Meta-Simulation of Large WSN on Multi-core Computers

Adnan Iqbal
NUST/SEECS
Islamabad, Pakistan
adnan.iqbal@seecs.edu.pk

Bernard Pottier
UBO/LabSTICC
Brest, France
pottier@univ-brest.fr

Keywords: Meta-Simulation, Wireless Sensor Networks, Distributed Algorithms, Scalability, Parallel Execution

Abstract

With the advances in wireless communications large scale Wireless Sensor Networks (WSN) are emerging with many applications. These networks are deployed to serve single objective application, with high optimization requirements such as performance enhancement and power saving. Application specific optimization is achieved using formal models and evaluation based simulations of distributed algorithms (DA) controlling such networks. The WSN design problem is of high complexity, and requires robust methodologies, including simulation support. Although we know several works on WSN simulation, these solutions fail to match requirements on a whole set of desirable criteria: scalability, flexibility, concurrent execution and performance. We present a model based approach of WSN application specification separating network organization from behaviors, allowing them to vary independently. Through this approach, network description can be achieved with high level tools independently from the programming syntax. We have specified and simulated number of WSN protocols with varying objectives and semantics using a time-driven execution model incorporated in the proposed meta-simulator. The empirical analysis has revealed flexibility, scalability and performance. The tool flow targets an Occam compiler producing efficient multi-threaded binaries. We expect the final flow of this project to enable sensor code production out of the simulated specification.

1. INTRODUCTION

Recent advances in technology have enabled practical deployment of very large scale distributed systems. For instance, Wireless Sensor Networks (WSN)[1] consisting of large number of tiny devices are being used in many applications such as transport management, energy saving, wild-life habitat monitoring, disaster recovery in urban areas and health care [3, 6–8]. Large scale deployments, consisting of thousands of nodes, are being realized in several projects. For instance, San Francisco parking system consists of around six thousand nodes and fire detection network of MIT is also a very large scale network [13].

[1]Figure 1 shows a small WSN with obtained communication links.

These networks are inherently different from other general purpose networks, because these are deployed to serve single objective application. Therefore, a great deal of optimization is expected and possible. Similarly, stringent power requirements make it utmost necessary to optimize power consumptions to the best possible level. It is usually required that guarantees must be provided for certain performance level before actual deployment. This requirement stems from the inability of designers/engineers to replace or modify the network once it is deployed. For example, it may not be possible to replace a fire detection network deployed in a remote forest, the only available option being discarding previous network and to deploy a new one. Given all these constraints, it becomes an unavoidable choice to evaluate every distributed protocol designed for such networks, for all possible scenarios, before deployment. Such evaluations are carried out using mathematical modeling and extensive simulation studies. However, simulation studies for such large scale distributed systems, is a challenging task. For instance, achieving scalability level close to real deployment is fairly complex. This complexity stems from issues like managing simultaneous execution of large number of processes in a scalar machine. Similarly, mimicking actual environment in a simulation is considered hard to realize. A key point is that specifications of process behaviours must strictly be isolated to enforce communications, eliminating false assumptions during algorithm design that could arise from a global knowledge. Furthermore, the simulation mechanism must preserve the isolation property and produce a concurrent execution similar to reality.

A detailed analysis of existing simulators [1, 5, 9–11, 15, 17] suggests that a major reason that these simulators are unable to achieve prescribed objectives, is their difficulties to utilize increasing available parallel processing computing machines. In fact, WSN can be modeled as large scale concurrent systems where all nodes execute protocols in a distributed fashion. Modeling WSN as a concurrent system brings in several advantages in terms of simulation. At first, we obviate the need of a global entity possessing complete knowledge of network. Instead, each node in simulation behaves exactly as a node in real network. Secondly, it is easy to synthesize local behaviors at node level. Thirdly, posing WSN as concurrent systems creates room for parallel execution of simulation.

Multi-core technologies are proliferating and have made

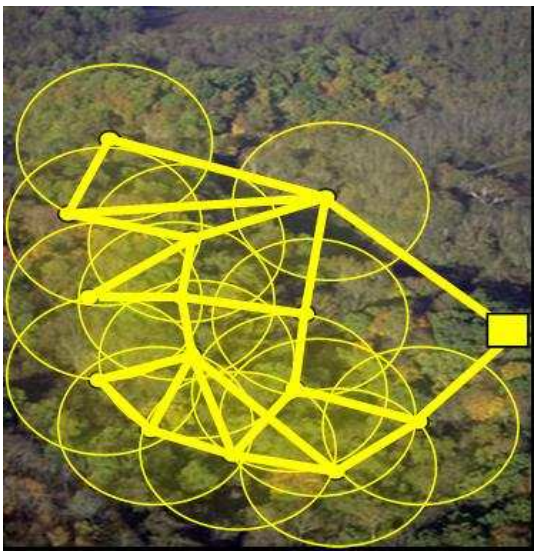

Figure 1. A Typical Sensor Network Connectivity Pattern (Courtesy [2])

inroads into domestic computing systems. Dual core and quad core processors are abundantly available in micro computers. Grid computing is also becoming a mature technique and is being used to solve many scientific computing problems. Many specific purpose devices, for example graphics cards, are based on processor assemblies. Device vendors are developing technology so that these special purpose devices can be used for general purpose computing. CUDA and OpenCL are few such programming paradigms developed to exploit multi-core technology to its maximum potential [14]. FPGA accelerated supercomputers are also appearing making it possible to map distributed processes in hardware.

In spite of easily available multi-core technology, most widely used simulators for distributed algorithms and protocols do not make use of this technology. In this paper, we present a meta-simulator developed with an objective of producing highly parallel simulations for the analysis and evaluation of distributed algorithms and protocols. We call it a meta-simulator instead of simulator because it generates editable simulation frame works virtually for any concurrent execution platform. Simulation programs are produced as network of processes executing DAs following the required organization. The meta-simulator supports a time-driven execution model and an asynchronous model based respectively on direct Communicating Sequential Processes (CSP) channel communications and an intermediate programmable channel process. Both the models are important in their own context. CSP channel communication is based on strict formalism of Hoare's CSP which makes mathematical modeling easier[4]. Intermediate programmable channel process allows to introduce stochastic error process in the communication, which is an essential part of any wireless ad hoc network.

Meta simulator proposed in this work is designed using a model based approach in which network description can be achieved with high level tools independently from target programming syntax. It can support fairly complex networks consisting of hundreds of nodes and thousands of channels. The meta-simulator is evaluated using large number of simulations covering various scenarios. The simulator also carries significant pedagogical importance as it is being used successfully in the class room to facilitate the study of general distributed algorithms.

The rest of the paper is arranged such that the next section describes objectives behind this work and Section 3 elaborates composition of proposed meta-simulator. Section 4 outlines evaluation strategy and example simulations carried out. Section 5 briefly discusses related work and section 6 draws key conclusions of this research effort.

2. META-SIMULATOR DESIGN OBJECTIVES

Existing simulators fail to obtain results on a whole set of criteria: scalability, flexibility, concurrent execution and performance. Our goal is to alleviate problems hindering us to achieve these criteria. Therefore, we started development of this meta-simulator with following objectives.

Top-Down approach: It has been emphasized thoroughly that the system as a whole should be modeled to cater for required performance and development speed [10]. This can be achieved by using Top-Down approach. In such methodology, overview of the system is first formulated on an abstract level, specifying but not detailing any first-level subsystems. Each subsystem is then refined in later stages. This approach is also essential for the evaluation of distributed algorithms as it is very important to understand the system from top-level.

Separation of Definition and Behavior: The idea here is to divide the complete simulation procedure into two phases. At first, system definitions are achieved. For instance, in case of WSN protocol simulation, network topology shall be defined with physical layout, range and connectivity. This phase shall not involve description of system behavior. The behavior is defined later. This objective leads us to modularity and re-usability of simulation code. In most existing simulators, network definition and behavior are intermingled during simulation which renders it impossible to use the same code for different simulations. We achieve this objective using previously mentioned Top-Down approach.

Concurrency: One of the major design objectives of this meta-simulator is to make use of increasingly available concurrent computing capabilities. By exploiting this feature, we

can achieve better performance and higher scalability. Therefore, we have incorporated the generation of Occam simulation code. Occam is a well known programming language based on CSP. We are using the compiler KRoC (Kent Retargetable occam-pi Compiler) producing native code for current i386 processors[16].

Flexibility: Flexibility is an important aspect of any simulation tool. However, it is very broad term and must be defined in some detail clearly. We define flexibility in terms of multiple objective. Our prime goal is to be flexible in terms of generating simulations for a variety of distributed systems. Although we have evaluated our meta-simulator using large number of WSN based simulations, it is perfectly possible to use this meta-simulator for any other distributed system such as transport network. Another flexibility objective is to be able to target multiple architectures and programming languages. At the moment, we are producing simulations only in Occam code. However, due to the Top-Down approach and separation of definition and behavior, it is not very difficult to generate simulations in other programming languages.

Scalability: Simulation scalability is duly emphasized in existing literature. Most existing simulators do not scale beyond few hundred nodes. This is due to the fact that achieving high scalability is a challenging phenomenon. On the other hands, distributed systems with large number of nodes are emerging rapidly, therefore, scalability objective can not be ignored any further. We have developed this meta-simulator with this clear objective. This simulator can scale up to hundreds of nodes and communication channels.

Reality abstraction: Simulation results are more reliable if simulations do not assume unrealistic assumptions. One of the most common simulation methodology is to assume that global knowledge of the system is possessed by a central entity. This global knowledge is different from physically applying distributed algorithms, especially in the case of sensors in contact with local reality. Global model may increase performance and scalability, however it can fail to produce results similar to actual environment. By doing so, very essence of distributed is extracted out of simulation. In this meta-simulator and in any of the evaluation carried out, we have not assumed the presence of any global knowledge possessor. Every simulation is carried out in a pure distributed manner inferring local decision observed by the designers.

3. ARCHITECTURE OF SIMULATION PROGRAMS

As it was explained, the meta-simulator environment is a higher level modeling allowing to produce and manage networks instances. Practically, models are described and produced from Smalltalk-80, and the simulations described in this article are all produced in Occam.

The core model is an object data structure with links and nodes that represent communication channels and sensors. Each node has a name (such as P1 or P2), and an associated program known as a symbol (Node, NodeZero). This symbol is used to bind to a coded behaviour (Occam procedure). The list within braces is the connectivity of the named process.

```
P1 { P2 , P3, P4 } Node
P2 { P1 , P3 } Node
P4 { P1 } Node
P3 { P1, P2 } NodeZero
P5 { P6 } Node
P6 { P5 } Node
```

The structural description shown above is graphically represented in Figure 2.

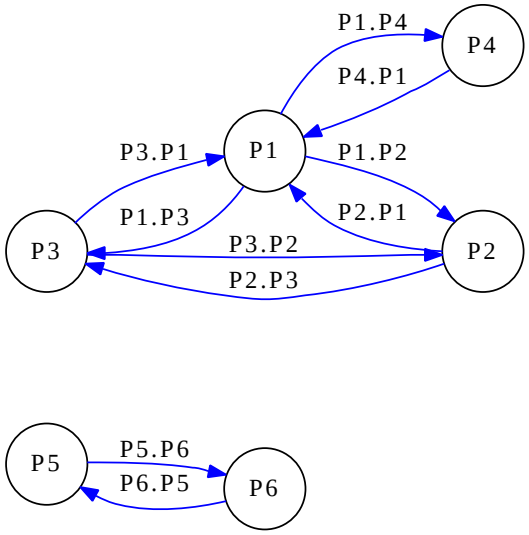

Figure 2. P1-P6 system is divided in 2 sub-networks

Occam is oriented toward concurrency and effectively support parallel execution on multiprocessors, and even on grid of processors. The level of concurrency varies from processes organized hierarchically to instruction level parallelism (PAR/barrier constructs). A single communication/synchronization mechanism is proposed conforming to CSP concepts : point to point channels with blocking operations from two corresponding channels. Occam also supports non-determinist primitives such as the ALT construct that selects a channel ready for communication from a proposed array of channels.

3.1. Execution models

Concurrent executions conform to two patterns:

synchronous model : participating processes represent sensor nodes. the working wireless point to point communications are represented by one Occam channel. We suppose that the sensor networks proceed by phases bounded by time considerations : emitters and receivers have time scheduled rendez-vous in which communication can happen.

We model this behaviour following Nancy Lynch proposal of distributed automata [12] that loops on exchanges between connected nodes, then state evolution, and then production of output messages for next phases.

Communications are represented by a parallel execution of input and output messages operated on channels, and we guarantee non-blocking behavior by sending and receiving null messages to denote absence of communication. This is a time driven simulation which behavior appears in regular pattern executed by all processes. The synchronous communication method assumes that sporadic TDMA exchanges take place at fixed period allowing to represent physical wireless broadcast by point to point operation on channels.

Erratic transmissions can be simulated on the message buffering mechanism by applying specific procedures before sending and after receiving.

asynchronous model : we again are following a well known model for distributed processing where *non empty channels* are used instead of blocking channels. This time each channel is represented by a process implementing channel characteristics. This can be useful to model actual computer networks where communication links have a storage capability, packet loss, disorder delivery, and in the case of wireless networks complex transmission situations.

The Occam program simply starts processes in parallel ensuring their connection through array of channels that are automatically synthesized.

3.2. Program organization

We have separated *process organization* from *process behavior*. Process organization is written by a program generator according to the higher level model: grid, random, or architecture specific.

- Process organization is an Occam master program file defining global constants for number of nodes, maximum fan-out to provide message buffer dimension, set of variables in which we can provide specific initialization data for each process.

  We also provide an *observation mechanism* by connecting each process to a multiplexer that output process message to a terminal. This is used to trace process state and results such as diameter computation, leader election, route calculation and packet transmission.

- Process behaviour relies on distributed operations and is hand-coded in Occam following the synchronous execution model. Behaviours are defined in *include* files for the master program that launches processes (P1, P2) on programmed behaviours (Node, NodeZero). Careful management of procedures will allow their reuse.

A strong point of this Occam compiler is its ability to manage variant array of channels, allowing processes to use whatever channel grouping is necessary for the network implementation.

3.3. Managing and running simulations

The top level has a Smalltalk GUI with a variety of functionality to control generation of different kind of networks, range specification, statistics, selection of outputs (Occam, graphics), compiling and executing, exploring network families, and choosing behaviours.

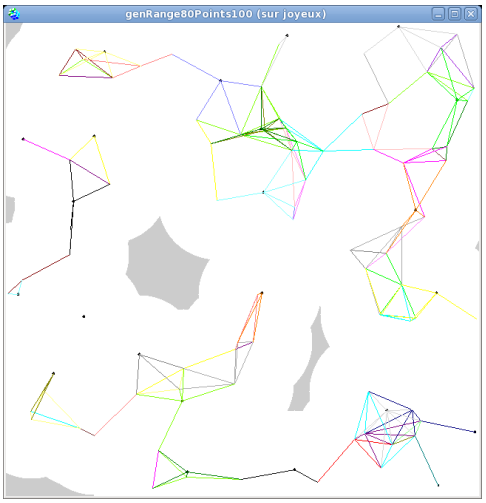

Figure 3. Random distribution of 100 sensors with range 80

4. EVALUATION

The meta-simulator proposed here is evaluated using a variety of scenarios, metrics and algorithms. We have performed evaluations using grid topology as well as random networks topology. In our simulations, number of nodes (distributed processes) varies from hundred to thousand nodes. Similarly, these simulations contain several thousands of communication channels. To keep evaluation discussion brief and to the point, we have divided this section into two subsections based on type of network topology (Process Distribution) used in the evaluation.

Described evaluations are thus achieved by varying algorithms operating on a single network, or by varying networks while executing a fixed set of operations. This section will describe shortly algorithms implemented in Occam, and discuss execution on random and regular networks.

4.1. Algorithms

Starting a network of distributed components, we are normally facing a first problem, which is to acquire a minimum knowledge about the network in which they participate.

To make progresses, two hypotheses can be taken which are an upper boundary of the number of nodes (*MaxNodes*), and the presence of different *Identity* number for each node.

Under these hypotheses, it is possible to bring in each node a number called the *diameter*, which represents the maximum distance between two nodes in the network:

distance computation : each node initializes a table of MaxNodes rows with its identity and a distance of zero in the first row. MaxNodes loops are executed where local tables are exchanged between nodes. At each exchange received tables are explored to find new identities, and each new identity is recorded in the local table with its distance incremented by one.

At the end of the loop, each node i in each network knows each other node j in *its network*, and the distance from this node $d(j,i)$.

local maximum distance : Max_i is computed on the local table that represents the local opinion on the maximum distance in the network.

diameter computation : in a second MaxNodes loop, each node sends its opinion of the global maximum, updating this opinion according to the maximum transmitted by other nodes. After these loops, $Max(Max_i)$ is known everywhere, representing the network *diameter*.

This operation is fundamental because it allows to find the exact number of loops necessary to bring any knowledge to each other node in the network. Lots of further algorithms can then be programmed, controlled by the diameter boundary: spanning tree, routing table or global computations. Diameter is usually far lower then MaxNodes.

In a second stage, these algorithms are explored that can be the core of a final application. We have taken measures on basic components that will be commented next section:

leader election : the image of a node critical for a group is the one of a leader holding the maximum *Identity* in a network. Leader election is a global maximum on the network, which can now be obtained in *diameter* steps.

routing table computation : *diameter* loops are executed where tables are exchanged, this time recording the channel index by which the first apparition of an identity has appeared. After these loops, every node i knows a path to join each other node j.

routing : Each node address a packet to the leader, then contributes to routing by forwarding received packet to the leader. In this formulation we accepted loss of packets, and finally each node print the trace of packets received and forwarded.

This code represents approximately 10 pages of Occam. It is traced progressively by printing through the multiplexer mechanism, and then serves on different network topologies. An interesting remark is that small networks *finish sooner* then largest ones, since the Occam processes of networks are logically independent.

4.2. Random networks

They are produced as shown in Figure 3, by spreading a N number of random points on a given surface then establishing a connectivity following the *range* specified for sensor emitters. Technically, we compute geometrical distances from node i to node j, and accumulate j in the neighbour set of i if this distance is less than *range*.

Once the connectivity is obtained, a network graph is developed according to the simulator model. Then this graph can be translated into whatever other representation is needed, as example graphical representations shown Figure 2, and more interesting into Occam programs that can run algorithms described in section 4.1.

We have collected a set of measures on small and large networks, showing topology characteristics, and execution times. This was achieved on an Intel I7 computer running Linux Ubuntu 9.10 and producing 8 processor on the basis of 4 core CPU. While topology discovery algorithms are bound to the whole system size, the second group of algorithms is bound to network diameters and execute at very high speed. To ease data collection, the meta-simulator GUI provides this functionality:

- selecting the algorithm Occam file from a repository
- defining number of nodes
- defining how the network space is swept (start,end,step)
- running several times the same network characteristics to smooth irregularities.

Figures 4 and 5 show computed topologies for two families of random networks. Figure 6 shows time values collected during problem generation, compilation and execution

of topology discovery, then routing. Notice the difference between the number of nodes and the diameter, that controls distributed algorithms performance.

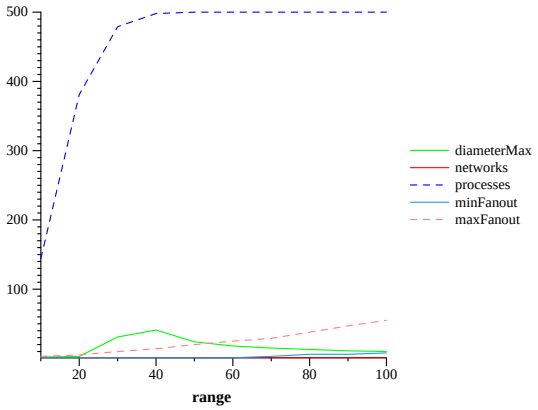

Figure 4. Topologies in a 500 node networks. For low ranges, curves show that several processes were isolated.

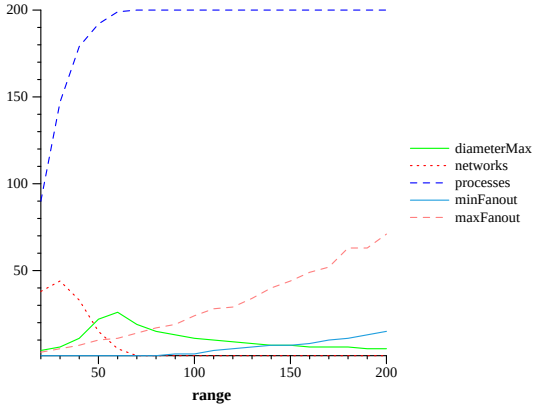

Figure 5. Topologies in a 200 node networks. The number of subsystems first increased, then decreased toward a single network when enough connectivity was obtained. The diameter follows the same rule.

4.3. Regular Distribution

Many WSN deployments follow a regular network structure. Therefore, support for a variety of grid based distribution of nodes is provided in our simulator. In Figure 7 and 8, some possible topologies are shown. We have run all our simulations with a topology similar to that of Figure 7, as it is the most commonly used in practical scenarios.

In these simulations, we have used varying number of nodes with a minimum of 400 nodes to more than thousand nodes. The complexity of such networks can be imagined using the diameter: the minimum diameter for simulated networks is 40 and in extreme cases it is more than 60. For each

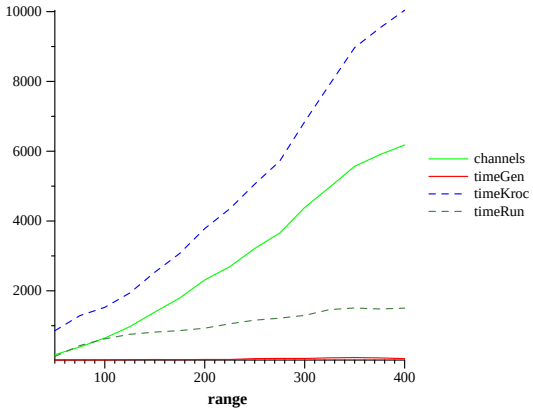

Figure 6. Generation, compilation, and execution times for a 100 node network. Diameter, Leader, and routing computations.

topology we have carried out two types of simulations. In first set of simulation, we have used standard CSP channel communication supported by Occam. So none of the message transmitted by a node is dropped due to channel error. In this phase, all the algorithms mentioned in the previous subsection have been simulated over Occam's default CSP channels. We have found that to simulate 1000 nodes simultaneously, only 755 seconds are required, which is roughly 12 minutes. We also find that an increase in simulation time against number of nodes is pretty much linear. These results are shown in Figure 9.

In a real ad hoc wireless scenario, data loss can occur due to bit-errors induced randomly by a physical channel. Therefore, in the second set of simulations, we have introduced a packet-error based stochastic channel model between two

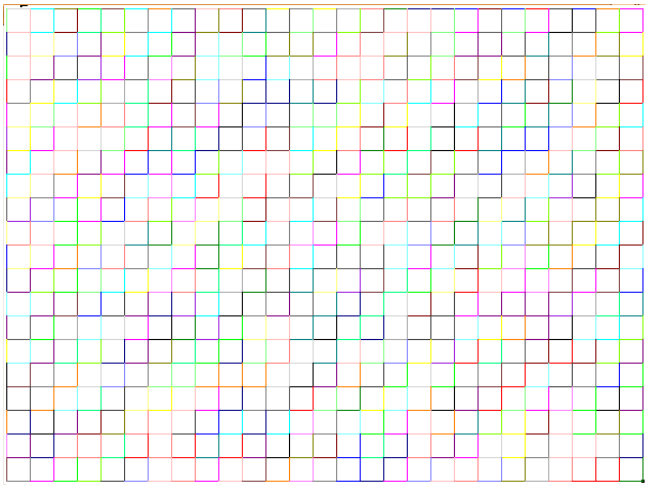

Figure 7. Grid based topology with maximum fanout = 4

communicating processes. This is to clarify that the model incorporated at the moment is not very sophisticated. However, it is not difficult to incorporate any complex stochastic channel model into the simulation. The in-depth discussion of wireless channel models is beyond the scope of this paper. In this set of simulations, we have simulated earlier algorithms over the stochastic channel model. The measured execution time was quite similar to that of non-faulty CSP channels, with only few more seconds consumed. We expect that the simulation time may slightly increase if more complex channel models are incorporated.

Simulating a sensor network of around thousand nodes, under 12 minutes, in a completely distributed manner is a very encouraging result. Even more important is the linearity of increase in time with the complexity of network. We were not able to compile larger simulations due to a 32 bit limitation from KRoC compiler, however, practical large problems mostly have low connectivity and produce more scalable simulations with smaller execution time.

5. RELATED WORK

Large number of simulators have been proposed in the past for the evaluation of distributed protocols and algorithms. Many of these simulators are particularly designed for Wireless Sensor Networks. There are few widely adopted simulators such as NS-2, Opnet and Tossim [1, 5, 9–11, 15, 17] .Although, designed for similar purposes, these simulators differ from each other greatly.

NS-2 is probably the most widely tool for simulating Ad Hoc networks. With the addition of SensorSim, now it supports simulation of WSN protocols. In NS-2, system definition and behavior are intermingled within simulation and it is fairly difficult to reuse simulations for different behaviors.

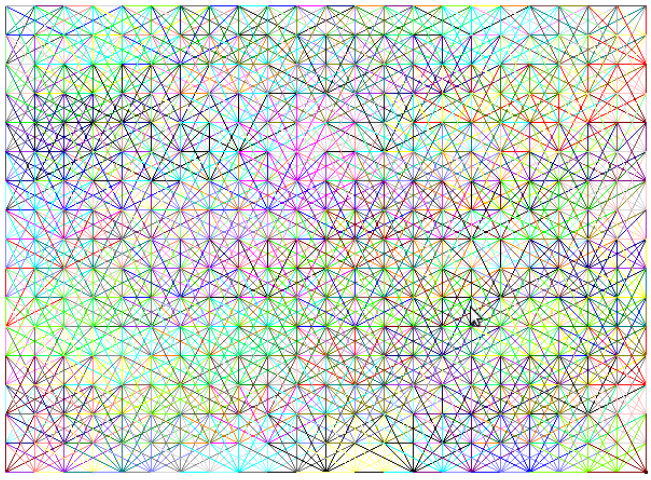

Figure 8. Grid based topology with maximum fanout = 16

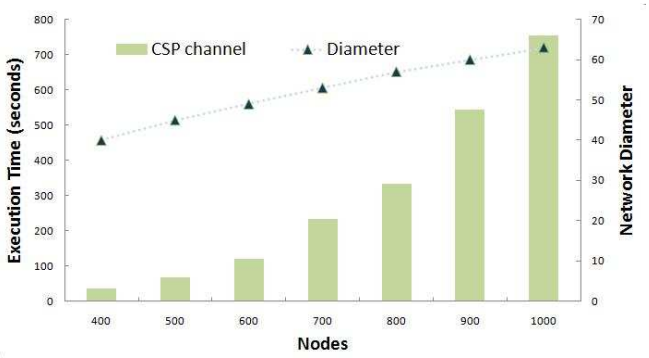

Figure 9. Simulation execution time for Grid based topology with maximum fan-out = 4

Similarly, the simulation development is only for simulation purpose. It is not possible to reuse simulation code to target any particular device or architecture.

TOSSIM [11] removes some of the problems of NS-2. It is a discrete-event simulator for applications on MICA and MICAZ Motes. Simulations must be coded as if being deployed on real devices. Therefore, code written for simulation purposes, can be directly ported to a hardware device. However, it is not possible for different nodes to behave differently during simulation.

GlomoSim [17] is a set of libraries basically developed to simulate wired networks. However, it can be used for the simulation of wireless networks as well. The simulator supports parallel execution of simulation components to some extent.

If we analyze carefully, we find that all these simulators sacrifice distributed nature of WSN to achieve performance goals. Global knowledge of all entities is generally assumed to speedup simulation process. In our proposed meta simulator, we have given due importance to the acrual distributed nature of wireless sensor networks.

Although it has been identified by reasearchers that support of concurrent execution should be incorporated in ad hoc networks simulation, however, full scale concurrency in WSN simulation is largely remained restricted to theoretical discussions and it has not been realized practically upto a significant level.

6. CONCLUSION AND FUTURE WORK

We have described a new way to simulate sensor networks on top of a concurrent language inheriting from CSP. The meta-simulator operates at a level above CSP on fairly complex problems allowing to generate networks with hundreds of nodes and thousands of channels.

Despite this complexity, programming remains simple since the designer is only concerned with writing of procedures representing local behaviours. Observation of the col-

lective operations is achieved by printing information to a multiplexer process. The meta-simulator is able to loop over classes of networks with pre-defined characteristics.

The meta-simulator is used in computer science master courses to produce complex network situations in which networking activities are implemented. The course covers local system topics explained on the basis of CSP, distributed algorithm design principles, and their applications in networking.

Part of the motivation to develop this framework was to obtain a clean separation between *networks* and *behaviours*. Networks are known to be highly variable and the need is to develop robust algorithms that would work whatever is the organization.

The work on this simulator is oriented to support cooperative projects. These projects are expected to cover a broader range, including communication channel modelling and mobility.

Another investigated direction is automatic synthesis from behaviour specifications to actual sensor code, or the direct execution of Occam binary from a virtual machine implemented on sensors.

REFERENCES

[1] Homepage of ns-2. `http://www.isi.edu/`.

[2] Luca Schenato home page. `http://www.dei.unipd.it/~schenato/`.

[3] A. Boucher, R. Canal, T.-Q. Chu, A. Drogoul, B. Gaudou, V. T. Le, V. Moraru, N. V. Nguyen, Q. A. N. Vu, P. Taillandier, F. Sempé, and S. Stinckwich. The around project: Adapting robotic disaster response to developing countries. In *IEEE International Workshop on Safety, Security, and Rescue Robotics*, 2009.

[4] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[5] X. Chang. Network simulations with opnet. In *WSC '99*, pages 307–314, New York, NY, USA, 1999. ACM.

[6] V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pásztor, N. Trigoni, and R. Wohlers. Wildlife and environmental monitoring using rfid and wsn technology. In *SenSys '09*, pages 371–372, New York, NY, USA, 2009. ACM.

[7] S. eun Yoo, P. K. Chong, T. Kim, J. Kang, D. Kim, C. Shin, K. Sung, and B. Jang. Pgs: Parking guidance system based on wireless sensor network. In *ISWPC '08*, pages 218–222, May 2008.

[8] L. Evers, P. Havinga, J. Kuper, M. Lijding, and N. Meratnia. Sensorscheme: Supply chain management automation using wireless sensor networks. In *IEEE ETFA*, 2007.

[9] J. C. H. H. Tyan, A. Sobeih. Towards composable and extensible network simulation. In *IEEE IPDPS*, 2005.

[10] A. Kroller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer. Shawn: A new approach to simulating wireless sensor networks. In *INSS '07*, 2007.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03*, pages 126–137, New York, NY, USA, 2003. ACM.

[12] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1996.

[13] J. Markoff. Can't find a parking spot? check smartphone. The New York Times, July, 12th 2008.

[14] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[15] G. A. S. Sundresh, W. Kim. Sens: A sensor, environment and network simulator. In *ANSS*, 2004.

[16] P. H. Welch and F. Barnes. Communicating mobile processes: introducing occam-pi. In A. Abdallah, C. Jones, and J. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 175–210. Springer Verlag, April 2005.

[17] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. *SIGSIM Simul. Dig.*, 28(1):154–161, 1998.