BIOLOGICALLY INSPIRED DISCRETE EVENT NETWORK MODELING

Ahmet Zengin*, Hessam Sarjoughian†, Huseyin Ekiz*

* Technical Education Faculty
Department of Computer Science Education
Sakarya University
Esentepe / Sakarya, TURKEY 54187
{azengin, ekiz}@sakarya.edu.tr

† Arizona Center for Modeling & Simulation
Computer Science & Engineering Dept.
Arizona State University
Tempe, Arizona, USA
sarjoughian@asu.edu

KEYWORDS

Beehive, DEVS, Networks, Routing, Scalability.

ABSTRACT

The simulation study of networks remains attractive due to desire to achieve better important traits such as scalability and performance. This paper describes a biologically inspired discrete-event modelling approach for simulating networks. It introduces a synergistic modelling approach by incorporating key attributes of honeybees and their societal properties into a set of simulation models described in the Discrete Event System Specification. We describe our approach with particular emphasize on how to model the behaviour of the honeybees and their cooperation as discrete event models. The simulation models and their experimental results are presented and discussed.

1. INTRODUCTION

The study of complex networked systems especially those that are large-scale are attractive for a variety of reasons such as the analysis and design of transportation systems, supply networks, management of social and ecological systems. Systems that are composed of components share common characteristics such as hierarchy, alternative configurations, patterns of interactions due to varying types of components and behaviour. To model such systems, one can employ a variety of methods to characterize structure and behaviour – e.g., we can use communicating processes or event systems as the basis for modelling components and their interactions. More specifically, from simulation point of view, we may use discrete event, differential equations, or cellular automata to describe behaviour of networks.

Some modelling techniques, not only can be developed based on *artificial computational models* such as *Von Neumann*, but also on *natural phenomena* such as *Ants societies*. One of the advantages of complementing artificial models of computation with their natural counterparts is that we can have a rich laboratory to develop models that can be experimented with. For example, developing biologically inspired models such as Honeybee enjoy from ample scientific and experimental studies developed based on studying details of the honeybees and their colonies. The knowledge about how honeybees behave and interact offers key insights as how to model inherent complexity of networked systems. Armed with simple yet subtle emerging behaviour of honeybees we may be able to develop models of complex large-scale network systems that can offer desirable performance and scalability qualities (Lunceford and Page 2002).

In the remainder of the paper, we will review Honeybee (Seeley 1995) and DEVS modelling techniques (Zeigler et al. 2000). Based on a general model of Honeybee society, we describe our approach to network modelling where artificial and natural computational models are combined. We then present network model specifications using DEVSJAVA (ACIMS 2004) and associated algorithms followed by describing simulation results and conclusions.

2. BACKGROUND

There exist many modelling approaches founded on systems theory (e.g., Mesarovic and Takahara 1989), agent theory (e.g., Wooldridge and Jennings 1995), and object theory (e.g., Abadi and Cardelli 1996) to characterize network systems such as computer networks and natural societies such as honeybees and ants.

Discrete Event System Specification

Discrete event systems can be described using the Discrete Event Systems Specification (DEVS) formalism (Zeigler et al. 2000) where model behaviour is characterized as events and their processing. This modelling approach supports hierarchical modular model construction, distributed execution, and therefore affords a basis to characterize complex, large-scale systems using formulation of components (atomic and coupled models) and their interactions. *Atomic models* characterize structure and behaviour of individual components via inputs, outputs, states and functions. The internal, external, confluent, output and time advance functions define a component's behaviour over time. Internal and external transition functions describe autonomous behaviour and response to external stimuli, respectively. Confluent transition function is used to account for concurrent occurrences of internal and external transition

functions. Time advance function represents passage of time. Output function is used to generate outputs. Atomic models can be coupled together in a well-defined manner to form more complex models.

A coupled model specifies constructs for composing modular models into hierarchical structures. Behaviour of a coupled model is defined by its constituent atomic (and/or coupled) models. With closure under coupling feature of DEVS, coupled models can be used as atomic models in a larger model. Coupled models can be constructed systematically using the concepts of ports and couplings. When a component sends messages via its output ports, the couplings relay the messages to their designated input ports (Wymore 1993). Upon receipt of messages by atomic models, they immediately process these messages which may result in new states and generation of outputs.

Parallel DEVS is capable of processing multiple input events and provides local control for handling of simultaneous internal and external events. DEVS atomic and coupled models have computational counterparts which may be executed in parallel manner using software engineering concepts (Sarjoughian and Singh 2004). DEVSJAVA is an implementation of the DEVS formalism and its associated simulation protocol (ACIMS 2004). There exist various implementations of the discrete event system specification approach based on single and multiprocessor environments. Parallel and distributed environments have been developed using technologies such as HLA (ACIMS 2004).

Agents

Agent based approaches are being widely used in distributed network applications. This research area is one of the most attractive and rapidly evolving software technologies of the last decades. A software agent concept has emerged from a specialized class of distributed artificial intelligence and is used to describe the concept of a software entity that automates some of the tasks (Hayzelden and Bigham 1998). Software agents can be defined as autonomous, proactive and reactive computational entities that can exhibit the ability to learn, cooperate and move. To make use of software agents in network management applications, agents must be able to migrate from node to node in network. Furthermore, agents must be able to create new agents, delete themselves, and determine their interaction with their environment (e.g., Uhrmacher et al. 2001).

Agent-based solutions are suitable for management of distributed systems since they are inherently distributed and decentralized (Minar et al. 1999). Decentralization is an efficient way to overcome scalability issues. Network systems are dynamic and highly unpredictable systems and therefore their control and monitoring cannot be readily centralized.

Honeybees

Biologically inspired modelling approaches have imported some metaphors from biological systems to engineering systems to develop network management frameworks. Particularly distributed, parallel, robustness and fault tolerant nature of some social insects (ants, bees, and termites) have been a source of inspiration for researchers due to their desirable distributed characteristics. Insect societies have advanced mechanisms to maintain colony level survivability against environment conditions. For example, because nectar availability can change rapidly and unpredictably, honeybees are able to cope with such problems and scale themselves to huge number of population (Seeley 1995). Honeybees have sophisticated regulation mechanisms to adapt their capacities against fluctuating and ephemeral resources.

Foraging behaviour in honeybees is a good example to investigate social insect metaphors such as *self-organization*. *Honeybees* collectively decide selection of nectar and pollen resources and allocation of workers among them through self-organization. This selection and allocation of processes among honeybees in their hive is performed by absence of any central authority. In a decentralized and concurrent way, each bee obeys a set of simple rules based on some metrics (e.g., nectar concentration, location of the source, and travel time to the food source). All of the metrics including parameters such as the number of bees responsible for storing food in the hive determine profitability of a nectar source. If colony encounters more than one source of nectar, highest profitable source 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 nectar amount in a certain source changes, then whole colony changes its concerns to that source. Furthermore, the colony deploys certain portion of foragers for searching nectar, namely scouts. Rich sources are found by scouts and nectar availability in environments is monitored by them (Anderson 2001). This assignment of foragers to sources according to profitability is called *scout-recruit system* in honeybees. One of the most well-known mathematical models was developed by Seeley and documented in his Book (Seeley 1995).

3. NETWORK MODELLING APPROACH

In order to model a distributed networked system, we have defined a set of *network component models* called *nodes* which communicate with one another via *links* (see Figures 1 and 4). Using node and link capacity assignments, we can develop a variety of complex network configurations. The node and link models are defined as the DEVS atomic components. Other network elements such as packets and scouts are also represented as DEVS models. With this approach, a network model exhibits agent-like behaviour and thus supports decentralized control.

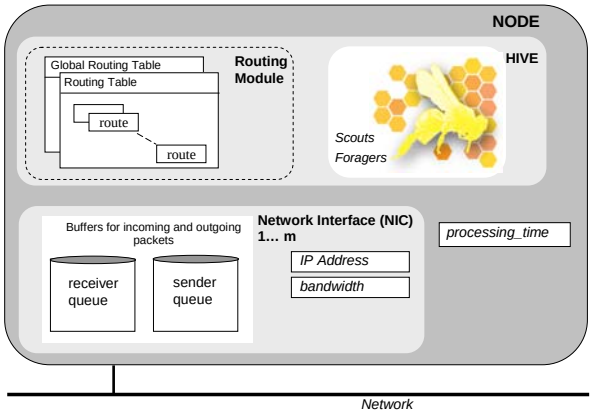


Figure 1: Design of a network node

To realize simulation experiments we utilize the concept of experimental frame where the conditions under which a model can be experimented with and observed are defined. For this work, a typical experimental frame consists of *generator* and *transducer* atomic models. We employ generator in order to create network traffic and to schedule special events such as unavailability of links or nodes. To realize this, a generator model sends messages (see Section 4) to all the appropriate components in the network.

Atomic and coupled models are represented using the parallel DEVS formalism and developed within the DEVSJAVA modelling and simulation environment. In this approach, the dynamics specified within the nodes and links can be used to determine the behaviour (e.g., throughput time) of the network model.

4. NETWORK MODEL DESIGN

In order for modelling a distributed networked system, we have defined a set of basic network simulation model components including nodes and links as detailed next. By coupling these model components in DEVSJAVA, we can develop a variety of network configurations and study network characteristics. Since it is assumed that only nodes and links of a network are able to cause bottleneck, they are modelled 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 realised as stateless entities. Network itself is a coupled model. Defined dynamics in node and link atomic models determine the behaviour of network coupled model. All atomic models in our implementation are modelled and defined using the Parallel DEVS formalism (Chow 1996) and realisation in Java (ACIMS 2004).

Node

Each node in the network represents a switching unit where it is able to process packets that are described below. Due to a node can be considered as a router. With simple changes, a node can represent other network elements such as hub. Nodes are connected to other nodes called neighbours via links. To determine the behaviour of a node, we use two parameters: *packet process speed* which directly influences processing time of a node, and *queue* in which incoming and outgoing packets are stored. By toggling these capacities, different kinds of bottlenecks in the network can be modelled.

As shown in Figure 1, one of the main parts of our nodal structure is the *Network Interface*. It provides the fundamental internetworking services such as packet exchanging with neighbouring nodes. *Routing Module* reflects node's routing capability and simple intelligence. At each node, packets are forwarded to their destination nodes by routing module. A routing module includes a local routing table for *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. Global routing table fragments the entire network into manageable sizes and therefore it is possible to investigate Internet-like (large-scale) networks. These routing tables reflect state of the network and have resemblance with distance vectors. Also, we have equipped our node model with the *beehive* to implement and test *swarm-based routing algorithms*. In our swarm application, beehive launches scouts or other kind of entities to monitor the network and to reconfigure network resources.

Link

All links are communication channels and therefore are viewed as pipes which are characterized with bandwidth (bits/sec) and transmission or propagation 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 first-in first-out (FIFO) strategy. Links are modelled as *bidirectional*, thus supporting concurrent bidirectional interactions. Links are able to carry traffic of a certain bandwidth up to the total capacity of the link. Each link atomic model has input and output ports for connecting two nodes in a duplex manner (see for example Link1 atomic model in Figure 4).

Data Packets

All 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* (see Figure 2). Control packets allow the node to obtain whole network view and to measure the traffic. For example, they are Routing Information Protocol packets in our distance vector application, while they may be cooperative scouts or ants in swarm based routing. Packets traverse intermediate nodes to go to their destination. As depicted in Figure 2, all packets have a priority field which is used for handling them in some way. For example, while control packets and scouts have high priority, data packets have low priority. The data packets, therefore, are queued and served in FIFO setting. Besides handling data packets in FIFO manner, control

packets have higher priority ranging to 7 by which their queuing order is determined. 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.

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

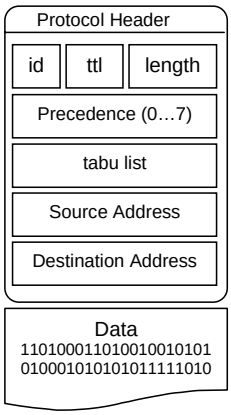


Figure 2: A data packet model with a protocol header

As shown in the Figure 2, we have modelled a packet type with 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*. All these fields excluding data constitute protocol header and are 20 bytes in our model. Data size can vary across applications. Packet id characterizes the packet. In order to avoid a packet to travel around the network for a long time, we restrict the packet with specific hop count, namely TTL value. Total size of packet is stored in the length field of a packet. Packet storing sequence in queue is determined by precedence value ranging from the lowest priority 0 to the highest priority 7. Tabu list allows us to keep track of visited nodes. Source and destination address fields denote packet's origin and final node's IP addresses. Data field is simply used to contain data object.

Routing Table

By equipping each node model with a routing table, data packets can be systematically routed through the network. The Java implementation of the routing table consists of a collection of *Route* objects where each is an instance variable of the routing module class (see Figure 1). 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.

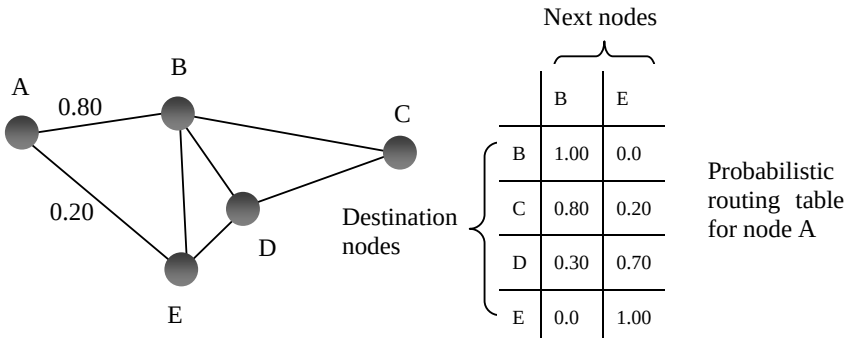
Each node has a routing table for every possible destination in the network, and each table has an entry for every neighbour (see Figure 3). According to the routing algorithm, these routing tables are constructed previously (in static algorithms), dynamically adapted to network load state (in dynamic algorithms) or based on node's (insect's) selection probabilities of the next node to its destination – e.g., using swarm based algorithms. The routing tables are initialised at the simulation start up with routes to directly linked interfaces of cost 1. Routing table can be imagined as a matrix in which rows correspond to destinations and columns to neighbours. During simulation execution new entries may be added to table or current entries may be removed or adjusted according to network traffic. All the values of the entries in the routing table range between 0 and 1, a probabilistic value. We called these entries as *profitability values* through which most profitable routes can be chosen.

Generator and Transducer Models

In order to experiment with the above network model, it is necessary to model user traffic. To make realisations of network traffic and examine specific scenarios, the experimental frame concept and its DEVSJAVA realisation are employed. In our implementation, a typical experimental frame consists of an event generator and event transducer. The generator generates packets with fixed time intervals by randomly choosing source and destination addresses. As mentioned earlier, generator also can create and schedule specific events in the network such as link down and node congestion events. The transducer observes and analyses the network outputs, and stores these results in trace files. Transducer simply converts data to information which is meaningful for us.

Coupled Simulation Models

We have developed a discrete event simulation model for networks with varying topologies and structures. As mentioned above, the developed framework is capable of representing the behaviour of different routing algorithms (e.g. shortest-path, distance vector and various swarm algorithms). Hence, the approach serves as a framework to test and evaluate alternative network configurations. By using basic components and tools which have been described above, networks can be built by coupling node and link atomic models in DEVSJAVA simulation viewer (see Figure 4). Furthermore, by coupling these coupled networks, increasingly larger networks can be systematically developed and experimented with.



	B	E
B	1.00	0.0
C	0.80	0.20
D	0.30	0.70
E	0.0	1.00

Figure 3: An example of a routing table

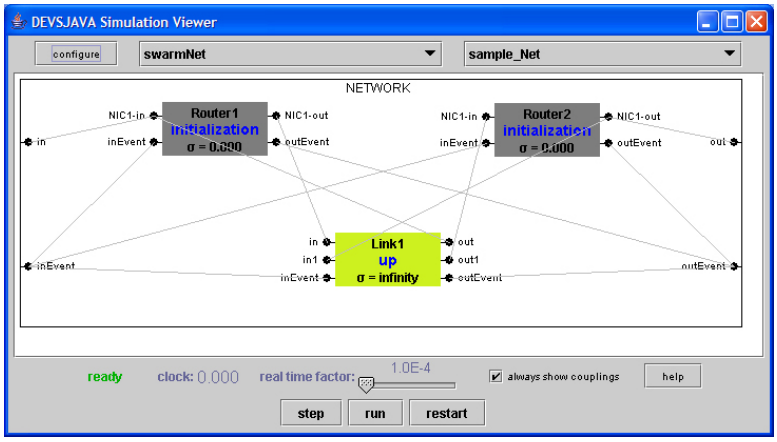


Figure 4: Synthesis of a network model

Some applications have been created in DEVSJAVA to simulate routing algorithms over some network traffic patterns. In our experiments, we have used a simple packet switched network and a set of large-scale networks with increased complexity and connectivity. These networks hereafter are referred to as the simple and complex networks, respectively. They have been designed for testing the model design as well as testing the framework itself (e.g., scalability). The simple network would be sufficient for initial testing of whether routing tables are updated correctly and whether, when links go down or come alive, routing tables are correctly updated. Large-scale and complex networks are used for uncovering dynamics and performance measurements of the models in the DEVSJAVA environment.

Scaling Coupled Models

To support scalability, we employed and implemented a clustering approach. Clustering provides manageable network sizes by abstracting a subnet to single node in a higher level network. By considering a coupled model as an atomic model, DEVS coupled model concept has a resemblance with clustering. There exists a hierarchy of networks within the total of all nodes and routers. Each coupled model has a number of *border nodes* which are used for connecting it to other coupled networks. In our approach, clustering is done in *addressing level* of nodes.

Hierarchical and modular structure of DEVS formalism facilitates implementation of clustering approach. Border nodes have an additional routing table consisting of the cluster names. This approach substantially decreases the information stored in routers.

5. CREATING HONEYBEE INSPIRED NETWORK MODELS

To show the capability (applicability) of the modelling approach, first we started with well-known routing algorithms. We have implemented static link state algorithm (Dijkstra 1959) to initialize network and distance vector to calculate distances between nodes. In the implementations of these algorithms, we used hop number as a metric, but other metrics such as available link bandwidth may also be used.

As pointed out earlier, our biologically inspired approach was derived based on honeybees and their interactions. For example, the movement of packets (artificial bees) can be used to balance network loads. Focused on biological inspired load balancing mechanism is analogous to honeybee scout-recruit system. In honeybee colonies, a colony deploys certain portion of its foragers

for searching nectar, namely scouts. Scouts find rich nectar sources and monitor their availability. If colony finds additional sources of nectar, the highest profitable source 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.

We have developed a set of models which are capable of exhibiting an ensemble of scouts controlling congestion in a distributed environment. In our implementation, analogous to honeybee scout-recruit system, each network node is a beehive. Network corresponds to the world of honeybees who seek rich nectar sources, finding paths with higher capacity to profitable nectar sources, light-weight scout entities searching for nectar, and control packets foraging for information to aid survival of the network (honeybee colonies). Each hive deploys a number of scouts to find the most profitable paths for a given destination. Each router then uses the information received from all the nodes in the network obtained by its scouts to calculate the shortest path to each destination in terms of a chosen metric. Scouts control congestion by making alterations to routing tables in order to route new traffic away from congested nodes. Then, packets are dispatched from a source to a destination according to information gathered by scouts.

The developed approach offers some useful properties such as probabilistic routing, optimal system performance by tuning parameters, event-driven updates based on network flow, low convergence time, low control packet traffic and scalability via clustering which reduces routing information stored in a node's memory. Social insect inspired approaches bring a probabilistic routing method to network routing domain, therefore we use probabilistic cost values in order to represent source profitability. The cost metric can be based on the bandwidth of the link or can be dynamically measured as in the case of delay or load.

The routing table is then updated with the new information. In the networks we have experimented with, initially no apriori knowledge is known about the routes. All routes were computed in parallel during initialization phase. Each route determined for a given destination node based on the Dijkstra shortest path about the minimum hop. When an event occurs, such as a link going down or a node failing, then routing scheme has to be able to handle the situation. An event-driven update is 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 route, which reduces the change for growing data and route loops to occur. By doing so, the entire procedure can be completed in a less time than periodical updates.

6. SIMULATION RESULTS & DISCUSSION

In order to compute the performance of the routing approach, we developed a set of models and experimented with them. Using node and link models defined above, various network topologies can be formed. Then, developed network models are run under traffic load by using experimental frame model. The simplest network modelled has 11 nodes and 18 bidirectional links (see Figure 5), while larger models has up to 3520 components.

Each node in the network is represented a routing table storing the neighbouring node to which traffic should be routed. Each simulation run consisted of an adaptation to topology (initialisation phase) and test period. During the initialisation phase, system runs without load and initial routing tables are formed according to the number of hops (shortest path estimation). During the test period we measured and recorded the network performance in terms of average packet delay, throughput, convergence time and packet loss ratio.

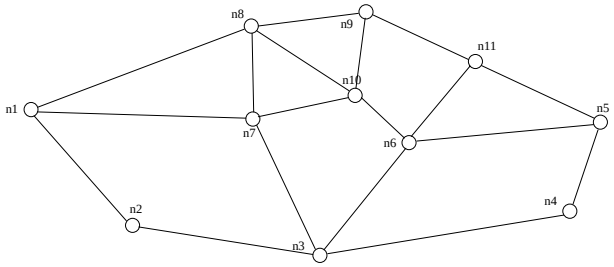


Figure 5: A simple network

Node and Link parameters

All nodes are designed as routers and each one has own interface(s) equal to their neighbours. Nodes have the same buffer size (1Mbit) but have different IP addresses. Moreover, node packet's processing time is selected as 1 msec.

Links are bidirectional and their bandwidths range between 1.5 to 6 mbps with propagation delays ranging between 1 to 5 msec.

Traffic model

Traffic flows in the network are simulated by a traffic generator model component. This model generates data packets which are then periodically sent out to the network using uniformly randomly selected source and destination nodes. We observe network for one second (see Figure 7). Generator sends 1000 packets to the network in course of one second which packet sizes varying from 10 bytes to 100 Kbytes.

We used two standard performance metrics: throughput and packet delay. We avoid 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.

We compare our approach with a state-of-the-art algorithm, namely RIP (Routing Information Protocol).

RIP is an instance of distance vector algorithm and still being widely used in Internet networks (Steenstrup 1995). In Figure 6, results obtained from both RIP and ecological approach are presented together. As mentioned earlier, average packet delay and throughput are major performance criteria for evaluation.

In Figure 6, it can be shown that ecological approach shows better throughput than RIP. After a short time (~ 200 msec), the throughput reaches steady values and remains constant to the end of the simulation. This means load balancing is achieved successfully.

Average packet delay values are almost same, 9 msec (see Figure 7). However, bees approach's packet delay remains low up to 0.5 msec and later has greater values than RIP. The reason is that probabilistic routing forwards the packets alternative routes for load balancing, while RIP selects shortest paths. But, ecological approach has better load balancing and lower packet loss ratio.

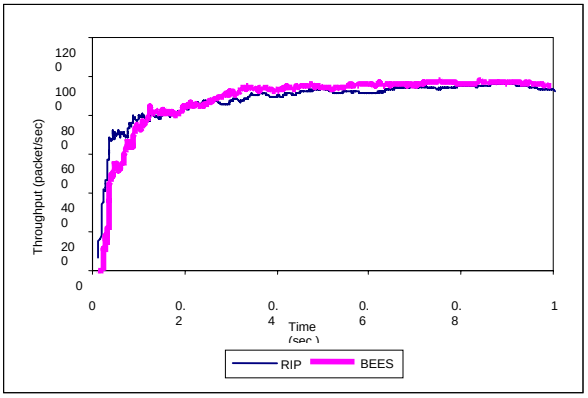


Figure 6: Throughput comparison of different algorithms

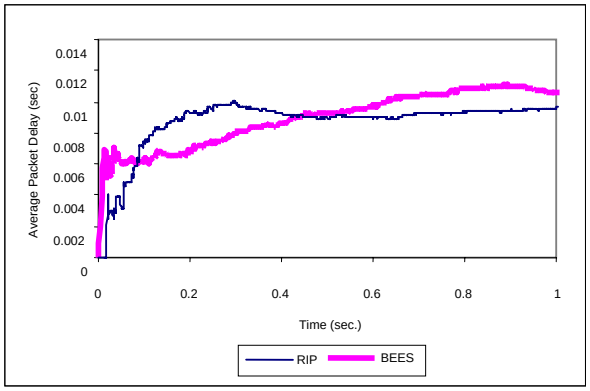


Figure 7: Average packet delay comparison

Performance of the framework on increased scalability and connectivity

In our experiments, one of the key independent variables was the degree of connectivity and scalability. In order to examine the scalability aspect of our approach, we developed various networks ranging from 29 to 3520 components (see Table 1). These models were executed with acceptable performance in the DEVSJAVA environment. The largest network took less than three hours on a 2.4 GHz processor and 512MB RAM while the simple one took a few minutes.

Table 1: Large-scale network models

Network	Number of component	Number of colonies
NET 1	116	4
NET 2	319	11
NET 3	960	87
NET 4	3520	125

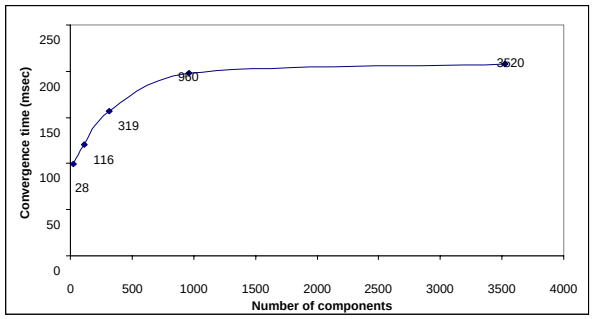


Figure 8: Convergence time of networks with different scales

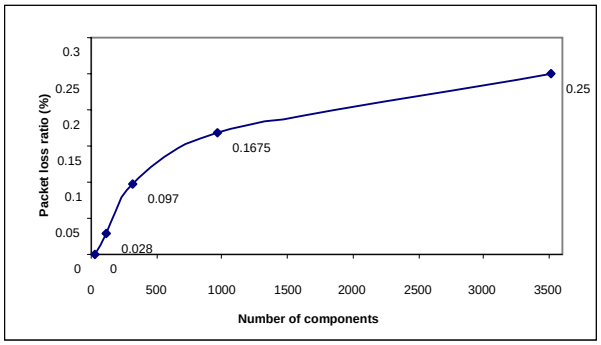


Figure 9: Packet loss ratio of networks with different scales

As shown in Figure 8, larger models exhibited lower than expected time for convergence. This is partially due to the DEVS discrete event modelling paradigm and its implementation in DEVSJAVA. The maximum number of components is partially due to the platform used for executing the simulations. Packet loss ratio gradually increases with the increase in the number of components.

However, packet loss remains acceptable even for large-scale networks (see Figure 9).

7. CONCLUSIONS

This paper proposed a discrete-event modelling approach for networks. The models are devised based on biologically-inspired routing mechanisms to tackle the scalability aspect of large-scale networks. The approach and its implementation are promising for handling models composed of hundreds to thousands of components. The routing strategy, which is based on the behaviour of beehives, is robust and exhibits similar or better performance compared to the contemporary routing RIP technique. This research suggests the proposed modelling approach can be used for the design and development of robust and scalable network systems.

Acknowledgement

This research is partially supported by NSF grant Scaleable Enterprise System (DMI-0122227).

REFERENCES

Abadi, M. and L. Cardelli. 1996. *A Theory of Objects*. Springer.

ACIMS. Arizona Center for Integrative Modeling and Simulation. 2004. http://www.acims.arizona.edu/SOFTWARE/software.shtml

Anderson, C. 2001. "The adaptive value of inactive foragers and the scout-recruit system in honey bee (Apis mellifera) colonies". *Behavioral Ecology*. 12, No. 1, p. 111-119.

Chow, A.C.-H. 1996. "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator", *Transactions of the Society for Computer Simulation International*, 13, No. 2, p.55-67.

Dijkstra, E.W. 1959. "A Note on Two Problems in Connexion with Graphs". *Numerische Mathematik* Vol. 1.

Hayzelden, A. and J. Bigham. 1998. "Heterogeneous Multi-Agent Architecture for ATM Virtual Path Network Resource Configuration", *Proceedings of Intelligent Agents for Telecommunications Applications*, Springer Verlag. 45-59.

Lunceford, W.H. and E.H. Page. 2002. Editors. *International Conference on Grand Challenges for Modeling and Simulation*, San Antonio, Texas, USA.

Mesarovic, M.D. and Y. Takahara. 1989. *Abstract Systems Theory*. Springer Verlag.

Minar N., Gray M., Roup O., Krikorian R., and Maes P. 1999. "Hive: Distributed Agents for Networking Things", *First Int'l Symp. Agent Systems and Applications and Third Int'l Symp. Mobile Agents*. IEEE Computer Soc. Press.

Sarjoughian, H.S. and R. Singh. 2004. "Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles", *Proceedings of the Advanced Simulation Technology Conference*, 99-104, Washington DC (April).

Seely, T.D. 1995. *The Wisdom of the Hive*. Cambridge, Mass: Harvard University Press.

Steenstrup, M. E. (Ed.). 1995. *Routing in Communications Network*. Prentice-Hall.

Uhrmacher, A.M., P. Fishwick, and B.P. Zeigler. 2001, Agents in Modeling and Simulation: Exploring the Metaphor (eds.). *IEEE Proceedings*.

Wooldridge, M. and N.R. Jennings. 1995. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10, No. 2. 115-152.

Wymore, W.A. 1993. *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, Boca Raton, CRC.

Zeigler, B.P., H. Praehofer, and T.G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Second Edition Academic Press.

AUTHORS BIOGRAPHIES

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

HESSAM S. SARJOUGHIAN is Assistant Professor of Computer Science and Engineering at Arizona State University, Tempe. His research includes modeling theory, collaborative modeling, distributed co-design, intelligent agents, and software architecture. His industrial experience has been with Honeywell and IBM. Visit <http://www.eas.asu.edu/~hsarjou/index.htm> and <http://www.acims.arizona.edu> for more information.

HUSEYIN EKIZ is received M.S., and Ph.D. degrees in computer science engineering in 1995 and 1998, respectively, all from the University of Sussex, England. He is currently a Professor of computer systems education and a Dean of the Technical Education Faculty at Sakarya University, Turkey. His research interests are in the fields of network systems, distance education, digital circuit and IC design with VHDL and microprocessor architectures.