Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. K. Rothermel
Betreuer:	Dipl. Inf. Daniel Minder Dipl. Inf. Jörg Hähner

Seminararbeit

A Comparison of the architecture of network simulators NS-2 and TOSSIM

im Rahmen des Seminars
“Performance Simulation of Algorithms and Protocols”

im Studienprojekt Cubus

Michael Karl

31. Januar 2005

Fakultät für Informatik, Elektrotechnik und Informationstechnik
Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universität Stuttgart
70569 Stuttgart

Contents

1. Introduction 3
1.1. Motivation 3

2. Architecture of Tossim 3
2.1. About Tossim 3
2.1. Architectural Overview 4
2.2. Component Graphs 5
2.3. Events (Execution Model) 6
2.4. Hardware Emulation 6
2.5. Models 6
2.5.1 Radio Models 7
2.5.2 ADC Models 8
2.6. Communication Services 8
2.7. Visualization Tool 8

3. Architecture of NS-2 9
3.1. About NS-2 9
3.1. Architectural Overview 9
3.2. Network Components 10
3.3. Event Scheduler 11
3.4. Hardware Emulation 12
3.5. Languages 12
2.5.1 OTcl: The user language 13
2.5.2 Tclcl 13
2.5.3 Tcl 8.0 13
3.6. Visualization tool 13
4. Conclusion 14

1. Introduction

1.1. Motivation

We live in a mobile world, where accessing information anytime and anywhere becomes more and more important. Recent changes in technology, such as the growth of the internet or the boom in the field of mobile phones have required robust, reliable and evaluated network protocols.

There are several methods for evaluating the behavior of protocols. Besides performance analysis methods like wide-area testbeds and small-scale lab evaluations the common performance analysis method is network simulations.

To carry out network simulations, network simulators are used. The development and employment of network simulators has a long history. For example the network simulator NS derives from REAL (Realistic and Large) which derives again from NEST (Network Simulation Testbed). [8]

But most network simulators have some drawbacks. Many have a high resource consumption and lack certain features, like modularity, adaptability and interactivity. Therefore, a student project called CUBUS has been initiated that is aimed at developing a new network simulation tool which supports these features.

Today's network simulators have widely varying focuses. VINTs network simulator NS focuses on the simulation of networks at the packet level, while TOSSIM was developed to aid in the simulation of TinyOS sensor networks.

Just as different network simulators have different focuses and are employed in different fields of research, they vary in their architecture.

To develop an understanding of how to design the architecture of the simulator CUBUS, this paper investigates the architecture of two prominent network simulators – that of Tossim and NS-2 – and compares them.

2. Architecture of Tossim

2.1. About Tossim

TOSSIM is a discrete event simulator for TinyOS wireless sensor networks. TOSSIM and TinyOS were developed at UC Berkeley. In contrast to the network simulator NS, TOSSIM captures the behavior and interactions of networks not on the packet level but at network bit granularity.

TinyOS is a sensor network operating system that runs on so-called motes. Motes are tiny sensing and computational devices that have very limitted communication, computational and energy resources. TinyOS sensor networks are often composed of large numbers of these little hardware devices [7].

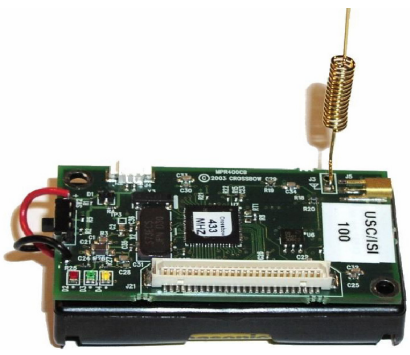

Figure 1. A hardware mote

TOSSIM simulates these motes by abstracting all of their hardware functionality and modeling in software.

2.1. Architectural Overview

The TOSSIM architecture is composed of five parts [4]:

- TinyOS Component Graphs (Frames)
- Execution Model (Events)
- Models (Radio and ADC Models)
- Hardware abstraction components
- Communication Services

Figure 2 shows these five parts that make up TOSSIM.

Every TinyOS Program is a graph of components. Each of these TinyOS components represents an independent computational entity.
The first part of the TOSSIM architecture, the “Component Graphs” or also designated as “Frames” support the user in compiling TinyOS component graphs into the simulation infrastructure. Here, TOSSIM takes advantage of TinyOS’s structure.

The second part of the architecture, the Execution Model is essentially a discrete event queue that enqueues and dequeues events and is responsible for the actuation of the simulation.
To model the characteristics of TinyOS network motes, TOSSIM provides mechanisms for extensible radio and ADC models. For a detailed description of these models see section 2.5.

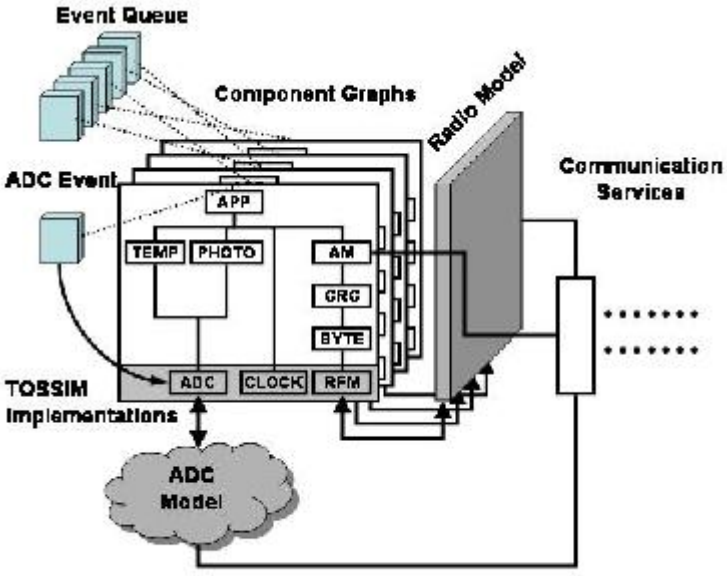

Figure 2. TOSSIM architecture: Frames, Events, Models, Components and Services (from[4]).

TinyOS abstracts all hardware resources as components. Another part of the TOSSIM architecture contains TOSSIM implementations and a small number of re-implemented TinyOS hardware abstraction components.

Last but not least the simulator engine provides a set of communication services for interacting with external applications which is shown in Figure 2.These services allow programs to connect to TOSSIM over a TCP socket to monitor or actuate a running simulation.

The next section describes the five parts of the architecture in detail.

2.2. Component Graphs

Basis for understanding the “Component Graphs” part of the architecture is a fundamental understanding of TinyOS’s program structure.

A TinyOS program is a graph of components [4]. Figure 3 shows a simplified component graph of an TinyOS Application.

The TinyOS component has five interrelated parts: a set of command handlers, a set of event handlers, an encapsulated private data frame, a structure of private variables that can only be referenced by the component itself and a bundle of simple tasks.
Tasks, commands, and event handlers execute in the context of the frame and operate on its state. Commands and events are mechanisms concerning communication between components, while tasks are used internal. To facilitate modularity, each component also declares the commands it uses and the events it signals.

A command is used to direct a request to a component to perform some sort of service. An example for a command may be the request to initiate a sensor reading. An event signals the completion of a service. In general the command returns immediately and the event signals completion at a later time.

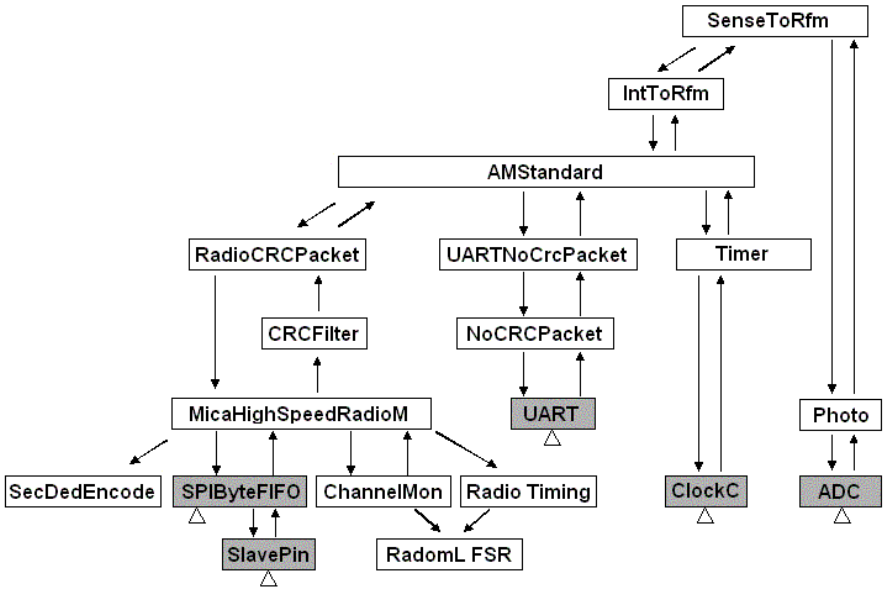

Figure 3. Component Graph of a TinyOS application (from [4]).

Tasks perform the primary work in a TinyOS application. Commands and events may post a task. A task is a function executed by the TinyOS scheduler at a later time. This means, that

a computation is not performed immediately but rather at a later time by deferring the computation to a task. Tasks are atomic with respect to other tasks and run to completion, though they can be preempted by events. Tasks can call lower level commands, signal higher level events, and schedule other tasks within a component. The run-to-completion semantics of tasks allows them to be much lighter-weight than threads and make it possible to allocate a single stack that is assigned to the currently executing task.This is essential in memory constrained systems.

TOSSIM takes advantage of this and generates discrete-event simulations directly from TinyOS component graphs. Essentially, the code that runs in TOSSIM is the same as on sensor network hardware except for a few low-level components that has to be replaced by TOSSIM allowing hardware interrupts to be translated into discrete simulator events.

2.3. Events (Execution Model)

Since TOSSIM simulates TinyOS mote hardware it has to abstract and model all of the characteristics of the underlying hardware in software. Hardware works with interrupts, so TOSSIM has to model this as well. For that purpose a simulator event queue sits at the core of TOSSIM that delivers the interrupts that drive the execution of a TinyOS application. This event queue may be considered as the heart of TOSSIM.

TOSSIM models each TinyOS interrupt as a simulation event [4]. But note that simulator events are distinct from TinyOS events. Each event is associated with a specific mote. Simulator events run atomically with respect to one another. Therefore, unlike on real hardware, interrupts cannot pre-empt one another. After each simulator event executes, TOSSIM checks the task queue for any pending tasks,and executes all of them in FIFO (First-In, First-Out) scheduling order. In TOSSIM, interrupts do not pre-empt tasks. The hardware abstraction components of the TOSSIM implementations contain a software implementation of an interrupt handler. When a simulator event calls an interrupt handler in the TOSSIM hardware abstraction components the interrupt handler signals TinyOS events and calls TinyOS commands. These TinyOS events and commands again post tasks and cause further simulator events to be enqueued, driving execution forward.

2.4. Hardware Emulation

The TinyOS operating system running on motes abstracts each hardware resource as a component. TOSSIM takes advantage of that by replacing only a small numer of these components (such as the ADC, the Clock, the EEPROM, etc.) it emulates the behavior of the underlying raw hardware. TOSSIM models these components in the hardware abstraction components part of the architecture. According to [4] the connection point for the simulated environment is provided by the low level components that abstract sensors or actuators.

2.5. Models

The TOSSIM architecture include two different models. Radio models for all kinds of transmission aspects and ADC Models for the Analog-Digital-Converter.

2.5.1 Radio Models

In general, a radio models defines all characteristics concerning the transmission from one node to another node, e.g. interference, collision, signal strength, etc. In TOSSIM, the models are external to the core simulator, which then can remain simple and efficient.

TOSSIM uses a very simple abstraction for a network signal, it is either a one or a zero.
All transmission signals have equal strength and collision is modeled as a logic OR.

A disadvantage in the TOSSIM radio models is that distance does not effect signal strength. Consider a situation where mote B is very close to mote A. It follows that B cannot cut through the signal from a far-away mote C. This makes interference in TOSSIM generally worse than expected real world behavior.

TOSSIM provides two built-in radio models [5], for what the developers of TOSSIM have observed to be common needs:

- the “simple” radio model
- the “lossy” radio model

The “simple” radio model places all of the nodes in a single radio cell, where every bit transmitted is received error-free. Although no bits are corrupted due to error, two motes transmitting at the same time can lead to problems (because every mote in the cell will hear the overlap of the signals). However, the probability of two motes transmitting at the same time is very low, due to the TinyOS CSMA protocol.

The “lossy” model on the other hand places the nodes in a directed graph with bit error probabilities. Each edge (u, v) in the graph means that the signal of node u can be heard by node v. Every edge has a weight that indicates the probability that a bit sent by node u will be corrupted (and therefore flipped) when node v hears it.

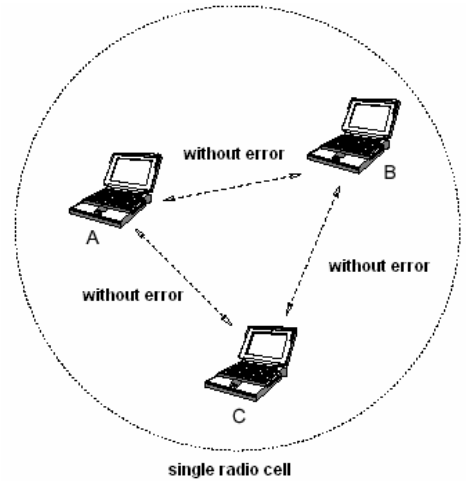

Figure 4. Simple radio model with error-free transmission

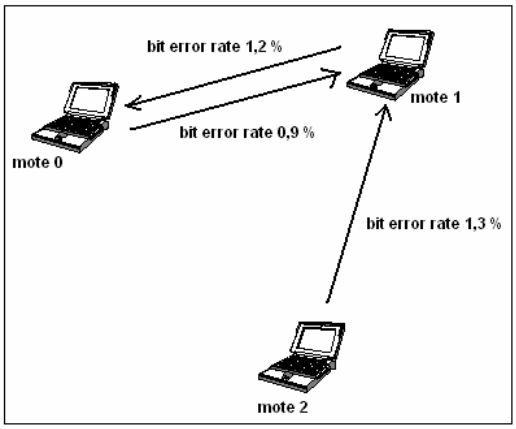

Figure 5. Lossy radio model with bit-error probabilities

2.5.2 ADC Models

ADC stands for Analog-Digital-Conversion. According to [3] an Analog-Digital-Conversion is "an electronic process in which a continuously variable (analog) signal is changed, without altering its essential content, into a multi-level (digital) signal."

TOSSIM provides two ADC models [5]:

- random and
- generic

The ADC has several channels that can be sampled. In the random model, if any of the channels are sampled, it returns a 10-bit random value. The generic model has the functionality of the random model but in addition to that provides also the possibility to be actuated by external applications. These external applications can actuate the generic ADC model by using the TOSSIM control channel, setting the value for any ADC port on any mote. In this way, TinyViz supports through the ADC plugin to set the 10-bit value read from any of the mote's ADC ports.

2.6. Communication Services

TOSSIM provides mechanisms, so-called communication services that allow applications running on a PC to communicate with TOSSIM over TCP/IP [4]. For that purpose, TOSSIM has a command/event interface that mediates between PC applications and the simulation. TOSSIM signals events to applications, providing data on a running simulation, such as debug messages or sensor readings and applications call commands on TOSSIM to actuate a simulation or modify its internal state.

2.7. Visualization Tool

TOSSIM comes with an Java visualization and actuation environment called TinyViz. [4][5] Figure 7 shows a screenshot of TinyViz.

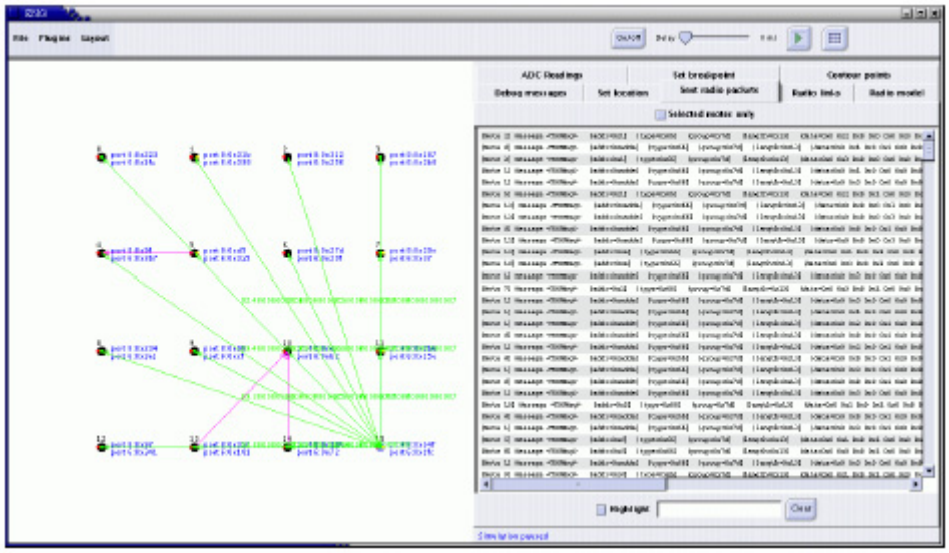

Figure 6. TinyViz connected to TOSSIM running an object tracking application (from [4]).

3. Architecture of NS-2

3.1. About NS-2

The network simulator NS is a discrete event network simulator developed at UC Berkeley that focuses on the simulation of IP networks on the packet level [3].

The NS project (the project that drives the development of NS) is now part of the Virtual InterNetwork Testbed (VINT) project, that develops tools for network simulation research [1].

Researchers have used NS to develop and investigate protocols such as TCP and UDP, router queuing policies (RED, ECN, CBQ), Multicast transport, Multimedia and more [8].

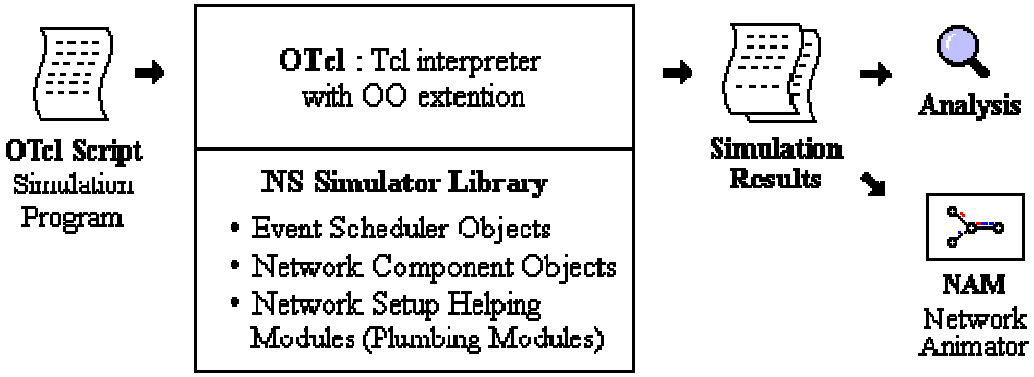

Figure 7. Simplified User's View of NS (from [3]).

NS is basically an Object-oriented Tcl (Otcl) script interpreter with network simulation object libraries. NS has a simulation event scheduler, network component object libraries and network setup (plumbing) modul libraries (see figure 7).

To use NS for setting up and running a network simulation, a user writes a simulation program in Otcl script language.

Such an OTcl script initiates an event scheduler, sets up the network topology and tells traffic sources when to start and stop transmitting packets through the event scheduler.

3.1. Architectural Overview

The NS-2 architecture is composed of five parts:

- Event scheduler
- Network components
- Tclcl
- OTcl library
- Tcl 8.0 scipt language

Figure 8 shows a graphical overview of the NS-2 architecture. According to [3] “a user can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library.” The event schedulers and most of the network components are implemented in C++ because of efficiency reasons. These are available to

OTcl through an OTcl linkage that is implemented using tclcl. These five components together make up NS, which is an object-oriented extended Tcl interpreter with network simulator libraries.

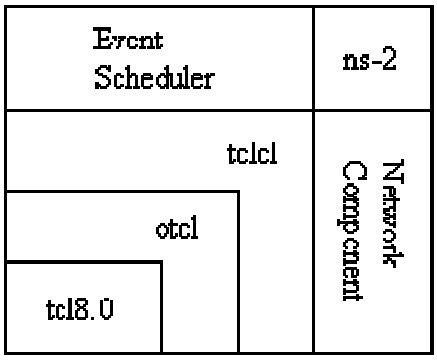

Figure 8. Architectural View of NS (from [3]).

3.2. Network Components

NS models all network elements through a class hierarchy. Figure 9 shows a partial OTcl class hierarchy of NS [3] because the complete NS class hierarchy would go beyond the scope of this paper.

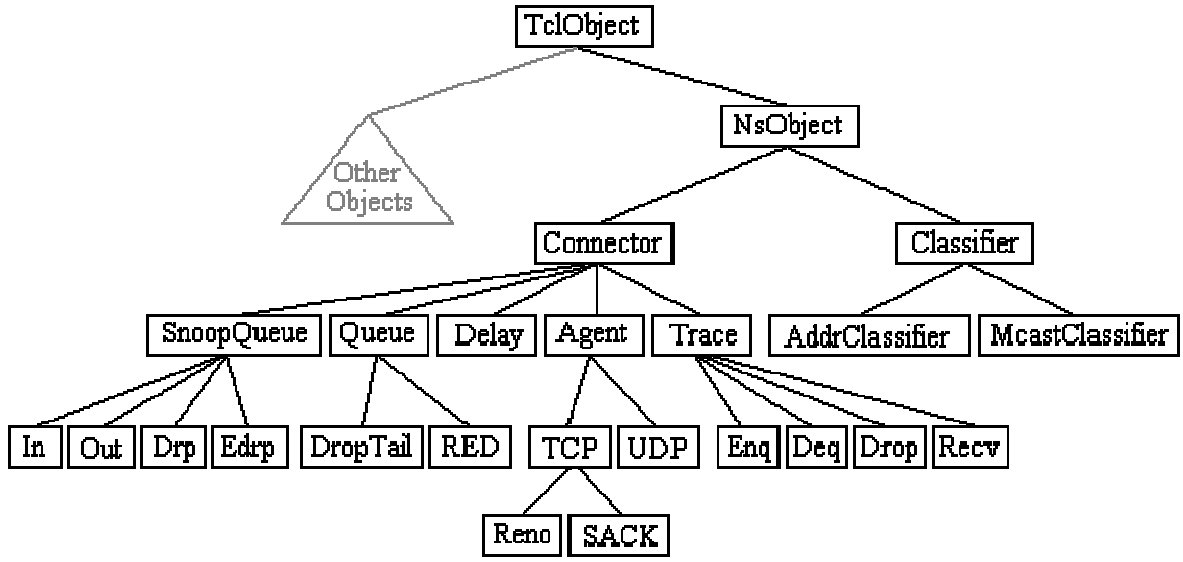

Figure 9. Architectural View of NS (from [3]).

In this class hierarchy, the TclObject class is the superclass of all OTcl library objects (network components, event scheduler, timers and others). A subclass of TclObject, NsObject again is the superclass of all basic network component objects that handle packets. Network objects, such as nodes and linkes can then be composed of this basic network components. Moreover, NsObject has two subclasses, Connector and Classifier. Connector is the superclass of all basic network objects that have only one output data path and Classifier is the superclass of all switching objects that have possible multiple output data paths.

Network objects can now be composed of all basic network component objects that are under the NsObject class.

For example a **node** is a compound object composed of a node entry object and of classifiers. NS has two two types of nodes, unicast nodes and multicast nodes. A unicast node consists of an address classifier and a port classifier while a multicast node consists of a classifier that classify multicast packets from unicast packets and a multicast classifier that performs multicast routing.

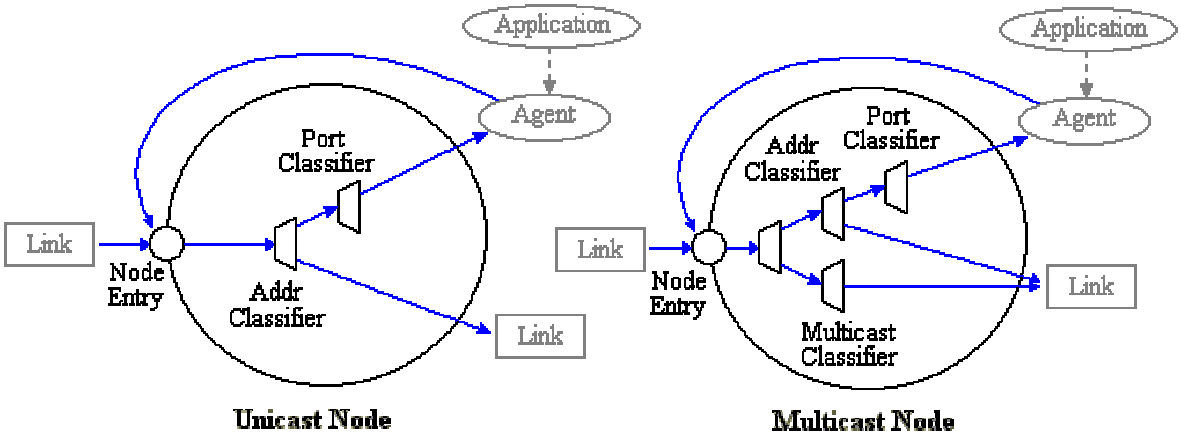

Figure 10. Unicast and multicast node as examples for compound objects (from [3]).

Another compound object in NS that is made up of subclasses of NsObject is a **Link**. When a user creates a duplex-link, two simplex links in both directions are created. Moreover, the links has a Queue object, Delay object, TTL object and a Null Agent as shown in Figure 11.

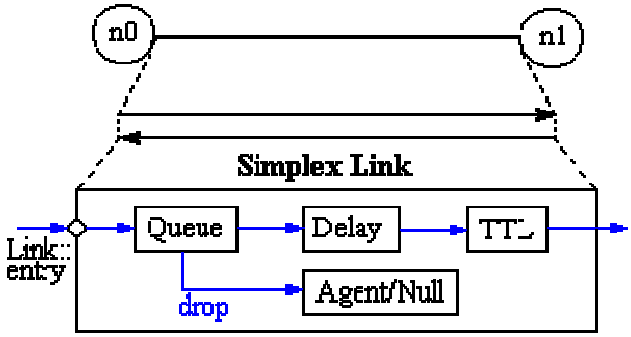

Figure 11. A Link as an example for a compound object (from [3]).

3.3. Event Scheduler

To drive the execution of the simulation, to process and schedule simulation events, NS makes use of the concept of discrete event schedulers [3]. In NS, network components that simulate packet-handling delay or that need timers use event schedulers.

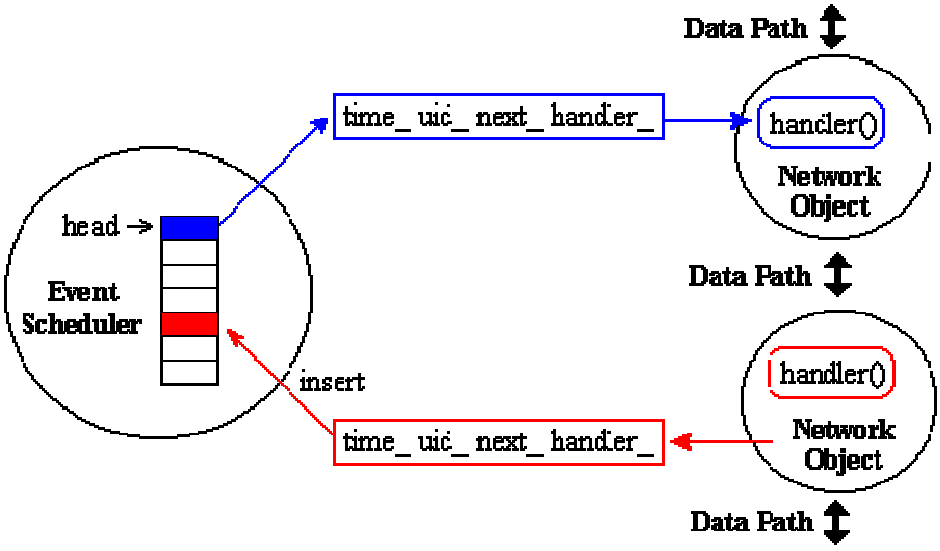

Figure 12. The discrete event scheduler (from [3]).

Figure 12 shows two network objects, each of it using an event scheduler. If an network object issues an event, it has also to handle the event later at scheduled time.
In NS, there are two different types of event schedulers – real-time and non-real-time schedulers. There are three implementations (List, Heap and Calendar) for non-real-time schedulers; the default is Calendar. For a description of real-time schedulers see section 3.4.

3.4. Hardware Emulation

The real time scheduler (one of the two types of NS event schedulers) is used for emulation. Emulation allow the simulator to interact with a real live network.

3.5. Languages

As depicted in section 3.1, NS is an OTcl script interpreter with network simulation object libraries. But NS in not only written in OTcl but also in C++. For efficiency reasons, NS exploits a split-programming model. This is because the developers of NS have found that separating the data path implementation from the control path implementation will reduce packet and event processing time.

Task such as low-level event processing and packet forwarding requires high performance and are modified infrequently, therefore the event scheduler and the basic network component objects in the data path are implemented in a compiled language that is C++.

On the other hand, task such as dynamic configuration of network objects and characteristics of traffic sources undergo frequent change, therefore the definition, configuaration and control of the simulation is expressed using a flexible and interactive scripting language that is Tcl.

The compiled C++ objects are made available also to the OTcl interpreter through an OTcl Linkage that creates a matching OTcl object for each of the C++ objects. In this way, the controls of the C++ objects are given to Otcl (see figure 13).

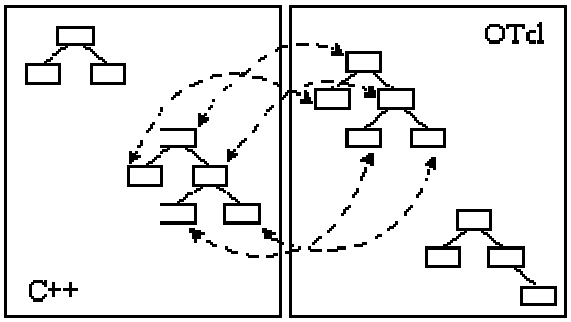

Figure 13. C++ and OTcl: The Duality (from [3]).

2.5.1 OTcl: The user language

OTcl, short for MIT Object Tcl, is an extension to Tcl/Tk scripting language for object-oriented programming developed at MIT.

OTcl is used to express the definition, configuration and control of the simulation in NS. In NS, an OTcl script creates an event scheduler, sets up the network topology with nodes and likes between them, creates traffic, inserts errors and sets tracing options.

2.5.2 Tclcl

The OTcl linkage is implemented using tclcl.

2.5.3 Tcl 8.0

Tcl is a scripting language.

3.6. Visualization tool

The VINT project provides a visualization tool called nam (network animator). Below you can see a screenshot of a nam window.

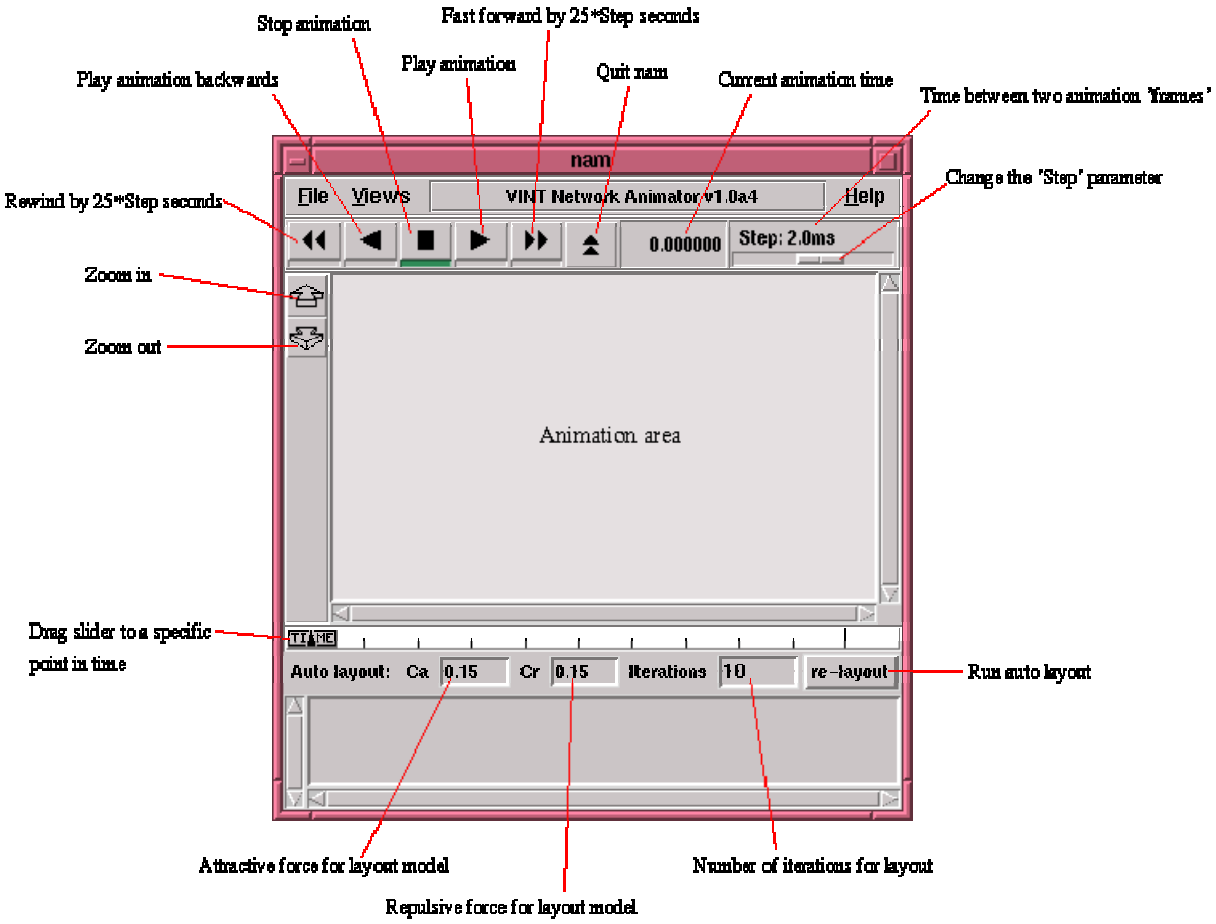

Figure 14. The network animator NAM (from [2]).

4. Conclusion

The network simulators NS-2 and TOSSIM focus on different aspects and have therfore totally different architectures. While NS-2 focuses on the simulation of network nodes at the packet level TOSSIM's approach is to simulate the TinyOS sensor networks at bit level granularity.

However, the architecture of NS-2 and TOSSIM have some aspects in common.
Both simulators have some kind of event handling mechanism to cope with the processing of events. TOSSIM calls it event queue, NS name it event scheduler.

Moreover, both simulators model the components of the simulated network in some way. TinyOS abstracts each hardware device as a component and a graph of these components forms a TinyOS program. And in NS, a network is composed of compound components which are moded through the NS class hierarchy.

Bibliography

[1] The network simulator
http://www.isi.edu/nsnam/ns/.

[2] Marc Greis's tutorial (now maintained by VINT group)
http://www.isi.edu/nsnam/ns/tutorial/index.html.

[3] J. Chung, M. Claypool. NS by Example. Worcester Polytechnic Institute (WPI)
http://nile.wpu.edu/NS/purpose.

[4] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).*

[5] P. Levis and N. Lee. TOSSIM: A Simulator for TinyOS Networks, 2003.

[6] Whatis.com / analog-to-digital-conversion
http://searchsmb.techtarget.com/sDefinition/0,290660,sid44_gci213760,00.html

[7] Simulating TinyOS Networks
http://www.cs.berkeley.edu/~pal/research/tossim.html

[8] L. Breslau, D. Estrin, K. Fall. Advances in Network Simulation. 2000.