Parallel Simulation Made Easy With OMNeT++

András Varga[1], Ahmet Y. Şekercioğlu[2]
[1] OpenSim Ltd
Budapest, Hungary
andras@omnetpp.org

[2] Centre for Telecommunication and Information Engineering
Monash University
Melbourne, Australia
asekerci@ieee.org

***Abstract*—This paper reports a new parallel and distributed simulation architecture for OMNeT++, an open-source discrete event simulation environment. The primary application area of OMNeT++ is the simulation of communication networks. Support for a conservative PDES protocol (the Null Message Algorithm) and the relatively novel Ideal Simulation Protocol has been implemented. Placeholder modules, a novel way of distributing the model over several logical processes (LPs) is presented. The OMNeT++ PDES implementation has a modular and extensible architecture, allowing new synchronization protocols and new communication mechanisms to be added easily, which makes it an attractive platform for PDES research, too. We intend to use this framework to harness the computational capacity of high-performance cluster computers for modeling very large scale telecommunication networks to investigate protocol performance and rare event failure scenarios.**

Keywords - parallel simulation, discrete-event simulation, PDES

I. INTRODUCTION

Telecommunication networks are increasingly becoming more complex as the trend toward the integration of telephony and data networks into integrated services networks gains momentum. It is expected that these integrated services networks will include wireless and mobile environments as well as wired ones. As a consequence of the rapid development, reduced time to market, fusion of communication technologies and rapid growth of the Internet, predicting network performance, and eliminating protocol faults have become an extremely difficult task. Attempts to predict and extrapolate the network performance in small-scale experimental testbeds may yield incomplete or contradictory outcomes. Application of analytical methods is also not feasible due to the complexity of the protocol interactions, analytical intractability and size [1]. For large scale analysis in both the spatial and temporal domain, accurate and detailed models using parallel simulation techniques offer a practical answer. It should be noted that simulation is now considered as a tool of equal importance and complementary to the analytical and experimental studies for investigating and understanding the behavior of various complex systems such as climate research, evolution of solar system and modeling nuclear explosions.

This paper reports about the results of implementing parallel simulation support in the OMNeT++ discrete event simulation tool [17]. The project has been motivated by and forms part of our ongoing research programs at CTIE, Monash University on the analysis of protocol performance of large-scale mobile IPv6 networks. We have developed a set of OMNeT++ models for accurate simulation of IPv6 protocols [7]. We are now focusing our efforts to simulate mobile IPv6 networks in very large scale. For this purpose, we intend to use the computational capacity of APAC (http://www.vpac.org) and VPAC (http://www.apac.edu.au) supercomputing clusters. In a series of future articles, we will be reporting our related research on synchronization methods, efficient topology partitioning for parallel simulation, and topology generation for mobile/wireless/cellular Internet.

II. PARALLEL SIMULATON OF COMMUNICATION NETWORKS TODAY

Discrete event simulation of telecommunications systems is generally a computation intensive task. A single run of a wireless network model with thousands of mobile nodes may easily take several days and even weeks to obtain statistically trustworthy results even on today's computers, and many simulation studies require several simulation runs [1]. Independent replicated simulation runs have been proposed to reduce the time needed for a simulation study, but this approach is often not possible (for example, one simulation run may depend on the results of earlier runs as input) or not practical. Parallel discrete event simulation (PDES) offers an attractive alternative. By distributing the simulation over several processors, it is possible to achieve a speedup compared to sequential (one-processor) simulation. Another motivation for PDES is distributing resource demand among several computers. A simulation model often exceeds the memory limits of a single workstation. Even though distributing the model over several computers and controlling the execution with PDES algorithms may result in slower execution than on a single workstation (due to communication and synchronization overhead in the PDES mechanism), but at least it is possible to run the model.

It is a recent trend that clusters (as opposed to shared memory multiprocessors) are becoming an attractive PDES platform [12], mainly because of their excellent price/performance ratio. Also, very large-scale network simulations demand computing capacity that can only be provided with cluster computing at affordable costs.

Despite about 15-20 years on research on parallel discrete event simulation (see e.g. [3]), PDES is today still more of a promise than part of everyday practice. Fujimoto, a PDES veteran [4], recently expressed this as: "Parallel simulation provides a benefit, but it has to be transparent, automatic, and virtually free in order to gain widespread acceptance. Today it ain't. It may never be." [5]

What parallel simulation tools are available today for the communication networks research community? A parallel simulation extension for the traditionally widely used ns2 simulator has been created at the Georgia Institute of Technology [11], but it is not in wide use. SSFNet [15] claims to be a standard for parallel discrete event network simulation. SSFNet's Java implementation is becoming popular in the research community, but SSFNet for C++ (DaSSF) does not seem to receive nearly as much attention, probably due to the lack of network protocol models. J-Sim [6], another popular network simulation environment does not have PDES support. Parsec [1] with its GloMoSim library have morphed into the commercial Qualnet network simulation product [13]. The optimistic parallel simulation tool SPEEDES [14][16] has similarly become commercial, and it is apparently not being used for simulation of communication networks. The best-known commercial network simulation tool, OPNET [10], does support parallel simulation, but little has been disclosed about it. It appears that OPNET simulations can make use of multiprocessor architectures, but cannot run on clusters.

Apparently, the choice is limited for communication networks research groups that intend to make use of parallel simulation techniques on clusters. SSFNet for Java appears to be a feasible choice, but in the C/C++ world there is probably no really attractive choice today. The project effort published in this paper attempts to improve this situation, and there is a good chance that OMNeT++ can fill this niche.

III. Parallel Simulation Support in OMNeT++

A. About OMNeT++

OMNeT++ [17] is a discrete event simulation environment. The primary application area of OMNeT++ is the simulation of communication networks, but because of its generic and flexible architecture, it has been successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well. OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. The distinguishing factors of OMNeT++ are its strongly component-oriented approach which promotes structured and reusable models, and its extensive graphical user interface (GUI) support. Due to its modular architecture, the OMNeT++ simulation kernel (and models) can be easily embedded into your applications. OMNeT++ is open-source and free for academic and non-profit use.

An OMNeT++ model consists of modules that communicate with message passing. The active modules are termed simple modules; they are written in C++, using the simulation class library. Simple modules can be grouped into compound modules. Both simple and compound modules are instances of module types. While describing the model, the user defines module types; instances of these module types serve as components for more complex module types. Finally, the user creates the system module as an instance of a previously defined module type.

Modules communicate with messages which – in addition to usual attributes such as timestamp – may contain arbitrary data. Simple modules typically send messages via gates, but it is also possible to send them directly to their destination modules.

Gates are the input and output interfaces of modules: messages are sent out through output gates and arrive through input gates. An input and an output gate can be linked with a connection. Connections are created within a single level of module hierarchy: within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected.

Due to the hierarchical structure of the model, messages typically travel through a chain of connections, to start and arrive in simple modules. Compound modules act as "cardboard boxes" in the model, transparently relaying messages between their inside and the outside world. Connections can be assigned properties such as propagation delay, data rate and bit error rate.

B. PDES Features

This section introduces the new PDES architecture in OMNeT++ (OMNeT++ has had some support for parallel simulation before, but it was sufficient only for experimental purposes). In its current form it supports conservative synchronization via the classic Chandy-Misra-Bryant (or Null Message) Algorithm [3].

The OMNeT++ design places a big emphasis on *separation of models from experiments*. The main rationale is that usually a large number of simulation experiments need to be done on a single model before a conclusion can be drawn about the real system. Experiments tend to be ad-hoc and change much faster than simulation models, thus it is a natural requirement to be able to carry out experiments without changing the simulation model itself.

Following the above principle, OMNeT++ allows simulation models to be executed in parallel without modification. No special instrumentation of the source code or the topology description is needed, as partitioning and other PDES configuration is entirely described in the configuration files (in contrast, ns2 requires modification of the Tcl code, and SSFNet requires modification of the DML file(s)).

OMNeT++ supports the Null Message Algorithm (NMA) with static topologies, using link delays as lookahead. The

laziness of null message sending can be tuned. Also supported is the Ideal Simulation Protocol (ISP) introduced by Bagrodia in 2000 [2]. ISP is a powerful research vehicle to measure the efficiency of PDES algorithms, optimistic or conservative; more precisely, it helps determine the maximum speedup achievable by any PDES algorithm for a particular model and simulation environment. In OMNeT++, ISP can be used for benchmarking the performance of the NMA. Additionally, models can be executed without any synchronization, which can be useful for educational purposes (to demonstrate the need for synchronization) or for simple testing.

For the communication between logical processes (LPs), OMNeT++ primarily uses MPI, the Message Passing Interface standard [9]. An alternative communication mechanism is based on named pipes, for use on shared memory multiprocessors without the need to install MPI. Additionally, a file system based communication mechanism is also available. It communicates via text files created in a shared directory, and can be useful for educational purposes (to analyze or demonstrate messaging in PDES algorithms) or to debug PDES algorithms. Implementation of a shared memory-based communication mechanism is also planned for the future, to fully exploit the power of multiprocessors without the overhead of and the need to install MPI.

Nearly every model can be run in parallel. The constraints are the following:

- modules may communicate via sending messages only (no direct method call or member access) unless mapped to the same processor
- no global variables
- there are some limitations on direct sending (no sending to a *sub*module of another module, unless mapped to the same processor)
- lookahead must be present in the form of link delays
- currently static topologies are supported

PDES support in OMNeT++ follows a modular and extensible architecture. New communication mechanisms can be added by implementing a compact API (expressed as a C++ class) and registering the implementation – after that, the new communications mechanism can be selected in the configuration file.

New PDES synchronization algorithms can be added in a similar way. PDES algorithms are also represented by C++ classes that have to implement a compact API to integrate with the simulation kernel. Setting up the model on various LPs as well as relaying model messages across LPs is already taken care of and not something the implementation of the synchronization algorithm needs to worry about it (although it can intervene if needed, because the necessary hooks are present).

The implementation of the NMA is also modular in itself in that a lookahead discovery mechanism can be plugged in via a defined API. Currently implemented lookahead discovery uses link delays, but it is possible to implement more sophisticated ones and select them through the configuration file.

C. *Parallel Simulation Example*

For demonstrating PDES capabilities of OMNeT++, we use the closed queuing network (CQN) model described in [2]. The model consists of *N* tandem queues where each tandem consists of a switch and *k* single-server queues with exponential service times (Figure 1). The last queues are looped back to their switches. Each switch randomly chooses the first queue of one of the tandems as destination, using uniform distribution. The queues and switches are connected with links that have nonzero propagation delays. Our OMNeT++ model for CQN wraps tandems into compound modules.

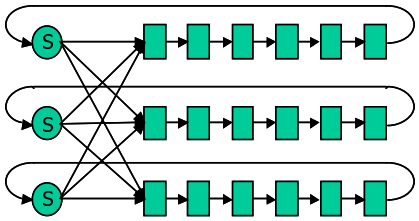

Figure 1. The Closed Queueing Network (CQN) model

To run the model in parallel, we assign tandems to different LPs (Figure 2). Lookahead is provided by delays on the marked links.

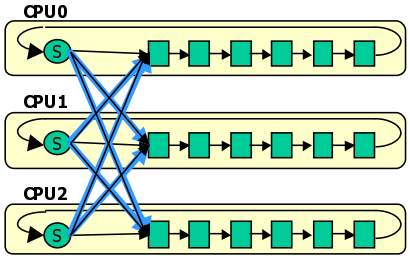

Figure 2. Partitioning the CQN model

To run the CQN model in parallel, we have to configure it for parallel execution. In OMNeT++, the configuration is in a text file called `omnetpp.ini`. For configuration, first we have to specify partitioning, that is, assign modules to processors. This is done with the following lines:

```
[Partitioning]
*.tandemQueue[0].partition-id = 0
*.tandemQueue[1].partition-id = 1
*.tandemQueue[2].partition-id = 2
```

The numbers after the equal sign identify the LP. Also, we have to select the communication library and the parallel simulation algorithm, and enable parallel simulation:

```
[General]
parallel-simulation=true
parsim-communications-class="cMPICommunications"
parsim-synchronization-class="cNullMessageProtocol"
```

When the parallel simulation is run, LPs are represented by multiple running instances of the same program. When using LAM-MPI [8], the mpirun program (part of LAM-MPI) is used to launch the program on the desired processors. When named pipes or file communications is selected, the opp_prun

OMNeT++ utility can be used to start the processes. Alternatively, one can launch the processes manually:

```
./cqn -p0,3 &
./cqn -p1,3 &
./cqn -p2,3 &
```

Here, the `-p` flag tells OMNeT++ the index of the given LP and the total number of LPs. For PDES, one will usually want to select the command-line user interface of OMNeT++, and redirect the output to files (OMNeT++ provides the necessary configuration options.)

The GUI of OMNeT++ can also be used (as evidenced by Figure 3), independent of the selected communication mechanism. The GUI interface can be useful for educational or demonstration purposes as OMNeT++ shows the operation of NMA in a log window, and one also can examine EIT and EOT values.

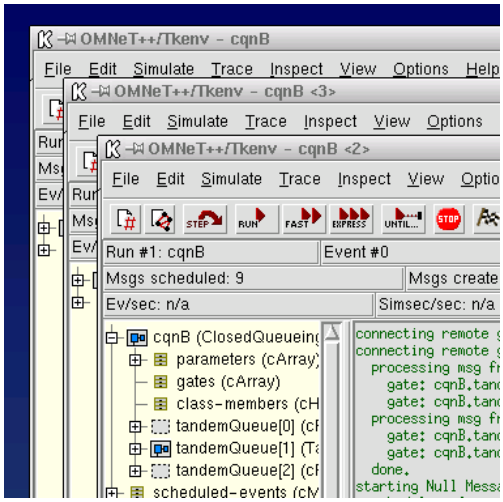

Figure 3. Screenshot of CQN running in three LPs

D. *Instantiation of Modules*

When setting up a model partitioned to several LPs, OMNeT++ uses placeholder modules and proxy gates. In the local LP, placeholders represent sibling submodules that are instantiated on other LPs. With placeholder modules, every module has all of its siblings present in the local LP – either as placeholder or as the "real thing". Proxy gates take care of forwarding messages to the LP where the module is instantiated (see Figure 4).

The main advantage of using placeholders is that algorithms such as topology discovery embedded in the model can be used with PDES unmodified. Also, modules can use direct message sending to any sibling module, including placeholders. This is so because the destination of direct message sending is an input gate of the destination module, thus if the destination module is a placeholder, the input gate will be a proxy gate which transparently forwards the messages to the LP where the "real" module was instantiated. A limitation is that the destination of direct message sending cannot be a *submodule* of a sibling (which is probably a bad practice anyway, as it violates encapsulation), simply because placeholders are empty and so its submodules are not present in the local LP.

Instantiation of compound modules is slightly more complicated. Since its submodules can be mapped to different LPs, the compound module may not be "fully present" on any given LP, and it may forced to be present on several LPs (on all LPs where if one or more submodules instantiated). Thus, compound modules are instantiated wherever they have at least one submodule instantiated, and are represented by placeholders everywhere else (Figure 5).

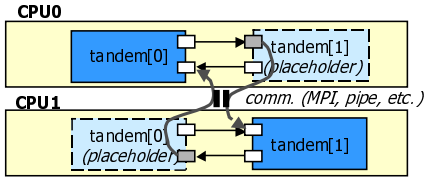

Figure 4. Placeholder modules and proxy gates

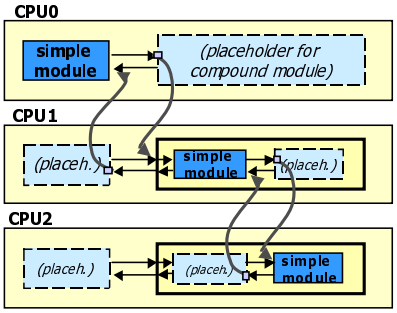

Figure 5. Instantiating compound modules

E. *Performance Measurements*

We have made several runs with the CQN model on 2 and 4 processors, with the following parameters: $N=16$ tandem queues, $k=10$ and 50 queues per tandem, with lookahead $L=1$, 5 and 10. The hardware environment was a Linux cluster (kernel 2.4.9) of dual 1 Ghz Pentium III PCs, interconnected using a 100Mb Ethernet switch. The communication library was LAM-MPI [8]. The MPI latency was measured to be 22µs. Sequential simulation of the CQN model achieved P_{seq}=120,000 events/sec performance.

We executed simulations under NMA and (for comparison) under ISP. The results are summarized in Table I. P_{ISP}, P_{NMA} are the performance (events/second) under the ISP and the NMA protocol, and S_{ISP}, S_{NMA} are the speedups under ISP and NMA, respectively. It can be observed that the L lookahead strongly affects performance under NMA. An analysis of NMA performance versus lookahead and other performance factors can be found in [18]. However, it is probably too early to draw conclusions from the figures below about the performance of the OMNeT++ parallel simulation implementation, because we are still optimizing the code.

TABLE I. COMPARISON OF NMA AND ISP SIMULATIONS

Configuration			Performance			
#LPs	*k*	*L(s)*	P_{ISP} *(ev/s)*	P_{NMA} *(ev/s)*	S_{ISP}	S_{NMA}
2	10	1	147,618	76,042	1.23	0.63
2	10	5	151,250	143,289	1.26	1.19
2	10	20	157,200	153,600	1.31	1.28
2	50	1	168,830	131,398	1.41	1.09
2	50	5	170,289	164,563	1.42	1.37
2	50	20	172,811	173,249	1.44	1.44
4	10	1	300,479	45,190	2.50	0.38
4	10	5	311,392	148,007	2.59	1.23
4	10	20	314,892	271,648	2.62	2.26
4	50	1	359,517	144,979	3.00	1.21
4	50	5	364,663	284,978	3.04	2.37
4	50	20	372,844	352,557	3.11	2.94

IV. DESIGN OF PDES SUPPORT IN OMNET++

Design of PDES support in OMNeT++ follows a layered approach, with a modular and extensible architecture. The overall architecture is depicted in Figure 6.

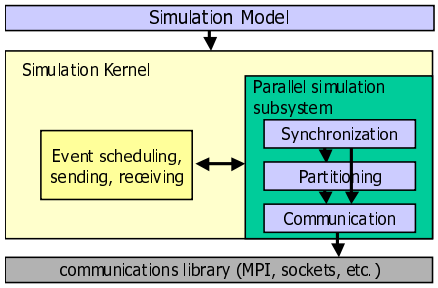

Figure 6. Architecture of OMNeT++ PDES implementation

The parallel simulation subsystem is an optional component itself, which can be removed from the simulation kernel if not needed. It consists of three layers, from the bottom up: communication layer, partitioning layer and synchronization layer.

The purpose of the *Communications Layer* is to provide elementary messaging services between partitions for upper layer. The services include send, blocking receive, non-blocking receive and broadcast. The send/receive operations work with *buffers*, which encapsulate packing and unpacking operations for primitive C++ types. The message class and other classes in the simulation library can pack and unpack themselves into such buffers. The Communications Layer API is defined in the `cFileCommunications` interface (abstract class); concrete implementations like the MPI one (`cMPICommunications`) subclass from this, and encapsulate MPI send/receive calls. The matching buffer class `cMPICommBuffer` encapsulates MPI pack/unpack operations.

The *Partitioning Layer* is responsible for instantiating modules on different LPs according to the partitioning specified in the configuration, for configuring proxy gates. During the simulation, this layer also ensures that cross-partition simulation messages reach their destinations. It intercepts messages that arrive at proxy gates, and transmits them to the destination LP using the services of the communication layer. The receiving LP unpacks the message and injects it at the gate pointed to be the proxy gate. The implementation basically encapsulates the `cParsimPartition`, `cPlaceholderModule` and `cProxyGate` classes.

The *Synchronization Layer* encapsulates the parallel simulation algorithm. Parallel simulation algorithms are also represented by classes, subclassed from the `cParsimSynchronizer` abstract class. The parallel simulation algorithm is invoked on the following hooks: event scheduling, processing model messages outgoing from the LP, and messages (model messages or internal messages) arriving from other LPs. The first hook, event scheduling is a function invoked by the simulation kernel to determine the next simulation event; it also has full access to the future event list (FEL) and can add/remove events for its own use. Conservative parallel simulation algorithms will use this hook to block the simulation if the next event is unsafe, e.g. the null message algorithm implementation (`cNullMessageProtocol`) blocks the simulation if an EIT has been reached until a null message arrives (see [2] for terminology); also it uses this hook to periodically send null messages. The second hook is invoked when a model message is sent to another LP; the NMA uses this hook to piggyback null messages on outgoing model messages. The third hook is invoked when any message arrives from other LPs, and it allows the parallel simulation algorithm to process its own internal messages from other LPs; the NMA processes incoming null messages here.

The null message protocol implementation itself is modular as it employs a separate, configurable lookahead discovery object. Currently only link delay based lookahead discovery has been implemented, but it is possible to implement more sophisticated ones.

The ISP implementation, in fact, consists of two parallel simulation protocol implementations: the first one is based on the NMA and additionally records the external events (events received from other LPs) to a trace file; the second one runs the simulation using the trace file to find out which events are safe and which are not.

Note that although we implemented a conservative protocol, the provided API itself would allow implementing optimistic protocols, too. The parallel simulation algorithm has access to the executing simulation model, so it could perform saving/restoring model state if the code of the simulation model supports this (unfortunately, support for state saving/restoration needs to be individually and manually added to each class in the simulation, including user-programmed simple modules).

We also expect that because of the modularity, extensibility and clean internal interfaces of the parallel simulation subsystem, the OMNeT++ framework has the potential to become a preferred platform for PDES research.

V. CONCLUSION

The paper presented a new parallel simulation architecture for OMNeT++. A merit of the implementation is that it features the "separation of experiments from models" principle, and thus allows simulation models to be executed in parallel without modification. It relies on a novel approach of

placeholders to instantiate the model on different LPs. The placeholder approach allows simulation techniques such as topology discovery and direct message sending to work unmodified with PDES. The architecture is modular and extensible so it may serve as a potential framework for research on parallel simulation.

REFERENCES

[1] R. L. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Y. Song. Parsec: A parallel simulation environment for complex systems. IEEE Computer, pages 77–85, October 1998.

[2] R. L. Bagrodia and M. Takai. Performance evaluation of conservative algorithms in parallel simulation languages. IEEE Transactions on Parallel and Distributed Systems, 11(4):395–414, 2000.

[3] M. Chandy and J. Misra. Distributed simulation: A case study in design and verication of distributed programs. IEEE Transactions on Software Engineering SE-5, (5). 440–452, 1979.

[4] R. M. Fujimoto. Parallel discrete event simulation. Communications of the ACM, 33(10):30–53, October 1990.

[5] R. M. Fujimoto. Parallel and distributed simulation in the 21th century. In Grand Challenges for Modeling and Simulation (Seminar 02351), 26-30 August 2002, Dagstuhl Castle, Germany, 2002.

[6] J-Sim home page. http://www.j-sim.org.

[7] J. Lai, E. Wu, A. Varga, Y. A. Şekercioğlu, and G. K. Egan. A simulation suite for accurate modeling of IPv6 protocols. In Proceedings of the 2nd OMNeT++ Workshop, pages 2–22, Berlin, Germany, January 2002.

[8] LAM-MPI home page. http://www.lam-mpi.org/.

[9] MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4):165–414, 1994. Message Passing Interface Forum.

[10] OPNET. OPNET Technologies, Inc. home page. http://www.opnet.com/.

[11] Atlanta PADS Research Group, Georgia Institute of Technology. PDNS - Parallel/Distributed NS home page. http://www.cc.gatech.edu/computing/compass/pdns.

[12] C. D. Pham. High performance clusters: A promising environment for parallel discrete event simulation. In Proceedings of the PDPTA'99, June 28-July 1, 1999, Las Vegas, USA, 1999.

[13] QualNet home page. http://www.qualnet. com

[14] SPEEDES home page. http://www.speedes.com

[15] SSFNet home page. http://www.ssfnet.org

[16] J. Steinman. Scalable parallel and distributed military simulations using the SPEEDES framework. ELECSIM'95, 2nd Electronic Simulation Conference, Internet, May-June, 1995.

[17] A. Varga. The OMNeT++ discrete event simulation system. In Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic, 2001.

[18] A. Varga, Y. A. Şekercioğlu, and G. K. Egan. A practical efficiency criterion for the null message algorithm. In Proceedings of the European Simulation Symposium (ESS2003), Oct. 2003, Delft, The Netherlands. Society for Computer Simulation, 2003.

VI. AUTHOR BIOGRAPHIES

András Varga received his M.Sc. in computer science with honors from the Technical University of Budapest, Hungary in 1994. He is the author of the OMNeT++ open-source network simulation tool currently widely used in academic and industrial settings. He worked for several years as software architect for Encorus (formerly Brokat Technologies) before founding OpenSim Ltd that develops OMNeT++, and Simulcraft Inc. that provides commercial licenses and services for OMNeT++ worldwide. He is currently working towards PhD, his research topic being large-scale simulation of communication networks. Between February and September 2003, and between February and June 2005 he visited CTIE at Monash University (Melbourne, Australia) to participate in parallel simulation research.

Y. Ahmet Şekercioğlu is a researcher at the Centre for Telecommunications and Information Engineering (CTIE) and a Senior Lecturer at Electrical and Computer Systems Engineering Department of Monash University, Melbourne, Australia. Between 1998 and 2006 he also held the position of Program Leader for the Applications Program of Australian Telecommunications Cooperative Research Centre (ATcrc, http://www.atcrc.com). He completed his PhD degree at Swinburne University of Technology, Melbourne, Australia (2000), MSc (1985) and BSc (1982) degrees at Middle East Technical University, Ankara, Turkey (all in Electrical Engineering). He has lectured at Swinburne University of Technology for 8 years, and has had numerous positions as a research engineer in private industry. His recent work focuses on development of tools for simulation of large-scale telecommunication networks. He is also interested in application of intelligent control techniques for multiservice networks as complex, distributed systems.