A performance comparison of recent network simulators

Elias Weingärtner, Hendrik vom Lehn and Klaus Wehrle
Distributed Systems Group
RWTH Aachen University
Aachen, Germany
Email: {weingaertner,vomlehn,wehrle}@cs.rwth-aachen.de

***Abstract*—A widespread methodology for performance analysis in the field of communication systems engineering is network simulation. While ns-2 has established itself as virtually the standard network simulation tool, other network simulators have gained more and more attention during the last years. In this paper, we briefly survey new developments in the field of network simulation and conduct a performance comparison study by implementing an identical simulation set-up in five simulators, namely ns-2, OMNet++, ns-3, SimPy and JiST/SWANS. Our results reveal large differences according to both run-time performance and memory usage.**

I. Introduction

Network simulation is without a doubt one of the most predominant evaluation methodologies in the area of computer networks. It is widely used for the development of new communication architectures and network protocols. So-called *network simulators* allow one to model an arbitrary computer network by specifying both the behavior of the network nodes and the communication channels. For example, in order to investigate the characteristics of a new routing protocol, it is usually implemented in a network simulator. Afterwards, the routing behavior can be easily studied in different topologies, given the fact that the network topology is merely a set of simulation parameters. Most available network simulation toolkits are based on the paradigm of discrete event-based simulation [5] (DES). Here, the simulated network nodes trigger events, for instance, when a packet is sent to another node. The simulator maintains an event queue sorted by the scheduled event execution time. The simulation itself is performed by successively processing the events in the queue.

The first approaches where DES was applied to the simulation of computer networks were published about two decades ago [4, 8]. ns-2 [13] is a direct successor of those early efforts and since then, it has become virtually the standard for network simulation. This can be attributed to the fact that numerous models, e.g. protocol models and traffic generators, are publicly available for ns-2. They can be used off-the-shelf, thus eliminating the need of implementing them by hand. However, a major shortcoming of ns-2 is its limited scalability [6, 20] in terms of memory usage and simulation run-time. This is especially a problem as new research domains in the field of computer networks, such as wireless sensor networks (WSNs), peer-to-peer networks or grid architectures, require the simulation of very large networks, potentially with hundreds of thousands of nodes. In order to face those challenges, a couple of enhancements of ns-2 have been proposed, for instance the incorporation of parallelization [16]. However, ns-2 is currently undergoing a major redesign [6]. One of the main development goals of its successor, ns-3, is the improvement of simulation performance.

Besides ns-2, over a dozen network simulators are presently used in academia and in the industry. Prominent examples include OMNeT++ [19], the Java-based JiST [3] and commercial tools such as the OPNet modeler [14]. In addition, specialized simulation tools, such as the WSN simulator TOSSIM [9], serve dedicated research domains. This leaves many researchers and graduate students with the question of which network simulator to use, especially if one is interested in achieving a high simulation performance.

In this paper, we focus on current developments regarding open source simulators. In Section II, we provide a brief overview of network simulators which have recently gained attention in the research community. The main contribution of this paper is a performance comparison study incorporating five different open source simulation tools, namely ns-2 [13], ns-3 [6], OMNeT++ [19], JiST [3] and SimPy [11]. By implementing the same simulation and equal simulation models from scratch for all of them, we are able to compare the simulator performance itself without any distortions caused by different implementations of simulation models. The design and the outcome of this performance comparison study are discussed in Section III. As a matter of fact, this is not the first performance comparison of network simulators. However, to our knowledge none of those studies include recent contributions like ns-3. In Section IV, we discuss such related performance evaluation studies and compare their results with the ones presented in this paper. We conclude in Section V with the lessons we learned from this performance comparison.

II. Investigated Simulation Tools

In this section, we concisely introduce the network simulators considered in the performance comparison. We emphasize that ns-3, OMNeT++ and JiST are all gaining more and more prevalence compared to the long-established ns-2. We include ns-2 here to form a baseline. In addition, SimPy was incorporated in the comparison as it represents a modern implementation of a process-oriented simulator in the popular Python language.

A. *ns-2*

Network simulations for ns-2 are composed of C++ code, which is used to model the behavior of the simulation nodes, and oTcl scripts that control the simulation and specify further aspects, for instance the network topology. This design choice was originally made to avoid unnecessary recompilations if changes are made to the simulation set-up [6]. Back in 1996 when the first version of ns-2 was released, this was a reasonable intent, as the frequent recompilation of C++ programs was indeed time-consuming and slowed down the research cycle. However, from today's perspective, the design of ns-2 trades off simulation performance for the saving of recompilations, which is questionable if one is interested in conducting scalable network simulations.

B. *ns-3*

Like its predecessor, ns-3 relies on C++ for the implementation of the simulation models. However, ns-3 no longer uses oTcl scripts to control the simulation, thus abandoning the problems which were introduced by the combination of C++ and oTcl in ns-2. Instead, network simulations in ns-3 can be implemented in pure C++, while parts of the simulation optionally can be realized using Python as well. Moreover, ns-3 integrates architectural concepts and code from GTNetS [17], a simulator with good scalability characteristics. These design decisions were made at expense of compatibility. In fact, ns-2 models need to be ported to ns-3 in a manual way. Besides performance improvements, the feature set of the simulator is also about to be extended. For example, ns-3 is slated to support the integration of real implementations' code by providing standard APIs, such as Berkeley sockets or POSIX threads, which are transparently mapped to the simulation [1].

C. *OMNeT++*

In contrast to ns-2 and ns-3, OMNeT++ is not a network simulator by definition, but a general purpose discrete event-based simulation framework. Yet it is mostly applied to the domain of network simulation, given the fact that with its INET package it provides a comprehensive collection of Internet protocol models. In addition, other model packages such as the OMNeT++ Mobility Framework and Castalia [15] facilitate the simulation of mobile ad hoc networks or wireless sensor networks.

OMNeT++ simulations consist of so-called *simple modules* which realize the atomic behavior of a model, e.g. a particular protocol. Multiple simple modules can be linked together and form a *compound module*. For instance, multiple simple modules which provide protocol models can be combined into a compound module representing a host node. A network simulation in OMNeT++ is implemented itself as a compound module which comprehends other compound modules, like the ones which model host nodes.

Like the aforementioned ns-2 and ns-3, OMNet++ rests upon C++ for the implementation of simple modules. However, the composition of these simple modules into compound modules and thus the set-up of network simulation takes place in NED, the network description language of OMNeT++. NED is transparently rendered into C++ code when the simulation is compiled as a whole. Moreover, NED supports the specification of variable parameters in the network description: For example the number of nodes in a network can be marked to be dynamic and later on be configured at runtime. In this case, the modules representing the nodes are dynamically instantiated by the simulator during execution. This feature is a direct consequence of the simulator's strict object-oriented design.

D. *JiST*

A fresh approach to network simulation is JiST ("Java in Simulation Time"), which in compliance with its name allows the implementation of network simulations in standard Java. It is mostly used in conjunction with SWANS[1], a simulator for mobile ad hoc networks built on top of JiST.

Network simulations in JiST are made up of entities which represent the network elements, for example nodes, with simulation events being formed by method invocations among those entities. The entities advance the simulation time independently by notifying the simulation core. While the code inside an entity is executed like any arbitrary Java program, only the interactions between the individual entities are carried out in simulation time. Thus, these interactions between entities correspond to synchronization points and facilitate the parallel execution of code at different entities, resulting in a potential performance gain. In order to execute the implementation in simulation time, JiST utilizes a custom dynamic Java class loader which dynamically rewrites the application's byte code.

Unfortunately, the official development of JiST has stalled, as it is no longer maintained by its original author, Rimon Barr. However, a couple of enhancements and improvements have recently been released by Ulm University[2]. We incorporate those enhancements in our performance analysis of JiST in Section III.

E. *SimPy*

With SimPy, we include a process-oriented discrete-event simulator in this performance comparison. Unlike the other simulators, no public available network models exist for SimPy. Instead, it is a bare simulation API written in Python. In SimPy, the basic simulation entities are processes. They are executed in parallel and may exchange Python objects among each other. Most processes include an infinite loop in which the main actions of the process are performed. Besides abstractions for processes and the related exchange of objects, SimPy provides instructions for the synchronization of simulation processes and commands for the monitoring of simulation data.

[1]JiST/SWANS Website: http://jist.ece.cornell.edu/
[2]Ulm University's JiST portal: http://www.vanet.info/jist-swans/

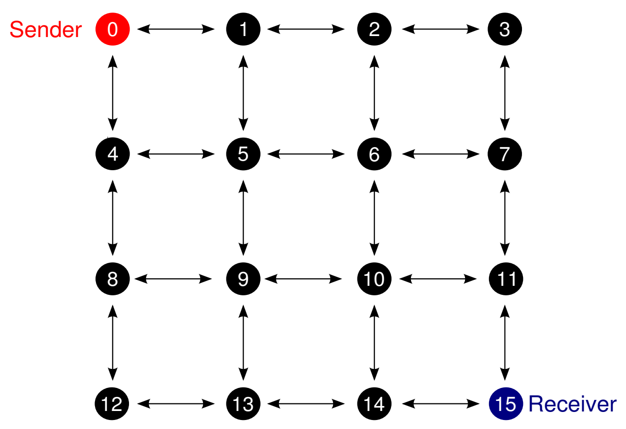

Fig. 1. Sample Network Topology (size=16)

III. Performance Comparison

This section describes the methodology and the outcome of the performance study which includes the simulation tools introduced in the previous section. The comparison is based on a benchmark scenario and discloses large differences according to simulation run-time and memory usage.

With the goal of comparing the simulators' core performance, we first implemented a reference simulation in all simulation toolkits from scratch[3]. Our benchmarking simulation does not rely on any existing simulation model for any simulator. This decision was made because a network simulation's performance is largely dependent on the code of the network models and their computational complexity.

The reference simulation models a basic network, where the nodes are arranged in a square topology as illustrated in Figure 1. One sending node generates one packet every second and broadcasts it to its neighbors. The neighboring nodes relay unseen messages after a delay of one second, thus flooding the entire network. The propagation delay is directly implemented by delaying the simulation events' execution, and the nodes do not implement a explicit queueing policy. With a fixed probability which is equal on every link, packets are dropped on the channel. The receiver is located at the corner opposite to the receiver. We chose this simulation scenario for its simplicity, not aiming at a simulation of a real network.

All simulation runs were conducted on a AMD Athlon 64 3800+ workstation with 2GB of RAM, running Ubuntu Linux 8.04 LTS. Our measurements were taken using ns-2 version 2.33, OMNeT 3.4b2, ns-3.1, SimPy 1.9.1 and JiST 1.06 with the extensions from Ulm University. We made use of SUN Java 1.5.0.11 for the execution of JiST, and SimPy was run with Python 2.5.1.

A. *Model equality*

As we implemented the same simulation set-up in five different simulators, we first checked if our implementations yield results which are on par with each other. For this purpose, we ran the simulation in all simulators for drop probabilities

[3]The respective source code is available at http://ds.cs.rwth-aachen.de/research/projects/simcompare/

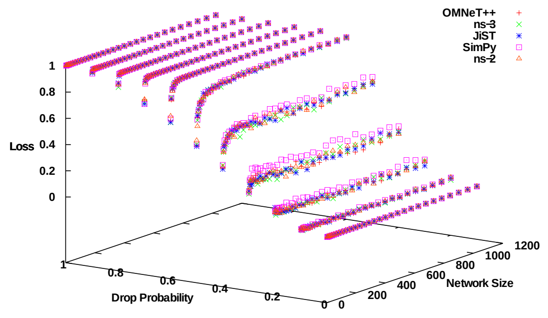

Fig. 2. End-to-End Packet Loss

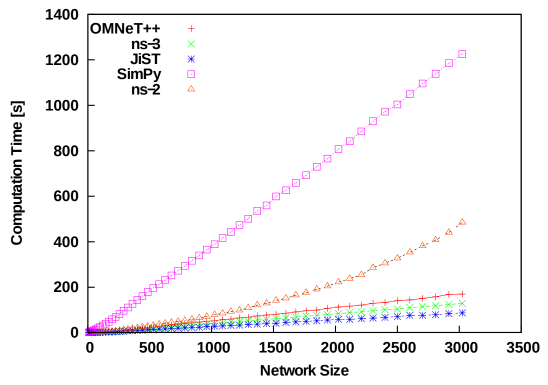

Fig. 3. Simulation runtime vs. Network size

between 0 and 1 with the square topology size ranging from 4 to 1024 nodes. The simulation time was set to 600 seconds. Figure 2 depicts the end-to-end packet loss retrieved from the five simulations, given the drop probability and a network size. Only SimPy produces slightly higher loss rates on average, however still within the limits of tolerance. From these results, we conclude that our independent implementations of the reference simulation in fact produce equivalent results.

B. *Performance comparison*

Given the fact that our simulation set-ups produce equal results, we now compare the individual simulation tools regarding two performance metrics: effective **simulation runtime** and **memory usage**. In order to evaluate the simulators' scalability, we conducted two series of different runs using the reference simulation. In the first series, the drop probability is set to a fixed value of 0.10 with the network size ranging from 4 to 3025 nodes. The second series uses a fixed network size of 3025 nodes, given drop probabilities between 0.0 and 1.0. All results provided in the following are averages over five executions of each simulation series. In both series, the simulation time was set to 600 seconds.

1) Simulation run-time: Figure 3 shows the measured **simulation runtime** at different network sizes for the compared simulation tools. First of all, these results reveal that SimPy does not scale well and hence is not applicable to large-scale network simulations: For a network size of 3025 nodes, it needs 1225 seconds on average to complete the simulation run. In contrast to that, JiST finishes the same task about 14 times faster, resulting in an average execution time of 86 seconds.

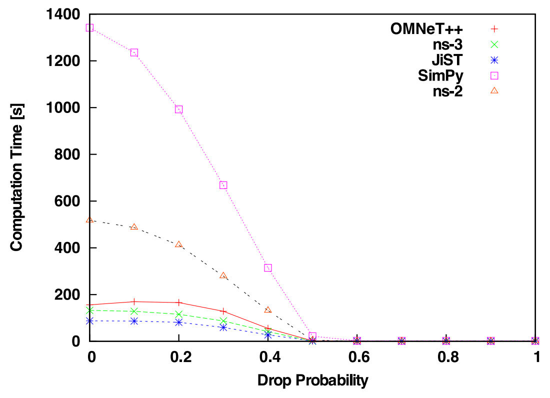

Fig. 4. Simulation run-time vs. Drop probability

The overall run-time performance of JiST is astonishing at the first glance, given the fact that it is based on Java and still outperforms OMNeT++ and ns-3, which are executed in a native manner. We attribute this winning margin to the architecture of JiST: Besides the parallel execution at different entities, JiST performs different run-time optimizations based on the analysis of the executed byte code. Moreover, it has been shown that the slowness of Java is merely a myth, and that recent Java run-time environments can keep up with the execution speed of compiled C++ code [10]. According to the run-time performance of ns-3, the architectural improvements, especially the abolishment of the oTCL/C++ duality, are clearly reflected in our results, as ns-3 is considerably faster than its predecessor. While the run-time performance of OMNeT++ is slightly inferior to ns-3 and JiST, all three simulation tools exhibit almost the same scalability according to simulation run-time.

Additional insight about the run-time behavior of the different simulators can be derived from the results in Figure 4. Here, we picture the averaged run-time from the second simulation series for a fixed network size of 3025 nodes and a varying drop probability. With increasing drop probabilities, the simulation run-time naturally decreases in a quick fashion for all simulators, as more and more packets are removed from the simulation, thus resulting in fewer events to be processed. In other words, the drop probability directly reflects the quantity of events prevalent in the simulation. We notice that SimPy's simulation run-time increases much faster at low drop probabilities than any one of the other simulators. From our results we conclude that SimPy in fact has a lower event throughput than the other simulation cores.

2) Memory usage: Similar to our analysis of the simulation run-time, we measured the maximum **memory usage** of the individual simulators during the two series of simulation runs. The outcome is depicted in Figure 5. Surprisingly, JiST uses up much more memory resources than the other simulation tools. We first attributed this behavior to the garbage collection mechanism, but the amount of used memory does not decrease if the garbage collection is manually triggered at times. In addition, the difference in memory usage between JiST and the other tools increases at larger network sizes, and hence, the

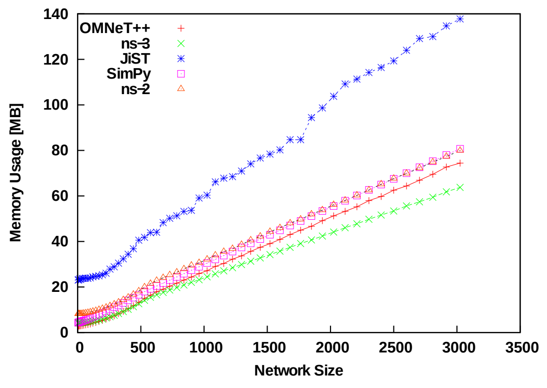

Fig. 5. Memory usage vs. Network size

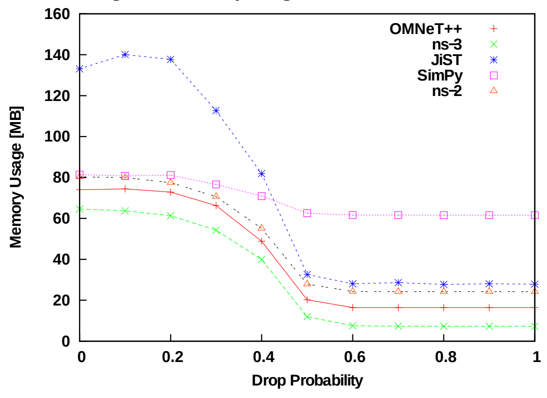

Fig. 6. Memory usage vs. Drop probability

additional memory requirements of the Java virtual machine do not suffice as single explanation. The memory usage performances of ns-2, OMNeT++ and SimPy share a similar linear growth of memory usage, with ns-3 being the most efficient simulation tool in this regard.

Figure 6 shows the memory usage measured during the second simulation series. As mentioned earlier, for larger drop probabilities the number of events prevalent in the simulation is considerably small and thus the memory usage here remains almost constant. This almost "constant" memory footprint is mostly constituted by the simulation core and a potential run-time environment, such as the JavaVM in the case of JiST or Python, which is required for the execution of SimPy. According to JiST, the footprint is slightly larger than that of ns-2 and the other simulators, but far smaller than the one of SimPy. However, for lower drop probabilities and hence more simulation events, the memory usage of JiST grows faster than with any other simulation tool.

IV. Related Work

A couple of network simulator performance comparisons have been published in recent years. Most of the more recent ones compare ns-2 with other simulation tools. One example is the work presented in [12], where the performance of a TCP-based reference simulation implemented in ns-2 is checked against SSFNet and JavaSim (now known J-Sim), two older simulators. In their work, the authors also observe large differences regarding memory consumption and simulation run-time, with ns-2 performing best according to computational demands and worst according to memory consumption.

A performance comparison, which in addition to ns-2, also includes earlier versions of SimPy and OMNeT is presented in [2]. The authors only provide results concerning the run-time performance in their paper, and the used simulation is small in terms of network size. However, the outcome is similar to ours, with OMNeT++ outperforming ns-2 and SimPy. Regarding SimPy, the authors also note its sluggish performance.

Two recent publications [7, 18] analyze the characteristics of JiST/SWANS in contrast to ns-2. Unlike our work, which focuses mainly on the performance of the simulation cores, the authors compare ns-2 and JiST/SWANS in a complex simulation of a mobile ad hoc network, using available implementations of routing protocols bundled with both simulators. Given the same parameters, the authors observe that ns-2 and JiST/SWANS require about the same time to finish the simulation run, with ns-2 exhibiting much higher memory demands than JiST in the given scenario. At first sight, this seems to contradict our results presented in Section III. However, this phenomenon can be explained by the fact that the radio models of ns-2 duplicate messages in memory, e.g. if a packet is broadcasted to other nodes. On the contrary, the radio models implemented in SWANS pass solely references to static packet data among the entities, resulting in a much smaller memory consumption. These results, in combination with ours, affirm that the scalability of simulations and related performance matters are in fact heavily influenced by the simulation models.

V. Conclusion

In this paper, we investigated the performance requirements and the scalability of five different simulation tools. Our results show that three of them, ns-3, OMNeT++ and JiST are all capable of carrying out large-scale network simulations in an efficient way. JiST has proven to be the fastest simulator by far in our experiments, however the exhaustive memory consumption may limit its applicability in some simulation scenarios. In our performance comparison, ns-3 demonstrated the best overall performance. Although it was surpassed by JiST in terms of simulation run-time, it still shows both low computational and less memory demands. However, at present ns-3 still is in the early stages, and just a few simulation models exist which one can use off the shelf. As the rich collection of models for ns-2 still needs to be ported from ns-2 to ns-3, OMNeT++ can be considered as viable alternative. While its performance is slightly inferior than that of ns-3 and JiST, over the last few years a very comprehensive set of models has been developed for this simulator. Moreover, OMNeT++ provides a rich graphical user interface and an abstract modeling language, while JiST and ns-3 rely on pure source code for the development of the entire simulation. In conclusion, the question of which simulator to use is a difficult one, and the answer is largely dependent on the specific use case. However, if scalability is the main concern, JiST, ns-3 and OMNeT++ are smart choices.

References

[1] ns-3 Overview (June 2008). http://www.nsnam.org/docs/ns-3-overview.pdf, June 2008.

[2] D. Albeseder and M. Fuegger. Small PC-Network Simulation - a comprehensive performance case study. Research Report 77/2005, TU Wien, Institut für Technische Informatik, 2005.

[3] R. Barr, Z. J. Haas, and R. van Renesse. JiST: an efficient approach to simulation using virtual machines. *Softw, Pract. Exper*, 35(6):539–576, 2005.

[4] A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon. Nest: a network simulation and prototyping testbed. *Commun. ACM*, 33(10):63–74, 1990.

[5] G. S. Fishman. *Principles of Discrete Event Simulation*. John Wiley & Sons, Inc., New York, NY, USA, 1978.

[6] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 13, New York, NY, USA, 2006. ACM.

[7] F. Kargl and E. Schoch. Simulation of manets: a qualitative comparison between JiST/SWANS and ns-2. In *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 41–46, New York, NY, USA, 2007. ACM.

[8] S. Keshav. Real: A network simulator. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1988.

[9] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *In Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).*, 2003.

[10] J. Lewis and U. Neumann. Performance of Java versus C++. http://www.idiom.com/~zilla/Computer/javaCbenchmark.html (accessed 08/25/2008), 2004.

[11] K. Mueller. SimPy documentation. http://simpy.sourceforge.net/discuss.htm.

[12] D. M. Nicol. Scalability of network simulators revisited. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, Orlando, FL, February 2003.

[13] The network simulator ns-2. http://www.isi.edu/nsnam/ns/.

[14] OPNET Technologies Inc. OPNET modeler website. http://www.opnet.com/solutions/network_rd/modeler.html.

[15] H. N. Pham, D. Pediaditakis, and A. Boulis. From simulation to real deployments in wsn and back. *Proceedings of the 2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007).*, pages 1–6, June 2007.

[16] G. Riley. PDNS project website. http://www.cc.gatech.edu/computing/compass/pdns/.

[17] G. Riley. Large scale network simulations with GTNetS. In *Proceedings of the 2003 Winter Simulation Conference*, 2003.

[18] E. Schoch, M. Feiri, F. Kargl, and M. Weber. Simulation of ad hoc networks: ns-2 compared to JiST/SWANS. In *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008')*, March 2008.

[19] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008')*, March 2008.

[20] Y. Xue, H. S. Lee, M. Yang, P. Kumarawadu, H. Ghenniwa, and W. Shen. Performance evaluation of ns-2 simulator for wireless sensor networks. *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE 2007)*, pages 1372–1375, April 2007.