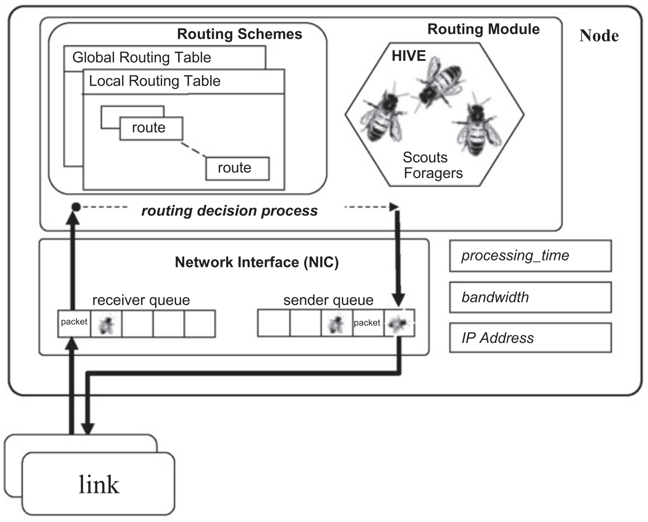
Property	ABC	AntNet-FA	BeeHive	RIP	BEE
Routing strategy	Single path	Multi path	Multi path	Single Path	Multi path
Fault tolerance	Weak	Weak	Strong	Strong	Strong
Hierarchy	Flat	Flat	Hybrid	Flat	2-layer hierarchical
Communication	Indirect	Indirect	Direct	–	Hybrid
Loop freedom	No	No	No	Yes	No
Operation	Proactive	Proactive	Proactive	Proactive	Hybrid
Route selection	Deterministic	Probabilistic	Probabilistic	Deterministic	Probabilistic

are modeled as atomic models and only their states as well as input and output variables are of interest. Other network components such as packets and routing tables are realized and modeled as non-DEVS or non-simulatable models [40]. The network itself is a coupled model which exhibits agent-like behavior and thus supports decentralized control.

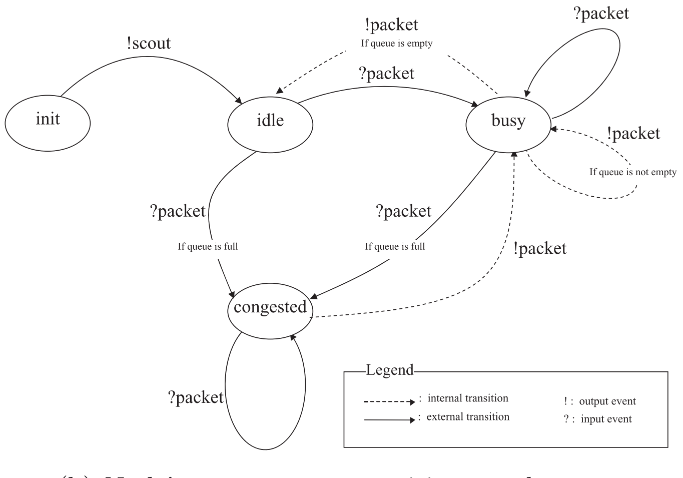
4.1. Node atomic model

Each node in the network represents a switching unit where it is able to process packets that are described below (see Fig. 2). Nodes have interfaces equal to the number of neighbors and each interface has input and output ports for incoming and outgoing packets. Each node has an IP address, which has a unique ID and a unique name. The IP address is used to identify each computer in the network so that packets can be directed to specific destinations. IP addresses also specify the location of a router in the network. Although DEVS coupling through port names is available, the use of the IP address scheme provides concrete representation consistent with real-world computer networks. Due to networks having various devices, with simple changes to a node, a device can be considered as a router, switch, hub, or gateway. Nodes are connected to subsets of other nodes called neighbors via links.

The main part of the node model is its network interface (NIC), which provides fundamental internetworking services. A network interface representing the communication layer is responsible for network topology management. As shown in the



(a) Schematic of the Node and its parts



(b) Node's states, state transitions, and output

Fig. 2. Specification of the node model.

Fig. 2(a), NIC has two queues for incoming and outgoing data and control packets. The NIC specification and implementation accommodate external transition and output functions of the model.

At each node, packets are forwarded to their destination by using information stored in the node's routing module. The routing module reflects the node's routing capability and intelligence. The routing module includes a local routing table for the local network as well as a global routing table which can be used to manage the routing between the local network and other parts of the global network. The global routing table is used to partition the entire network into manageable sizes and therefore it is possible to investigate large-scale networks such as the Internet. Routing tables reflect state of the network that gives information about which nodes are active and which links or paths are available at various time instances [29]. Dynamically changing topology or state of the network has to account for the routing tables as fast as possible. Table entries have a resemblance to distance vectors. The implementation of the routing table consists of a collection of Route objects where each is an instance variable of the routing module class (see the dashed line shown in Fig. 2(a)). When a node needs to send a packet to a given destination, decisions about which outgoing link (i.e., DEVS atomic output port) to be used are made by means of the information specified in its routing table. In Section 4.3.2 routing tables are given in detail.

Besides the above components, the generic node model includes a 'beehive' especially designed to implement and test swarm-based routing algorithms. But it can also be used to implement classical algorithms. For example, a beehive can launch RIP (routing information protocol) messages for distance vector routing protocols, special-purpose agents for agent-based routing protocols, and LSA (link state advertisement) packets for link states. Both RIP and LSA packets can be considered as control messages. RIP and RIP-derived protocols are based on the distance vector. They use mathematical formulations to compare routes to identify the best path to any given destination address. LSA is a basic communication architect of the open shortest path first routing protocol which transports a router's local routing topology to all other local routers. For the swarm application, the beehive model is configured to launch scouts, foragers, and others to monitor and reconfigure network resources. Scouts serve as control packets and search for spare network resources. Scouts find available network resources and monitor their availability.

To define the behavior of a node, the packet process speed and queue capacity parameters are defined. Packet process speed directly influences processing time in a node, and the queue capacity determines the number of packets that can be accepted by the node. Different kinds of bottlenecks in the network can be formed by varying these parameters. A simplified statechart of the node model is shown in Fig. 2(b). The statechart shows some of the main states and transitions given input and output events. For example, when the node is in phase 'busy' and the queue is full, if it receives a packet from another node, it will enter state 'congested' and the packet is lost. In the case of not receiving any packets from other nodes, if the node completes processing of a packet and its queue is not empty, it starts iteratively processing packets until the queue is empty. In Appendix A, DEVS node specification is given in Listing 1.

4.2. Link atomic model

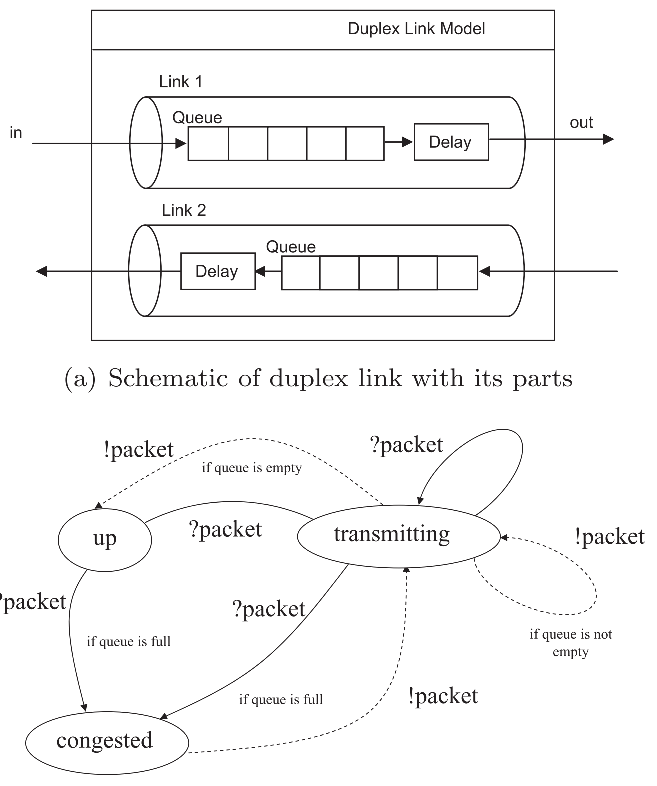
The link model has a central role in developing different kinds of networks with arbitrary topologies. All links are communication channels and therefore are viewed as bit-pipes which are characterized with bandwidth (bits/s) and transmission or propagation delay specified in milliseconds (see Fig. 3). Each link is defined to be bidirectional and thus supports concurrent bidirectional interactions (see Fig. 3(a)). Each duplex link has some finite capacity. The arriving packets are placed in the buffer and are transmitted to the next node using first-in first-out (FIFO) strategy. Links are able to carry traffic of a certain bandwidth up to the total capacity of the link. The specification of the link is akin to a simple processor with a queue that can process incoming packets according to FIFO or some other discipline. Each link atomic model has input and output ports for connecting two nodes in a duplex manner (see for example Link 1 atomic model in Fig. 3(a)). The basic dynamic behavior of the link in terms of a statechart is shown in Fig. 3(b) and its atomic model specification is given in Listing 2 in Appendix A.

4.3. Data and routing table models

4.3.1. Data packets

All data and control communicated among nodes and links are defined as DEVS messages. A packet type is defined to have the following fields: source address, destination address, source hop address, destination hop address, packet id, precedence, total length, tabu list, TTL (time to live), and data (see Fig. 4). All these fields excluding data constitute the protocol header and are 20 bytes as in standard IP packet. Data size can vary across the applications. Packet id characterizes the packet. In order to avoid a packet traveling around the network for a long time, we restrict the packet with a specific hop count, called TTL value. Total size of a packet is stored in the length field of a packet. The packet storing sequence in the queue is determined by the precedence value ranging from the lowest priority 0 to the highest priority 7. Tabu list keeps track of the visited nodes. Source and destination address fields denote a packet's origin and end IP addresses. Data field is used to contain data objects and it should be protected from malicious users.

The data and control packets, unlike node and link model components, do not have time-based behavior. Packets (messages) are defined based on basic IP packets, which carry information such as id and priority (see Fig. 4). Control packets allow gathering information about network nodes and traffic across the links. The generic specification of the messages supports modeling alternative routing schemes. For example, control packets may be routing information packets for distance



(a) Schematic of duplex link with its parts

(b) Link's states, state transitions, and output

Fig. 3. Specification of the link model.

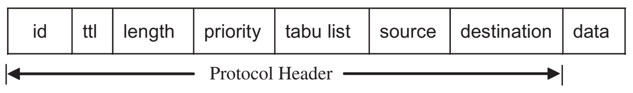


Fig. 4. Data packet specification consisting of protocol header and data.

vector applications, or they are cooperative scouts or traveling ants in swarm-based routing. As already mentioned, swarm-based routing is capable of exhibiting an ensemble of scouts controlling congestion. Scout-named control packets are searching for nectar in the network, similar to the honeybee colonies, to find paths with higher capacity to profitable nectar sources. Routers use the information obtained by their scouts to calculate the shortest path to each destination and then send information to the given destination using the calculated path in the form of data packets or foragers. Data packets (foragers) and control packets (scouts) are differentiated with the packet id field.

Packets traverse intermediate nodes to go to their destination based on their priorities. For example, control packets have higher priority than data packets. The reason is that assigning higher priority to the control packets results in accelerating network control and consequently supports faster responses to congestion. A control packet's priority is used for ordering the elements of its queue. Packets can be discarded upon arriving at a node due to lack of queue space or expired time to live, which is determined by the maximum number of hops a packet can have. In addition, when a packet traverses a link, if there is no available bandwidth, the packet is lost.

In this modeling approach, no arrival acknowledgement or error notification packets are generated and sent back to the source of the packet. Instead, a simple flow control mechanism is devised and implemented. The reason is that we make use of abstraction and focus on routing algorithms by minimizing the number of interacting components. Passing a packet within a link causes a delay that can be viewed as transmission delay. Packets may also be subject to the FIFO delay.

4.3.2. Routing tables

A key feature of the Internet is its structure, which can arbitrarily expand or shrink given a collection of networks - each subnetwork's scale may have few tens to many hundreds of thousands of parts. A host, or device, can only send messages within its own network. Each host has a routing table for every possible destination within its own network, and each table has an entry for every neighbor (see Fig. 5). Data packets can be systematically routed through the network by using routing tables. In defining the routing tables, the hierarchical modeling concept is used to manage the routing table sizes. Hierarchy among a collection of node models is realized given their IP addresses and routing tables.

The routing tables define the static part of the network routing schemes. The capability of the network to dynamically adapt to changing loads is defined in terms of the next node selection probabilities to destination nodes as shown in Fig. 5. The routing table can be viewed as a matrix in which rows correspond to destinations and columns to the node's neighbors. The routing tables can be used to devise, for example, swarm-based movement of packets. The routing tables are initialized at the simulation start up with direct routes between nodes. A direct route has a profitability value of one and lack of a direct route has zero profitability. The entries in the table, therefore, are referred to as profitability values. These values define a range from the least profitable route to the maximum profitable route given the next nodes `B` and `E` and a set of destination nodes such as `B`, `C`, `D`, and `E`. All the values of the entries in the routing table are probabilistic and range between 0 and 1. During simulation execution new entries can be added to the table or current entries can be removed or adjusted according to network traffic.

4.4. Network coupled models

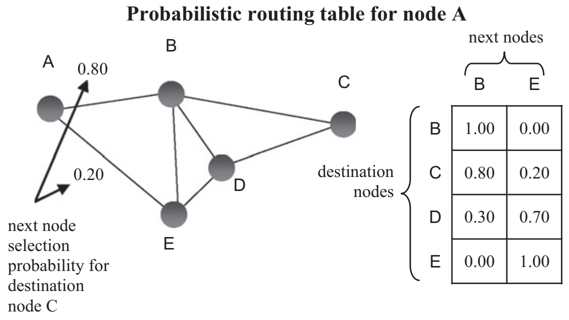
We have developed an approach to simulate models of networks that have varying topologies and structures. The approach is capable of representing the behavior of different routing algorithms (e.g., shortest-path, distance vector, and various swarm algorithms). Hence, it serves as a framework to test and evaluate alternative network configurations and dynamics. Using basic model components and SwarmNet simulation infrastructure, networks can be systematically built and simulated (see Fig. 6). The atomic components `Router1`, `Router2`, and `Link1` are shown as components of the Network coupled model. Every component's phase and sigma are shown, for example the phase is "initialization" and the sigma is "0" for `Router1` at the start of the simulation (see Fig. 6).

Some applications have been developed in SwarmNet to simulate routing algorithms over some network traffic patterns. They have been devised for verifying the design specification as well as testing the framework itself (i.e., composition). The simple network is used to verify that the routing tables are correctly updated when simulation execution links are going down or coming alive.

4.5. Building experimental models

Given the importance of formulating simulation experiments (formulating questions) and facilitating systematic simulation studies, experimental setups are devised using the concept of an experimental frame [37]. An appropriate experimental frame for network modeling and simulation objectives is defined to develop mechanisms to analyze a topological model of the networks, or Internet core networks in particular, including techniques to identify infrastructural vulnerabilities created by dependencies on critical components. To realize simulation experiments, experimental scenarios can be defined using atomic/coupled models. In this work, a typical experimental frame consists of generator and transducer atomic models (see Fig. 7). These state-based models create network traffic and schedule special events such as unavailability of links or nodes. Similarly, the dynamics of the network can be observed using simple to complex measurement algorithms. Raw data may also be collected and analyzed post simulation.

Earlier, we noted the importance of generating traffic for networks and observing how the network dynamics evolve. An event generator atomic model that can generate data packets was devised. The `EventGenr` atomic model generates packets with fixed time intervals by randomly choosing source and destination addresses. The number of packets and their lengths


Probabilistic routing table for node A
A
B
C
D
E
0.80
0.20
next node selection probability for destination node C
next nodes
destination nodes

	B	E
B	1.00	0.00
C	0.80	0.20
D	0.30	0.70
E	0.00	1.00

Fig. 5. An example of a routing table.

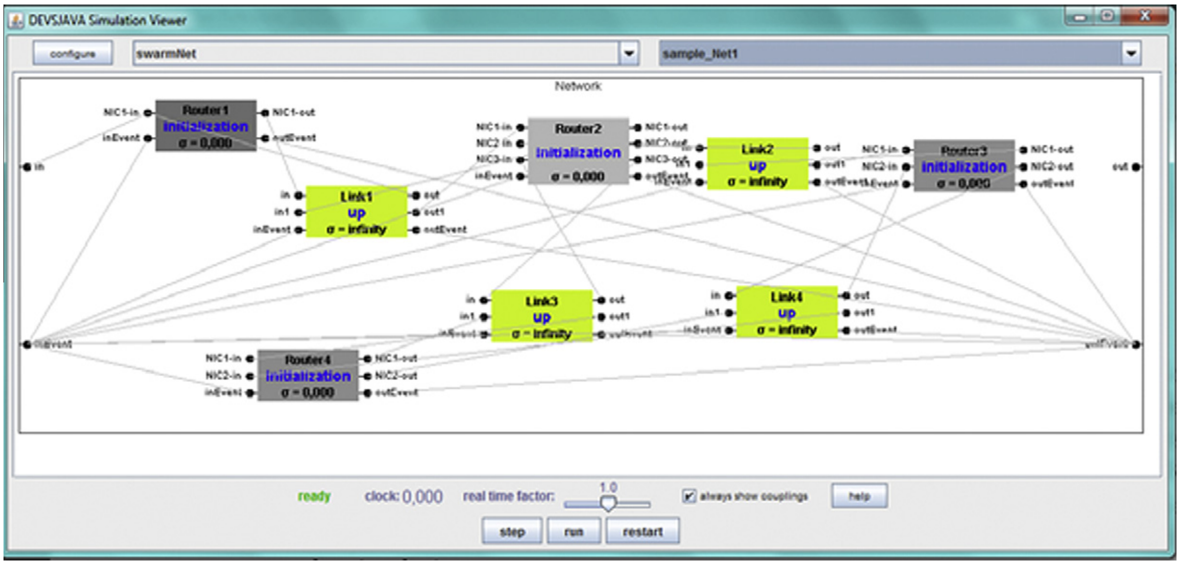


Fig. 6. Synthesis of a simple network model.

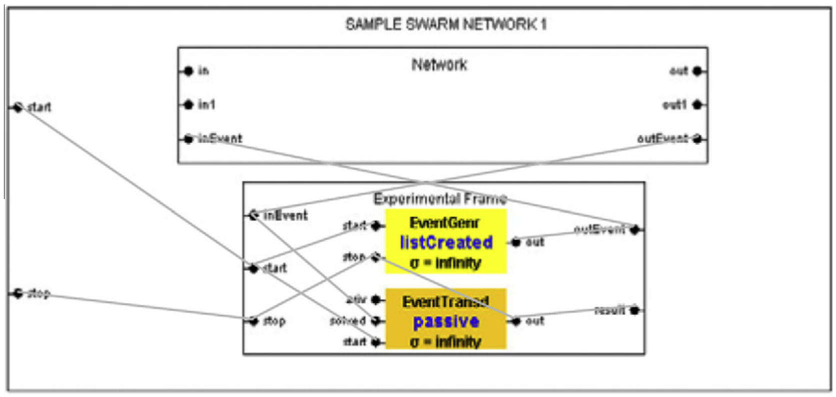


Fig. 7. An experimental frame connected with its network model using DEVS-Suite simulator.

and frequency are determined by selected probability or traffic models such as linear, random, and Poisson. The generator model can also create and schedule specific events for the `Network` model such as 'link down' and 'node congestion', as well as parameter variation, called bursts, to test algorithms in highly dynamic conditions. A transducer atomic model was devised to collect and analyze the network dynamics. The observed data is stored in trace (CSV) files. The transducer atomic

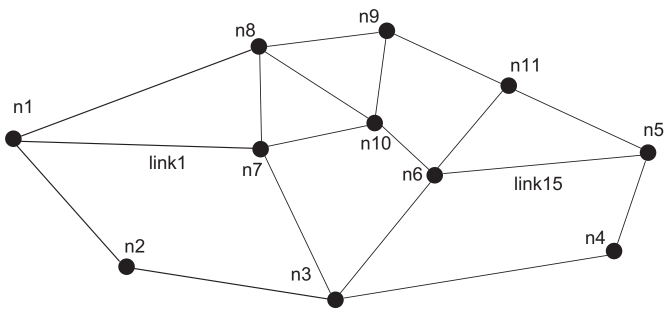


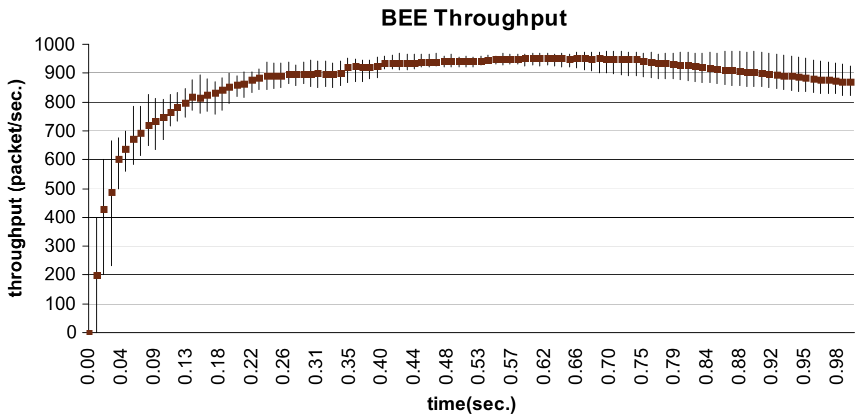
Fig. 8. A simple network consisting of router nodes and duplex links.

component called EventTransd (see Fig. 7) computes network raw data to information which is meaningful for one or more user-defined experiments. As mentioned earlier, parameters such as network throughput and average latency are observed, measured and stored in trace files and can be converted to graphs.

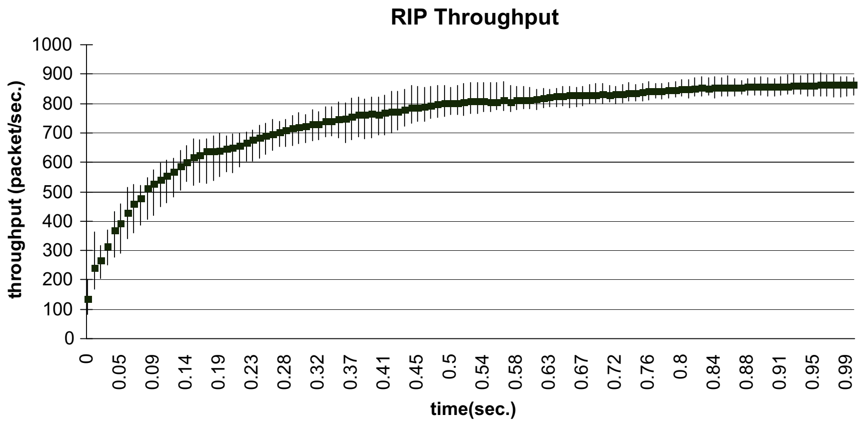
5. Simulation models and experiments

In this section, some example applications are presented to show models' performance on different fields and applications. In the experiments, we applied honeybees' regulation mechanisms to network design and management, and realized a distance vector algorithm for comparison. Both of these algorithms are run in SwarmNet simulator. The biologically inspired network modeling approach described above offers some useful properties such as event-driven updates based on network flow, probabilistic routing, low convergence time, control packet traffic, scalability via clustering, and performance tuning of the network by adjusting parameters in the nodes and links, which reduces routing information stored in a node's memory. Since social insect-inspired approaches bring a probabilistic routing method to the network routing domain, probabilistic cost values are used to represent resource profitability. The cost metric can be based on the bandwidth of the link and dynamically measured in terms of delay or network load. Network load is the sum of the packets' sizes queued in the links, which plays a key role in optimization of the network traffic.

The routing table can be updated with current information about links and network load. In the networks we experimented with, initially no prior knowledge is known about the routes. All routes were computed in parallel during the initialization phase. The cost of each route was determined for a given destination node based on the Dijkstra [13] shortest path algorithm and the minimum number of hops. When an event occurs, such as a link becoming inaccessible or a node becoming dysfunctional, then the routing scheme is able to handle the situation autonomously. An event-driven update was selected for routing information update in response to changes that are detected. The use of event-driven updates rapidly disseminates the data about the failed routes, which reduces sending data through dead routes and decreases the num-



(a)



(b)

Fig. 9. Simulated simple network throughput dynamics using BEE and RIP routing algorithms.

ber of loops. The routing tables and algorithms support a more efficient operation of the network as compared with periodical updates.

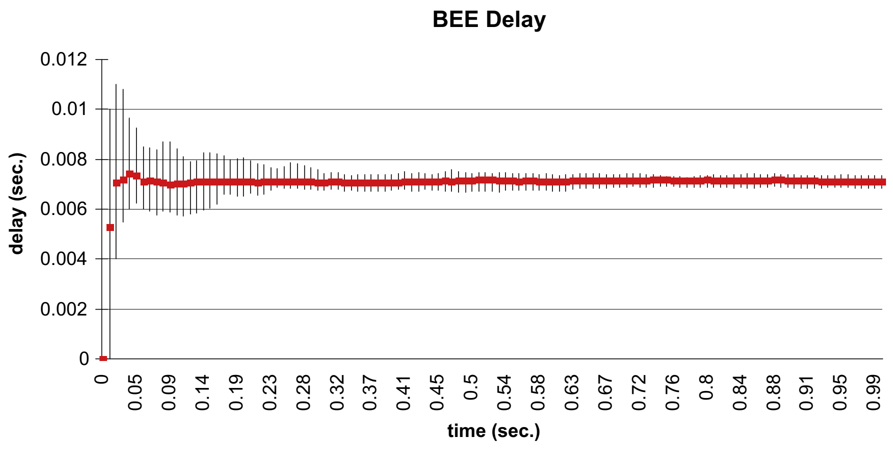
Since one of the main objectives of this approach was to test performance of the new routing approach against the state-of-the-art algorithms, a representative network was used (see Fig. 8). The network model used for comparing the above algorithms has 11 nodes (i.e., `n1,...,n11`) and 18 bidirectional links (`link1,...,link18`). The developed approach was also tested in large-scale network models and difficult network conditions. Results are presented.

Each simulation run consisted of an adaptation to topology (initialization phase) and a test period. During the initialization phase, system runs without load and initial routing tables were formed according to the number of hops (i.e. Dijkstra shortest path estimation algorithm). During the test period, the network performance was measured and recorded in terms of average packet delay, throughput, convergence time, and packet loss ratio.

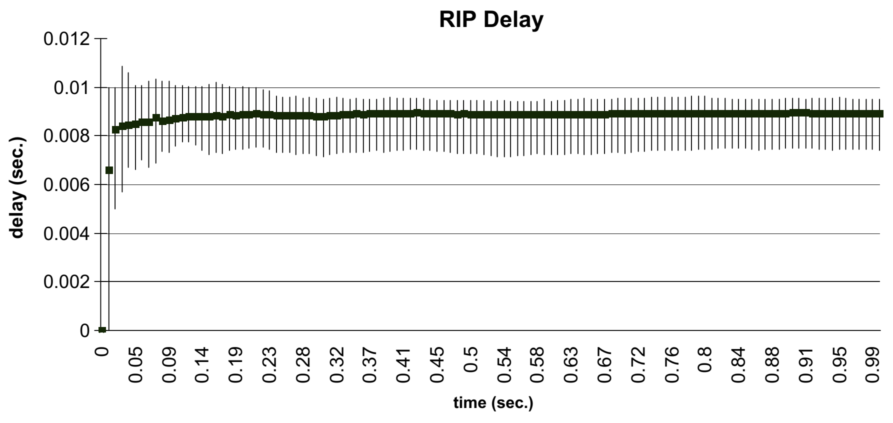
5.1. Node, link, and traffic model configurations

All nodes are configured as routers and each has its own interface (s) equal to the number of their neighbors. Nodes have same buffer size (1 Mbit) but have different IP addresses. Packet processing time of all nodes is equal and selected as 1 ms. Links are bidirectional and their bandwidths range between 1.5 to 6 mbps with propagation delays ranging between 1 and 5 ms. In simulation experiments, real network parameters are selected so that realism is established. Today's Internet has infrastructure in terms of bandwidths ranging from 1.5 to 6 mbps. Therefore these realistic values are implemented.

Traffic flows in the network are simulated by a traffic generator model component. This model generates data packets, then periodically sends out to the network using uniform and randomly selected source and destination nodes. We observe



(a)



(b)

Fig. 10. Simulated simple network dynamics using BEE and RIP routing algorithms.

the network for one second. Generator sends 1000 packets to the network in one second with packet sizes varying from 10 bytes to 100 Kbytes.

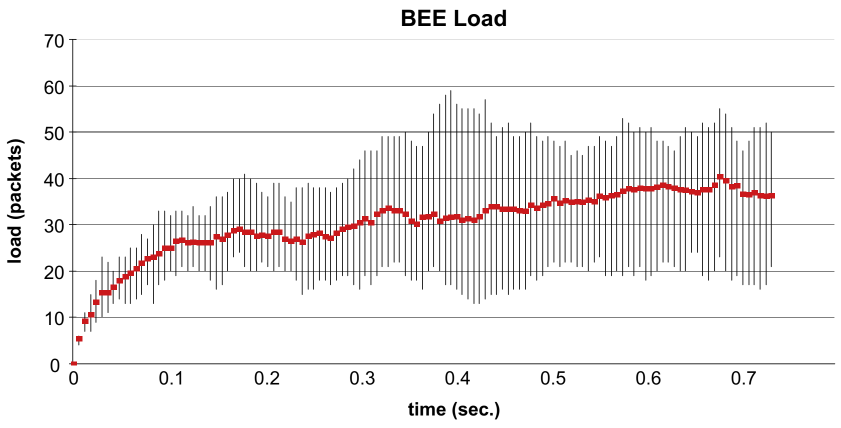
We used two standard performance metrics: throughput and packet delay. We avoided generation of packets with the same source and destination. The amount of network traffic is determined by the number of packets in the network. Generally, many packets must wait in limited capacity (FIFO queue) for processing at the nodes.

5.2. Simulation results and discussion

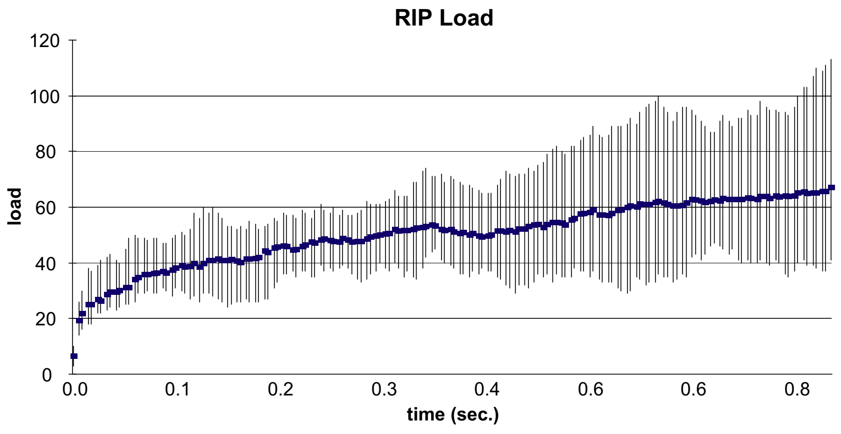
We compare our approach with the RIP (routing information protocol) algorithm which is commonly used in today's Internet. In Figs. 9 and 10, under heavy network traffic, results obtained from the RIP and BEE algorithms are presented in terms of the throughput and packet delay. The average measures of these two metrics are obtained from 10 simulation runs (see Fig. 9). The BEE routing algorithm is shown to yield approximately 15.94% better throughput with a 95% confidence interval; the t test statistic was −0.0147, with 694 degrees of freedom and an associated P value of $P = 0.99$.

Since throughput in the BEE algorithm reaches its maximum value in a shorter time (approx. 200 ms), the BEE algorithm has better response time compared with the RIP algorithm (refer to Fig. 9(a) and (b)). Once the network model with the BEE algorithm reaches the steady state throughput, the throughput remains nearly constant to the end of the simulation. In comparison, the network model with the RIP algorithm reaches its maximum throughput at a relatively much later time (approx. 500 ms). Therefore, the load balancing provided by the BEE algorithm is reached rapidly and evenly in the presence of heavy network traffic conditions.

We also compared the response times for each of these algorithms based on the turnaround time of the packets that are transmitted through the network (see Fig. 10). In our implementation, the turnaround time is defined as a packet's



(a)



(b)

Fig. 11. Simulated simple network load variations for the BEE and RIP routing algorithms.

Table 4
Comparison of the performance results of synthetic topology networks; CT: convergence time, TH: throughput, TA: turnaround time.

# of nodes	# of AS's	# of nodes per AS	Logical CT (steps)		Ave. TH (packets/s)		Ave. TA (msec)	
			BEE	RIP	BEE	RIP	BEE	RIP
28	1	28	99	254	200	190	6.9	7.36
116	4	28	120	227	193	182	12.37	14.23
319	11	28	157	286	179	178	15.5	20.5
960	33	28	198	526	162	160	15.28	27.02
3520	121	28	208	662	140	108	35.4	29.44

life-span time which starts from the packet generator and ends at the packet transducer while going through the nodes and links of the network model (see Fig. 7). The BEE routing algorithm yielded approximately 1.72 ms. Less delay (see Fig. 10(a) and (b)) with a 95% confidence interval; the t test statistic was 0.034, with 636 degrees of freedom and an associated P value of $P = 0.48$. The primary reason for this difference is attributed to the scouts in the BEE algorithm. The scouts find optimum routes quickly and thus allow faster response time to network changes. The ecological approach has better load balancing since the probabilistic routing used in the BEE algorithm forwards the packets' alternative routes. Furthermore, the network resources are better utilized and thus the network traffic load is distributed evenly across nodes and links. This reduces network congestions and results in the packets reaching their destinations faster. Referring to Fig. 10(a) and (b), it can be seen that the difference between minimum and maximum values is smaller as compared with the RIP algorithm. The simulation experiments show improved precision, stabilization and consistency of the BEE routing scheme. Also, the use of random traveling of the agents (scouts) increases the robustness of the network operation.

The difference between the network loads in BEE and RIP algorithms was approximately 20 packets, with a 95% confidence interval; the t test statistic was −0.07, with 1232 degrees of freedom and an associated P value of $P = 0.47$. Based on these observations, the network has higher survivability against surges as shown in Fig. 11(a) and (b).

The developed BEE approach was also compared to RIP routing protocol in increasing network size. In Table 4, performance comparison of the BEE and RIP approaches are summarized. According to the results in Table 4, BEE protocol yields better throughput for all network sizes and less turnaround time except for largest network case. In the largest network, RIP gives better delay against the BEE algorithm (29 msec for BEE and 35.4 ms for RIP). The reason is that RIP causes to loose approximately 50% packets and this decreases packet's delay more than BEE case. The differences between throughput values of the both protocols are kept small until largest network. In the largest network, the difference is big (140 against 108) and it can be said that BEE protocol has a better and efficient run in the larger networks.

The convergence behavior of the BEE algorithm is also considerably better, for example BEE protocol convergence is about three-times less than RIP protocol (208 against 662). Detailed large-scale model definitions and behaviors of the BEE algorithm and experiments can be found in [39] and large-scale OSPF behaviors can be found in [38].

6. Conclusions and future directions

The SwarmNet approach supports modeling and simulating adaptive, robust, and survivable network applications. Since the formulation and implementation of this approach are based on the DEVS formalism and the DEVS-Suite simulator. The resulting simulation environment supports modeling of networks using the biologically derived rules instead of complex formulas; simulation models can be developed systematically and simulated efficiently. The BEE algorithm yields improved response time for management of networks, discovery and deployment of new routes and provides higher robustness. In the case of network surges, the model supports faster response time given network traffic fluctuations and can find alternative paths for destinations at run-time. The BEE routing algorithm derived from the concepts found in the social insect societies shows better performance compared with the commonly used RIP algorithm even as the network size and complexity increases. For future work, the algorithm can be adapted to support wireless networks and the developed SwarmNet simulator can be extended to cover distributed capabilities for ultra large-scale models.

For future directions, developed algorithm can be implemented to support wireless networks and developed simulator can be extended to cover distributed capabilities for ultra large-scale models.

Appendix A. DEVS node model specification

Listing 1, 2.

$DEVS_{node} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$ *where*

To simplify the specification, only one network interface card (NIC) port is assumed.

```
// Input ports and values
X = inports × invalues
inports : {NIC1_in, inEvent},    invalues : {packet, scout, forager, drone}
// Output ports and values
Y = outports × outvalues
outports : {NIC1_out, outEvent},  outvalues : {packet, scout, forager, drone}
// State sets
S = phase × σ × $\overline{Q}$
phase : {initialization, idle, busy, congested}
σ = $\Re^{+}_{0,\infty}$
$\overline{Q} = Q_{receiver} \times Q_{sender}$
where $Q_{receiver}$ and $Q_{sender}$ are queues for incoming and outgoing packets, respectively.
// External transition function
$\delta_{ext}$((phase, σ, $\overline{Q}$), e, (("NIC1_in", x), ("inEvent", x))) =
    if inport ≠ "inEvent",
                    (busy, processing_time, packet)       if phase = "initialization" and $Q_{receiver}$ is not full
                    (congested, processing_time, packet)  if phase = "initialization" and $Q_{receiver}$ is full
    else            (busy, processing_time, packet)
// Internal transition function
$\delta_{int}$(phase, σ, $\overline{Q}$) =
                    ("idle", ∞, $Q_{receiver}$)                if $\overline{Q} = \varnothing$
                    ("busy", processing_time, $\overline{Q}$)  else
//Confluent transition function
$\delta_{con}$((phase, σ, $\overline{Q}$), e, x) = $\delta_{ext}$($\delta_{int}$(phase, σ, $\overline{Q}$), 0, x)
//Output function
λ(phase, σ, $\overline{Q}$) =
("NIC1_out", scout)                 if phase = "initialization"

1. retrieve routing table entry     if phase = "busy" or "congested" and
2. (route.outport, packet)          node.address ≠ packet.destination
                                    28          and packet.id ≠ agent

1. add new routing table entry      if packet.id = agent and packet.name = scout
2. ("NIC1_out", forager)

1. check route for eligibility      if packet.id = agent and packet.name = forager
2. add routes to routing table
3. ("NIC1_out", forager)

1. check route for eligibility      if packet.id = agent and packet.name = drone
2. add routes to global routing table
3. ("NIC1_out", drone)

(inEvent, packet)                   if phase = "busy" and node.address = packet.destination

// Time advance function
ta (phase, σ, $\overline{Q}$) = σ
```

Listing 1.

Listing 2: The Link model specification

$DEVS_{link} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$ *where*

// Input ports and values

$X = inports \times invalues$

inports : {in, in1, inEvent},   invalues : {packet}

// Output ports and values

$Y = outports \times outvalues$

outports : {out, out1, outEvent}, outvalues : {packet}

// State sets

$S = phase \times \sigma \times q$

phase :{up, transmitting, congested}

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

q= buffer

// External transition function

$\delta_{ext}((phase, \sigma, q), e, (("in", packet), ("in1", packet), ("inEvent", packet))) =$

$(\text{“transmitting”}, p\_delay, packet)$ *if q is not full*

$(\text{“congested”}, p\_delay, packet)$ *else*

// Internal transition function

$\delta_{int}(phase, \sigma, q) =$

$(\text{“up”}, \infty, q)$ *if* $q = \varnothing$

$(\text{“transmitting”}, p\_delay, q)$ *else*

//Confluent transition function

$\delta_{con}((phase, \sigma, q), e, x) = \delta_{ext}(\delta_{int}(phase, \sigma, q), 0, x)$

//Output function

$\lambda(phase, \sigma, q) =$

$(\text{“out1”}, packet)$ *if phase =“transmitting” and packet port = in*

$(\text{“out”}, packet)$ *if phase =“transmitting” and packet port = in1*

// Time advance function

$ta(phase, \sigma, q) = \sigma$

Listing 2.

References

[1] ACIMS, DEVSJAVA modeling and simulation tool. Available via <http://www.acims.arizona.edu/SOFTWARE>, 2011.
[2] C. Anderson, The adaptive value of inactive foragers and the scout-recruit system in honey bee apis mellifera colonies, Behavioral Ecology 12 (1) (2001) 111–119.
[3] S. Appleby, S. Steward, Mobile software agents for control in telecommunications networks, BT Technology Journal 12 (2) (1994).
[4] Ascape, Ascape simulation tool. <http://ascape.sourceforge.net/>, 2011.
[5] G. Beni and J. Wang, Swarm intelligence in cellular robotic systems, in: Proceeding of NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, 2630 June 1989, pp. 122–129.
[6] E. Bonabeau, M. Dorigo, G. ThTraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, 1999.
[7] K.-Y. Cai, B.-B. Yin, Software execution processes as an evolving complex network, Information Sciences 179 (12) (2009) 1903–1928.
[8] G.D. Caro, M. Dorigo, Ant colonies for adaptive routing in packet-switched communications networks, in: Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 673–682.
[9] J.-F. Chang, P. Shi, Using investment satisfaction capability index based particle swarm optimization to construct a stock portfolio, Information Sciences 181 (14) (2011) 2989–2999.
[10] Y. Cho, RTDEVS/CORBA: a Distributed Object Computing Environment For Simulation-Based Design of Real-Time Discrete Event Systems. Ph.D thesis, Electrical and Computer Engineering Dept, University of Arizona, 2001.
[11] Y. Cho, B. Zeigler, H. Sarjoughian, Design and implementation of distributed real-time DEVS/CORBA, in: IEEE Systems Man and Cybernetics Conference, Tucson AZ, Oct 2001, pp. 3081–3086.
[12] A. Chow, Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator, Transactions of the Society for Computer Simulation International 13 (2) (1996) 55–67.
[13] E. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.
[14] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics Part B 26 (1) (1996) 1–13.
[15] M. Dorigo, T. Stutzle, Ant Colony Optimization, MIT Press, 2004. ISBN 0-262-04219-3.
[16] M. Farooq, From the Wisdom of the Hive to Intelligent Routing in Telecommunication Networks. Ph.D thesis, Universitat Dortmund, 2006.
[17] M. Farooq, Bee-Inspired Protocol Engineering, From Nature to Networks, Springer, 2009. ISBN 978-3-540-85953-6.

[18] M. Heusse, D. Snyers, S. GuTrin, P. Kuntz, Adaptive agent-driven routing and load balancing in communication networks, in: Advances in Complex Systems, 1998, pp. 15–16.
[19] I. Kassabalidis, M.A. El-Sharkawi, R.J. Marks, P. Arabshahi, A.A. Gray, Swarm intelligence for routing in communication networks, in: Global Telecommunications Conference (IEEE GLOBECOM), vol. 6, 2001, pp. 3613–3617.
[20] D. Kundu, K. Suresh, S. Ghosh, S. Das, B. Panigrahi, S. Das, Multi-objective optimization with artificial weed colonies, Information Sciences 181 (12) (2011) 2441–2454.
[21] F. Lin, H. Ying, R. MacArthur, J. Cohn, D. Barth-Jones, L. Crane, Decision making in fuzzy discrete event systems, Information Sciences 177 (18) (2007) 3749–3763.
[22] S. Mittal, J.L.R. Martin, B.P. Zeigler, DEVS/SOA: a cross-platform framework for net-centric modeling and simulation in DEVS Unified Process, Simulation-Transactions of the Society for Modeling and Simulation International 85 (7) (2009) 419–450.
[23] Q.-K. Pan, M.F. Tasgetiren, P. Suganthan, T. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences 181 (12) (2011) 2455–2468.
[24] Repast, Repast. <http://repast.sourceforge.net/index.html>, 2011.
[25] H.S. Sarjoughian, DEVS-Suite Simulator. <http://devs-suitesim.sf.net/>, 2011.
[26] R. Schoonderwoerd, Collective intelligence for network control. Master's thesis, Delft University of Technology, Faculty of Technical Informatics, 1996.
[27] T. Seely, The Wisdom of the Hive, Harvard University Press, Cambridge, 1995.
[28] A. Sen, Swarm intelligence based optimization of manet cluster formation, Tucson, az, Electrical and Computer Engineering Dept., University of Arizona, 2006.
[29] M. Steenstrup, Routing in Communications Network, Prentice-Hall, 1995.
[30] D. Subramanian, P. Druschel, J. Chen, Ants and reinforcement learning: a case study in routing in dynamic networks, in: 15th Joint Conference on Artificial Intelligence (IJCAI 97), Morgan Kaufman, San Francisco, CA, 1997, p. 832839.
[31] Swarm, Swarm software. <http://www.swarm.org>, 2011.
[32] C. Tovey, The honey bee algorithm: a biological inspired approach to internet server optimization, in: Engineering Enterprise, the Alumni Magazine for ISyE at Georgia Institute of Technology, Spring, 2004, pp. 13–15.
[33] H.F. Wedde, M. Farooq, A comprehensive review of nature inspired routing algorithms for fixed telecommunication networks, Journal of Systems Architecture 52 (8-9) (2006) 461–484. ISSN 1383-7621. Nature-Inspired Applications and Systems.
[34] T. White, B. Pagurek, F. Oppacher, Connection management using adaptive agents, in: International Conference on Parallel & Distributed Processing Techniques & Applications, 1998, pp. 802–809.
[35] G. Zaft, Social science applications of discrete event simulation: a devs artificial society, Master's thesis, University of Arizona Electrical and Computer Engineering Department, 2001.
[36] B.P. Zeigler, S. Hall, H. Sarjoughian, Joint MEASURE an HLA-compliant environment: exploiting interoperability and reuse in lockheed's corporate environment, Simulation 73 (5) (1999) 288–295.
[37] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, Academic Press, New York, 2000.
[38] A. Zengin, Large-scale integrated network system simulation with devs-suite, KSII Transactions on Internet and Information Systems 4 (4) (2010) 452–474.
[39] A. Zengin, Modeling discrete event scalable network systems, Information Sciences 181 (5) (2011) 1028–1043.
[40] A. Zengin, H.S. Sarjoughian, H. Ekiz, Honeybee inspired discrete event network modeling, in: 16th European Simulation Symposium, Budapest, Hungary, 2004, pp. 176–182.