Simulation of MANETs: A Qualitative Comparison between JiST/SWANS and ns-2

Frank Kargl, Elmar Schoch
Ulm University, Institute of Media Informatics
89069 Ulm, Germany
{frank.kargl | elmar.schoch}@uni-ulm.de

ABSTRACT

Simulations play a vital role in the development and testing of ad-hoc networking protocols. However, the simulation of large networks is still a tedious task that consumes a lot of computing power, memory, and time. JiST/SWANS is a relatively new, Java-based ad hoc network simulator that reaches very good runtime behavior. Though the creators of JiST/SWANS already proved the capability of the simulation engine, they did not provide results that show the validity of simulations using the SWANS ad hoc network stack.

In this paper, we compare protocols and models implemented in SWANS to the corresponding implementations in ns-2. Using identical input parameters, we show where results are comparable and analyze reasons for differences. By showing that results achieved with JiST/SWANS are equivalent to those of ns-2, we support the usage of JiST/SWANS. As ns-2 performance problems when simulating hundreds or thousands of nodes and the complex mixture of Tcl and C/C++ code in ns-2, JiST/SWANS could be an interesting alternative. For a further simplification when using JiST/SWANS, we shortly introduce our simulation execution framework, which allows for generation, execution and evaluation of complex simulation studies.

Categories and Subject Descriptors: I.6.7 [Simulation and Modeling]: Simulation Support Systems

General Terms: Performance, Measurement, Management

Keywords: Simulation, JiST/SWANS, ns-2, MANET

1. INTRODUCTION

The development and testing of mobile ad-hoc networking protocols heavily depends on the support of simulation. Simulations are used e.g. when scalability of MANETs is to be tested and cost constraints allow the usage of only a small number of real devices. Another important advantage of simulations is the fact that the environmental conditions can be repeated an infinite number of times. This allows simulations that compare the performance of different protocols in exactly the same scenario. In the real world tests, small changes in movement and environment influence the transmission of radio waves and change the outcome of experiments. Finally, simulations may be used to test new systems that have not been build in hardware yet.

For all these reasons simulations have been used extensively for modeling of ad hoc networks. The authors of [1] have reviewed all publications from the MobiHoc workshops between 2000 and 2005 and found that 114 out of 151 papers published there used simulations to verify their results. The same paper also discusses the fact that doing a solid simulation study is a demanding endeavor.

For simulating MANETs, one usually needs the following components:

- A simulation software. Whereas this can be a self written piece of software, most researchers base their studies on readily available network simulators like ns-2, OPNET Modeler, Omnet++ etc.
- A software emulation of the physical environment which encompasses radio propagation model, simulation area, node mobility model.
- A software implementation of all network elements below the network layer, like emulation of the radio, the layer-2 network (e.g. IEEE 802.11 WLAN), etc.
- An implementation of protocols and applications, like AODV or DSR for ad hoc routing, TCP and UDP and data traffic generation.
- A mechanism to configure parameters for all the above mentioned components of the simulation and to collect results, e.g. by writing statistics to a log-file or a database.

When verifying a MANET system, there are two important properties that must be taken into consideration. The most important is of course *accuracy*: real world conditions and results must be reproduced also by the simulation as closely as possible. The second important aspect is the *runtime behavior* of the simulation, namely required processing power and memory consumption. Since usually available computing power and memory are fixed, the setup of a simulation study needs to be adapted to be able to run it in a reasonable amount of time. In addition, resource consumption depends strongly on the implementations of the used simulation tool.

Next, we will describe some of the most commonly used simulators for MANETs. They all fall in the category of

MobiEval'07, June 11, 2007, San Juan, Puerto Rico, USA.

so called discrete-event simulators, which means that the simulation is run in discrete (time) steps and that events are used as mean of communication between simulation entities.

ns-2 [2] has its origin in wired network simulation, but has become one of the most prominent tools used for MANET simulation since CMU integrated their "wireless extensions" [3]. It uses a combination of C/C++ and OTcl as an object-oriented variant of the Tcl scripting language to model simulations. One big advantage of ns-2 is the availability of a large amount of modules ranging from radio propagation models over implementations of a lot of wireless networking technology to ad hoc routing protocols and application models. On the other hand, ns-2 simulations become relatively slow and consume a very large amount of memory, once the number of nodes exceeds several hundreds. We will give details on that later in this paper.

There is also a parallelized version called PDNS [4] that allows to distribute ns-2 simulations to a cluster of machines. In [5] the authors claim that they have successfully simulated a network with 600,000 nodes using using 136 processors and high-end hardware with a very fast interconnection network.

Another commonly used simulator is GloMoSim [6] which is based on the Parsec simulation language [7]. Parsec is a C-like programming language for discrete-event simulation which like GloMoSim has been developed by UCLA. GloMoSim allows the simulation of wired and wireless networks, it is configured through a configuration file and provides a reasonable number of readily available protocols, radio layers, etc. However, available modules by far do not match that of ns-2. Custom extensions are realized by implementing additional Parsec modules. Speed and memory of GloMoSim is slightly better than that of plain ns-2 and allows simulations of networks of roughly 10,000 nodes on standard hardware. There is also a commercially available parallel version called QualNet [8]. The manufacturer claims that QualNet scales up to "10s of thousands of nodes".

Another simulator is OPNET Modeler [9], which is a commercially available simulation environment. In contrast to the previous tools, OPNET simulations are usually created and configured using a mix of graphical editors, state-diagrams and C++. At the end, simulations are compiled into a runtime that is executed. OPNET provides a large collection of protocols, layers, and other modules. In [10], the authors show that OPNET Modeler also has a relatively poor scalability which is comparable to plain ns-2.

Thus, only few alternatives remain for large-scale MANET simulations. Therefore, most researchers keep on using ns-2 and restrict their evaluations to smaller networks. At Cornell University, Rimon Barr e.a. developed a very interesting alternative called JiST/SWANS that gained a lot of interest in the MANET community lately. Although the simulator is still in an early stage and support with readily available modules is quite poor by now, its surprising performance benefits and a number of other advantages attracts more and more attention.

Whereas the original authors did some performance comparison between ns-2 and JiST/SWANS, there has been no comparison yet that investigates the quality of the SWANS components, i.e. whether MANET simulations actually lead to results that can be reproduced by other simulators. But in order to compare simulations that have been done on different simulation environments, it is vital to show that the systems and the implemented protocols are actually comparable. It is our goal in this paper to compare AODV and DSR simulations done with both ns-2 and JiST/SWANS.

Listing 1: A simple JiST Entity

```
1 import jist.runtime.JistAPI;
2 class hello implements JistAPI.Entity {
3     public static void main(String[] args) {
4         System.out.println("simulation
              start");
5         hello h = new hello();
6         h.myEvent();
7     }
8     public void myEvent() {
9         JistAPI.sleep(1);
10        myEvent();
11        System.out.println("hello world,
              t="+JistAPI.getTime());
12    }
13 }
```

In the next section we will first give a short introduction to JiST, SWANS, and a custom extension called which simplifies the creation and analysis of simulations. After that, we present the setup and analyze the results of the simulations of AODV and DSR networks which we did both in JiST/SWANS and ns-2. We conclude the paper with remarks on the comparability of JiST/SWANS with ns-2 and where special care has to be taken to produce meaningful results in own simulations.

2. JIST/SWANS

JiST [11] is an abbreviation of "Java in Simulation Time" and describes a discrete-event simulator developed by Rimon Barr e.a. at Cornell University. The central idea is to transform the Java virtual machine into a scheduler for events by modifying the way how method calls between simulation entities are conducted. On top of this basis, the authors have created SWANS [12], the "Scalable Wireless Ad hoc Network Simulator", which provides all the mechanisms needed to simulate MANETs.

2.1 JiST

In JiST simulations, objects are implemented as ordinary Java objects which means that all the advantages[1] of Java can also be used in JiST. Simulation components are called Entities and are marked with the interface *JistAPI.Entity*. Each entity has its own entity simulation time t_e which determines, at what time calls to other entities are scheduled. An Entity progresses its t_e by calling the JistAPI.sleep() function. On the other hand, the system time t_s determines what scheduled calls are called in what time-step.

Before execution, the classes of a JiST simulation are transformed using a byte-code modifier. In this step, calls to public methods of entities are asynchronously decoupled. This means that a call will return immediately, the call is queued in an Event queue and the corresponding code in the invoked object is executed when the called entity reaches the same simulation time of the calling entity.

Listing 1 shows a basic example of a JiST entity. The entity `hello` is created in the main method of the class. At the beginning ($t_s = 0$), the simulation time for all entities is implicitly set to $t_e = 0$. Then, method `myEvent()` is called. This method first increases the local simulation time counter by one time unit ($t_e = 1$). After that, method `myEvent()` calls itself again. As the simulator engine decouples this call

[1] platform-independence, large library, familiarity with language, etc.

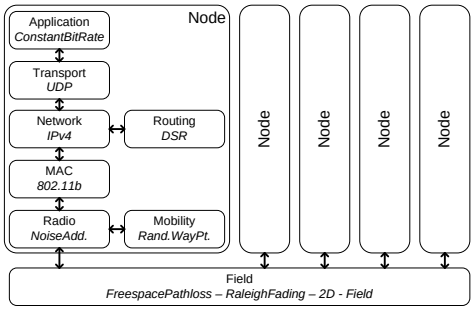

Figure 1: SWANS Architecture

to a public entity method, this is no recursion! Instead, the call returns immediately, the `println()` method is called and the `myEvent()` methods returns. As no other method calls are scheduled for $t_s = 0$, simulation time advances to $t_s = 1$ and the pending call to `myEvent()` is processed which starts over again. Scheduled calls for $t_s = 2$ are processed and the loop continues infinitely.

Note that non-public methods in entity objects are called directly. Internally, JiST replaces all entity references by proxy objects, so called separators, so that entity calls are never executed directly. Moreover, calls to public methods of entities are not allowed to return any values.

The JiST approach has a number of significant benefits:
Type-safety: As the "delivery of events" is simply a method call, the virtual machine always checks the compatibility of source and destination. So when receiving events, no additional type-checking needs to be done by the receiving entity.
No marshaling/demarshaling: As there are no explicit event data structures, there is no overhead for marshaling/demarshaling data. Instead, the entities just pass references within the VM. Using RMI, references can even be passed beyond VM boundaries.
Java: Using Java provides a large amount of benefits: robustness due to type safety, large library, ease of use, well-known to many developers, platform independence, garbage collection, etc.
Byte-code rewriting: Because class files are directly transformed, no source access is necessary. Existing libraries, protocol implementations, etc. can directly be used for simulations.
Parallel distribution: The JiST event scheduling can support parallel and optimistic execution, which crosses VM boundaries and is transparent for the application. However, in the current version, this is not implemented.

2.2 SWANS

SWANS is a collection of components to simulate ad hoc networks based on the JiST simulation engine. Figure 1 shows the architecture of typical SWANS simulations. Nodes consist of several entities that are linked together and represent the different stack elements of a network application. The lowest radio entity of each node is connected to a common field entity which is responsible for delivering packets passed down the stack to other nodes in wireless transmission range.

Packets are handed from stack to stack by simply passing references between entities, duplicating the packet via `clone()` where appropriate. As calling a method and passing a reference comes at virtually no cost, SWANS stays extremely fast, even with large node stacks and high data traffic load.

SWANS comes with a basic set of simulation entities covering basic scenario elements like different fading and pathloss models, several modules for positioning and mobility of nodes, radio noise models, an 802.11b MAC layer, an IPv4 layer, DSR, AODV, and ZRP MANET routing protocols, TCP and UDP transport layers. As an application, one can use e.g. a CBR traffic generator or an application proxy that allows any Java network application that uses the socket interface to be used atop of SWANS.

The field implementation of SWANS uses an optimization that uses hierarchical binning in order to further speed up simulations [11]. Whenever a node delivers a packet to the field, the field must deliver the packet to a set of other nodes that get affected by this packet after considering fading, gain and pathloss. A small subset of all nodes will be in reception range, whereas some other nodes get affected by interference above the sensitivity threshold. Depending on the field size and the transmission power, the majority of nodes will not be affected however, which is exploited by the hierarchical binning approach.

In contrast, ns-2 and GloMoSim implement a simple linear search through all nodes in order to determine affected ones. This clearly does not scale well with growing number of nodes and packets on large fields. ns-2 has recently implemented a better grid-based algorithm that enhances its scalability significantly [13]. The authors state that they have run simulations with up to 3000 nodes. SWANS' hierarchical binning performs even better and is one reason for its superior scalability. Next, we take a look at some performance evaluation results from the authors of JiST/SWANS at Cornell university.

2.3 Performance Evaluation

In [11], the authors evaluate both the micro-performance of JiST and the application performance of complete SWANS simulations.

In the micro-benchmarks they compare the raw event throughput in a simulation that is similar to the hello world program in listing 1. The program schedules an identical event for the next time step and then continues to the next step, which is repeated 5 million times. The authors show that JiST completes this task 1.97 times faster than the already highly optimized Parsec, 3.36 times faster than an ns-2 implementation in C, 9.84 times faster than GloMoSim and 78.97 times faster than an ns-2 event loop in Tcl. On the other hand, JiST is only 31% slower than a hand-crafted C program that inserts and removes elements from an efficient implementation of an array-based priority queue, which serves as a performance baseline.

At the same time, JiST uses only 36 Bytes per entity and event. As a comparison, ns-2 uses 544 Bytes per entity and 40 Bytes per event.

When looking at more complete simulations using SWANS, the numbers are similarly favorable for JiST/SWANS. The authors have implemented and compared a simple neighbor discovery protocol for JiST/SWANS, GloMoSim, and ns-2. With hierarchical binning enabled, JiST/SWANS outperforms all competitors. With 500 nodes, SWANS runs only 43 seconds and consumes 1,101 kB of memory. GloMoSim already takes 82 seconds and 5,759 kB and ns-2 uses 7,136 seconds and 58,761 kB. The memory and time requirements prevent running ns-2 with 5,000 nodes, but here GloMoSim takes 6,191 seconds and 27,570 kB compared to 430 seconds

and 5,284 kB for JiST/SWANS. SWANS even computes simulations with 50,000 nodes rather efficiently in 4,377 seconds using 49,262 kB of memory.

While previous work clearly showed the considerable performance advantages of JiST/SWANS, still the question remains open whether produced results are correct. This means that SWANS components have not been compared qualitatively with other ad hoc network simulators. As ns-2 simulations are widely accepted, an answer to this question may also raise the acceptance of JiST/SWANS. Thus, this work focuses on the problem of comparability of SWANS components corresponding to the results of a well-known simulator like ns-2.

2.4 DUCKS

Basically, doing simulations involves three steps: defining the simulation setup, executing the simulations, and finally evaluating gained results. In ns-2, this is usually the task of a mixture of various scripting tools. As an additional advantage of JiST/SWANS, setup and evaluation can also be done in plain Java. In the terminology of JiST/SWANS, a so called driver program is responsible to build up all required components and to start the execution. It is up to the developer to write own drivers for specific simulations.

As running complex simulation studies involves many single simulations with many parameters and large amount of output, a more sophisticated solution is desirable to handle the data. We have therefore developed an execution framework which we called DUCKS. DUCKS delivers solutions to every step in the typical simulation cycle. First of all, it implements a generic "driver" for SWANS simulations, which takes all necessary information from a configuration file and generates a set of corresponding simulations. For the execution, a central component is available which can distribute simulations to an arbitrary number of simulation servers. Finally, results from the simulation servers are collected and stored in a database. To simplify the evaluation of the results, there is also a graphical tool that allows visual generation of graphs from the data in the database.

3. SIMULATIONS

In order to evaluate qualitative properties of SWANS, we have developed simulations with identical parameter set both in ns-2 and SWANS. Focusing on application and routing level measurements, we are able to compare high-level key parameters of mobile ad hoc networks like successfully delivered packets or end-to-end packet delay.

3.1 Simulation Environment

The most important requirement for a valid comparison of the two simulators is a concise definition of the simulation setup. For instance, it is vital to assure that the packets to transport across the net are generated in the same way. Otherwise, different data traffic could influence underlying protocol layers and thus reduce comparability. This section gives a detailed description of all relevant simulation environment parameters and denotes differences in simulator basics where necessary.

The simulated scene comprises parameters like number of nodes, field dimension, node mobility model and simulated time. Table 1 briefly summarizes used values.

Field dimensions vary from 500m to 1500m, with a square shape. The network density, i.e. the average number of nodes in mutual vicinity, can be varied either by changing the field size while keeping the number of nodes fixed or by using different numbers of nodes for the same field size. In our simulations, we also use different numbers of nodes varying from 50 to 100 nodes, always keeping in mind that node density is the significant parameter.

Parameter	Value(s)
Number of nodes	50,100
Field	500–1500m square
Avg. node speed (m/s)	2.5, 5
Pause times (s)	0.0
Mobility model	Random Waypoint
Link-/MAC-Layer	IEEE 802.11 DCF
Transmission range (m)	250
Data traffic	bursts of 15 messages, 0.5/second
Simulation time (s)	120
Simulation runs	10

Table 1: Short overview on simulation parameters

Another important property of the scenario is node mobility. Both ns-2 and SWANS implement the widely used random waypoint mobility model which we also selected for our simulations, applying maximum node speeds of 5m/s and 10m/s and no pause times at waypoints. Setting node pause time when having reached a waypoint to 0.0 seconds means that nodes move around continuously. Following the JiST paradigm of scheduling, SWANS does not use pre-generated node movements like ns-2 does, but computes node movements in parallel to virtual time progress.

Finally, each simulation runs for 120s of virtual time, mantled with a start offset of 10 seconds and a settle period of 20 seconds at the end of the simulation.

Beyond the scene, main components of the simulation are network nodes. Each node consists of a stack of layer, including radio, MAC, IP, routing and application entities that are described in the following.

The radio interfaces' omnidirectional antenna of a node is supposed to be attached 1.5m above the ground, therefore wave reflection is modeled according to the two-ray-ground principle. Transmission power is set to a dedicated value in each of the simulators that results in a wireless transmission range of ∼250m. The MAC/Link layer uses the 802.11 implementations of the simulators, using a bandwidth of 1 Mbps. As routing protocol, we apply AODV, which is well-known and is implemented both in ns-2 and SWANS.

While we use the original code of each simulator distribution regarding all ISO/OSI layers including routing, we have developed a dedicated application to send packets. Thus we can guarantee that data packets are created with the same timing and size both in ns-2 and SWANS. This application generates data packets in bursts of 15 packets which are sent to the same destination. After a waiting time of 30s, the application selects another destination for the next burst. This rather lightweight data traffic scheme tries to avoid congestion effects while still keeping a considerable effort for routing.

In order to get representative and meaningful result values, each simulation configuration is executed ten times. For these ten simulations, all input parameters remain identical, whereas of course, all randomly chosen values are changed in each simulation. In our results, we show mean values and standard deviations of the 10 runs. Randomization particularly influences generation of data traffic and node movements. With each single simulation, source–destination pairs and send jitter as well as node waypoints and speeds differ.

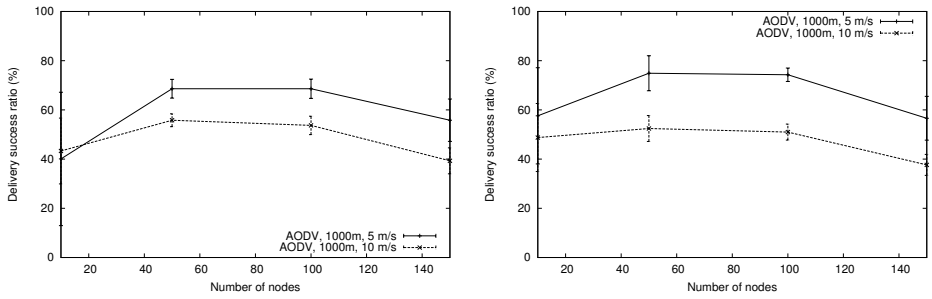

Figure 2: Comparison of delivery ratio with different node numbers (left ns-2, right JiST/SWANS)

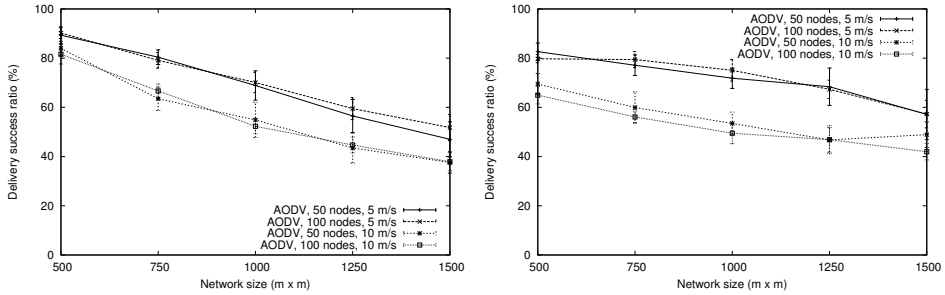

Figure 3: Comparison of delivery success ratio with different node density (left ns-2, right JiST/SWANS)

3.2 Simulation results

Four our analysis, we mainly focus on measurements on application level, namely successfully delivered packets and end-to-end delay. Because the communication stack includes the same components in SWANS and ns-2 and packets are handled by all of them, these measurements implicitly reflect their operation as well. A more fine-grained investigation of implemented components is subject to further work.

Fig. 2 shows simulations with different number of nodes and different mobility. These parameters are particularly interesting as they have a clear influence on the operation of MANETs. For each setup, we present a result set for ns-2 and JiST/SWANS separately. Essentially, the curves are very similar in both simulators. Though absolute values differ to some extent, both graphs show the increase in delivery ratio when nodes are less mobile and in both graphs, the highest delivery success is reached at the same node density with about 70 nodes. Therefore, this indicates a qualitative equivalence of ns-2 and SWANS regarding AODV and the lower layers.

Fig. 3 depicts delivery ratio results in dependence of node density and mobility. The graphs show decreasing delivery ratio of AODV when the network field size increases and thus node density decreases. Like before, although the absolute value decrease in ns-2 (left graph) is slightly larger than in SWANS (right graph), the main characteristics remain the same. For instance, node mobility clearly influences the delivery ratio, whereas different number of nodes has almost no impact. From this point of view, the AODV implementation of ns-2 and SWANS can again be regarded almost equivalent.

Besides delivery ratio, end-to-end delay is often an important factor for network applications. In Fig. 4, we compare average end-to-end delays of packets between sender and receiver. When using AODV in ns-2, packet delays stay at a low level throughout all field sizes, when 100 nodes reside on the field, whereas with 50 nodes, delay results rise abruptly starting with a field side length of $1000m$. We can see this difference between 50 and 100 nodes as well in the results of SWANS, yet all delay values raise slightly with increasing field size. Apart from that, the absolute delay values in SWANS are significantly higher than in ns-2 which is likely due to differences in default settings of the protocol implementation.

Finally, we present runtime performance measurements in Fig. 5 that we collected additionally during the simulations. The simulations for the data depicted in Fig. 5 were conducted on the same machine to rule out influences of hardware equipment. On the left, we see the time that was needed to process a single simulation. As one can expect, processing time increases when the network consists of more nodes. To compute the simulation, ns-2 and SWANS take nearly the same time. This indicates that not only the underlying simulation kernel is primarily responsible for the overall performance, but also the implementation of the simulated logic. Moreover, these results still include the asymmetry of mobility generation. Whereas in ns-2, movements are usually precomputed e.g. by the setdest tool, SWANS generates node mobility in parallel to the simulation. This shifts the small advantage of SWANS even a little further. On the right side, the graph depicts memory consumption of both simulators. While SWANS requires not much more than 10 MB, ns-2 consumes nearly 300 MB in the scenario with 150 nodes. Here, we clearly observe the major disadvantage of ns-2. It is very hard hard to use when scenarios get bigger and exceed several hundred nodes, which is not the case for JiST/SWANS.

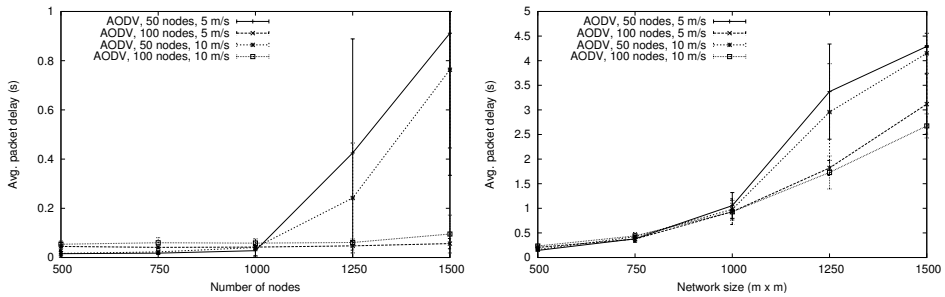

Figure 4: Delay comparison (left ns-2, right JiST/SWANS)

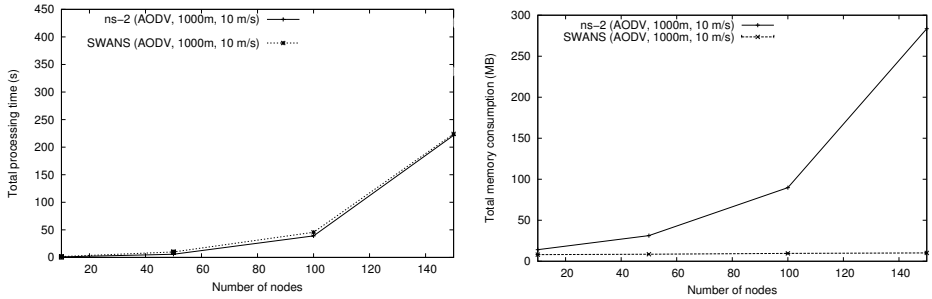

Figure 5: Processing time (left) and memory consumption (right) of ns-2 and SWANS

4. CONCLUSION

Our simulation experiments show that JiST/SWANS delivers results similar to that of ns-2, at least when including the components we decided to use, which actually reflect a typical ad hoc network scenario.

Our comparison also clearly confirms one core problem of every simulative analysis: deriving absolute values from a simulation and compare them with other simulations (on the same simulator or on different simulators) is almost impossible, even if *all* parameters and the implementation of all participating protocols are identical. The authors of [1] discuss this problem in more detail. Our study presented here has also some open questions. We used the default implementations of the protocols available in ns-2 and JiST/SWANS without further analyzing the comparability of the implementations. For instance, they may implement different optimizations, have other default settings of tunable parameters, etc.. In our future work, we plan a more detailed review of the component implementations. Additionally, we also plan to provide a more detailed analysis of the simulations, taking into account additional values like average hop count or protocol overhead in terms of packet generated. Moreover, it would be interesting to implement and include protocols into the evaluation, which are currently not available for JiST/SWANS, and test them against their ns-2 counterparts. Until then, we have provided some first evidence supporting the usage of JiST/SWANS for simulation of MANETs.

5. REFERENCES

[1] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso, "Manet simulation studies: the incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, October 2005.

[2] "Network Simulator ns-2," `http://www.isi.edu/nsnam/ns/`

[3] D.B. Johnson, "Validation of wireless and mobile network models and simulation," in *DARPA/NIST Workshop on Validation of Large-Scale Network Models and Simulation*, May 1999.

[4] G. Riley, R.M. Fujimoto, and M.A. Ammar, "A generic framework for parallelization of network simulations," in *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication*, March 1999.

[5] G. Riley, "PDNS website" `http://www-static.cc.gatech.edu/computing/compass/pdns/`.

[6] X. Zeng, R.L. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks.," in *Workshop on Parallel and Distributed Simulation*, May 1998.

[7] R. 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*, vol. 31, no. 10, pp. 77–85, October 1998.

[8] SNT, "Qualnet product homepage," `http://www.scalable-networks.com/products/qualnet.php`.

[9] OPNET, "Opnet modeller homepage," `http://www.opnet.com/products/modeler/home.html`.

[10] B.P. Zeigler and S. Mittal, "Modeling and simulation of ultra-large networks: Methodology responds to challenges," in *ULN Workshop*, November 2001.

[11] R. Barr, Z.J. Haas, and R. van Renesse, "Jist: An efficient approach to simulation using virtual machines," *Software Practice & Experience*, vol. 35, no. 6, pp. 539–576, 2005.

[12] R. Barr, Z.J. Haas, and R. van Renesse, *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks*, chapter 19 - Scalable Wireless Ad Hoc Network Simulation, Auerbach, 2005.

[13] Valeri Naoumov and Thomas Gross, "Simulation of large ad hoc networks," in *MSWIM '03: Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, New York, NY, USA, 2003, pp. 50–57, ACM Press.