A Mobility Framework for OMNeT++

Witold Drytkiewicz, Steffen Sroka, Vlado Handziski,
Andreas Köpke, Holger Karl
Telecommunication Networks Group
Technische Universität Berlin
drytkiew, sroka, handzisk, koepke, karl@ft.ee.tu-berlin.de

January 22, 2003

Abstract

Communication networks are an important application area for OMNeT++. While fixed topology networks are relatively easy to implement with the available means, an easy to use and effective extension for mobile networks is still missing.

We present a framework for OMNeT++ to support simulations of mobile networks. The framework is intended to be deployable 'as is', without the need to adapt existing modules, as well as to be easily extensible to fit the requirements of a particular problem. Especially, it should provide a unified structure, allowing switching between different levels of abstraction, without changing the communication logic on top of it.

The framework has three main components. First it provides a set of independent modules that implement node mobility, dynamic connection management and a wireless channel model. Second, it includes a library of standard modules for common protocols. And third, it suggests design guidelines for simulation models of wireless devices, e.g., a blackboard for automatically exchanging information between various communication protocols.

1 Introduction

Mobile networks are complex to simulate and require the collaboration of at least tens of different entities to realistically represent some high-level behavior. While other simulators like ns2 [1] have working extension packages to support mobility, available extensions to OMNeT++ lack usability and flexibility. Mobile network simulations built from scratch are error prone, bind a lot of resources and are usually not intended to be reused afterwards.

In this paper, we propose a simulation framework for mobile networks in OMNeT++. The framework consists of an architecture for mobility support and dynamic connection management, a model of a mobile node in OMNeT++ and a specification of the internal structure (see Fig. 4). We also provide a library of standard modules for the lower layers of the ISO/OSI protocol stack. Our framework is suited for ad hoc networks as well as for access point-based networks. A current version is available on the TU Berlin Dept. of Telecommunication Networks web site [3].

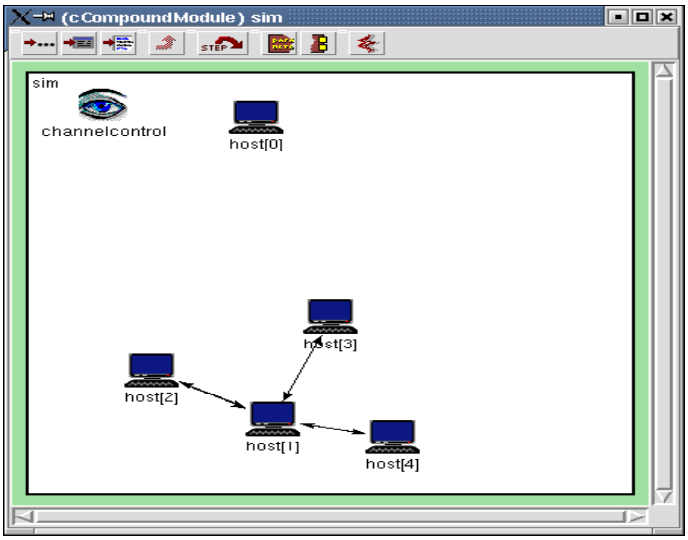

Figure 1: Snapshot of an ad hoc network simulation using the mobility framework

Figure 1 shows a snapshot of a running simulation of an ad hoc networks using our framework.

Learning from experiences with other simulation tools, we designed the framework with usability, extensibility, performance and scalability in mind. As a non-functional requirement we put special emphasis on maintaining a complete and sound documentation.

The design goals are achieved as follows:

Usability: Even the most highly performing and flexible software will not be deployed if the users struggle to put it to work. To keep the integration effort low, we created well defined, simple interfaces and provide a programming reference and design guide. To enable rapid prototyping of higher layer protocols, we provide standard modules for the lower layers of the protocol stack and exemplary network models. We use Doxygen [4] for documentation of our code and design concepts and advise users to do so with own extensions to our framework.

Extensibility: No matter how thorough the design and how voluminous the implementation, there will always be problems where the final user will need to adapt the software to his own needs. We therefore took care that new submodules can be easily derived from existing ones or created from scratch and integrated into the system just like the original ones.

Performance and Scalability: Simulations of mobile networks often involve hundreds of nodes (sensor networks for instance) and millions of events to complete. A key design factor in our work is that execution time is reasonable and memory requirements do not grow faster than the simulated network.

In the remainder of the paper we will present the architecture for mobility support and dynamic connection management, introduce the structure model of a mobile node and discuss its specification. In the last part we will comment on experiences made with the design and implementation and outline future work.

2 Mobility Architecture

2.1 Overview

The main questions to answer when simulating a mobile network are where to process mobility information and how to efficiently handle connections between nodes.

We decided to use a centralized approach for connection management since knowledge of the positions of all nodes is needed for this. Mobility is handled in a distributed manner since decisions how to move neither affect other nodes nor do they require global knowledge, as long as the current position is known. Figure 2 shows the communication and control relationships between the different modules.

The core of the mobility architecture is the *Channel Control* module, which dynamically sets up and tears down connections between nodes, depending on distance and the physical characteristics of the nodes. Movement is controlled by independent *Mobility Control* submodules in each node.

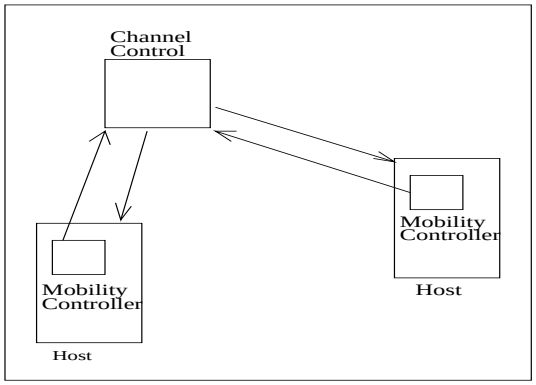

Figure 2: Architecture for mobility support and dynamic connection management

2.2 Channel Control

The *Channel Control* module takes care of establishing communication channels between nodes that are within communication distance and tearing down the connections when they move out of range. We decided to keep the concept of links between nodes, as opposed to direct message passing, since visible communication paths are an important source of information in early development stages.

Unfortunately, distinct links between nodes require at least two gate objects on each side. To make sure that a node does not run out of gates, when it comes into a crowded corner of the simulation area, one is tempted to define a pair of gates for each node in the simulation. When pursuing that strategy, the number of gates for a network of n nodes is $4n^2$ (since one needs a pair of gates per node, and a set of gates at the submodule and the enclosing compound module). A more memory-efficient approach is to create gates automatically if needed. However, resizing gate vectors requires reconnection of all gates in the vector. Hence, to reduce the computational overhead, gates are allocated and freed in bulk.

Once the simulation is running as desired, separate links can be replaced by direct message passing to speed up the execution or to enable simulations with large numbers

of nodes. This also solves any problems with dynamic gate allocation.

Updating connections between nodes is a computationally expensive operation. Calculating the distance between every pair in a network of n nodes has complexity $\mathrm{O}(n^2)$. Our *Channel Control* uses a more efficient approach, found also in GlomoSim [2], updating only connections between nodes that are in reasonable communication range.

Upon initialization, the *Channel Control* determines the interference distance between nodes, meaning the distance at which a node can still disturb the communication of a neighbor. This distance depends mainly on the channel model in use. The *Channel Control* can use a fixed value or calculate it based on a threshold SNR. The whole network is subdivided in quadrants with edge lengths of one interference range. A node within such a quadrant can only interfere with other nodes in the same or in the adjacent neighbor quadrants. Since separate node lists are kept for each quadrant, the *Channel Control* only updates connections between nodes in the corresponding lists, rather than iterating through all nodes.

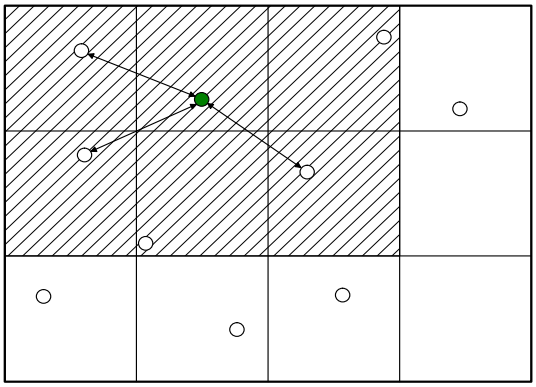

Figure 3: Interference area of a mobile node

This algorithm's complexity grows as a function of the network density, and is particularly efficient if the network size is large compared to the transmission range.

2.3 Mobility Controller

With object-oriented development it seems natural to make each node responsible for its movement. This helps to keep mobility state local and allows independent mobility patterns for each node.

We defined a *Mobility Controller* submodule for each host to handle mobility in the nodes. The *Mobility Controller* recalculates its position regularly and updates the graphical representation of its *Host* super module. It also communicates the current location information to the *Channel Control*. This communication is simply a function call, to avoid the overhead of another communication gate between the *Host* and the *Channel Control* and a message creation and destruction for each location update.

3 Structure of a Mobile Host

The overall structure of the host module has two basic sources: the ISO/OSI architecture and real-life hosts. Figure 4 shows the host module with its submodules. While this motivated the initial module structure that is presented here, there is nothing that would hinder a user of this framework to change this structure by deleting or modifying some modules or by introducing new ones.

The application layer, transport layer, and network layer are represented by compound modules. Below the network layer are one or more network interfaces. Each of these modules is representing a particular network interface of a host. A radio access point, for instance, will have at least two interfaces: one for the radio access and one for the fixed network connection.

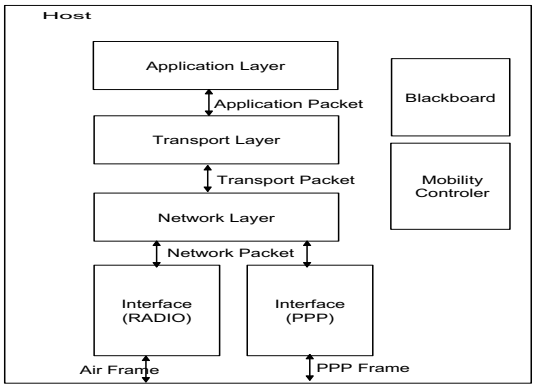

Figure 4: Structure of a mobile host

Some of these interface modules, especially the radio interface, have to be accompanied by other modules that coordinate their communication.

To achieve a "plug and play" use of these modules and allow others to use their own modules it is necessary to accurately define the interface for each module.

For each of these "top level" modules we have defined (regardless of the internal structure):

- The basic features that this module has to offer and enhanced features a particular type of this module might have.
- The format of the messages used by this module. Each top-level module should have its own message format that is exclusively used by itself and its submodules.
- The interface to the upper layer. To create a unified point of service between layers, each module has a simple adaptation submodule, where the fields of an internal message are set according to the incoming message (e.g. protocol type field in IP).
- The assumptions a layer makes about the lower layers.

3.1 Blackboard

To enable anonymous and group communication, we introduced the concept of a *Blackboard*. This feature is to be used by submodules within a host to exchange information

across layers, not unlike interprocess communication in a real computing device. There is exactly one *Blackboard* module per host. Modules that wish to use the *Blackboard* have to be derived from the *BlackboardAccess* class (which itself inherits from from *cSimpleModule*).

Communication takes place using the publish-subscribe paradigm. Modules that provide certain data (like a physical layer link sensing for carrier sense MAC protocols) can publish these on the *Blackboard*, using the *publish()* method family defined in the *BlackboardAccess* class. Internally, publications are mapped onto a sendDirect() call, so subscribers can actually find out who the publisher was.

Modules that are interested in certain types of information can subscribe for a certain topic using the *subscribe()* method of *BlackboardAccess* and will receive a notification message when matching data is published. The published data itself is not sent along with the notification message, but is stored in the *Blackboard* module and can be accessed via a *lookup()* method. This allows both asynchronous lookups and multiple access to the data.

This feature has proven very effective especially when integrating modules from different sources, and leads to an efficient coding style.

3.2 The Radio Interface

Overview

Since this is a *mobility* framework the radio interface is one of the most important and feature-rich modules. It consists of several submodules that are oriented on the ISO/OSI stack. Figure 5 shows the general structure of the radio interface. There are two modules (Decider, Adaptation) not represented in the ISO/OSI model – we describe them in separate sections.

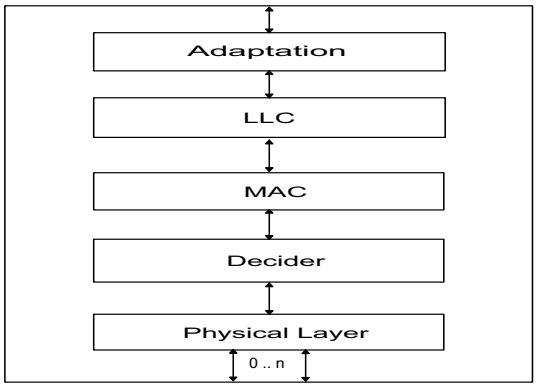

Figure 5: Radio Interface

The Decider Submodule

For investigation of many protocols, the pure bit error rate in the physical medium is sufficient. With others, like IEEE 802.11b, where headers are transmitted at a different rate than the packet bodies, a closer look at the transmission characteristics is necessary.

To support differen levels of abstraction in the physical layer, we have introduced the concept of a decider module. While the physical layer module is responsible for

recording the signal and noise power at different points o in time (at each transmitted *signal*), the decider module uses that information along with the applied modulation scheme to calculate the bit error rate or to mark erroneus bits if needed.

This decoupling of bit error simulation from transmission also allows to reuse the physical layer module with different modulation schemes.

The Adaptation Module

We suppose that all top-level modules of a host (see Fig. 4) should have their own message format and an *adaptation module* that works as "point of service" for the upper modules.

This layer should know about both message formats, and handle the control and communication between two adjacent layers (for instance the unicast/broadcast flag or TCP/UDP identifier in IP). In the implementation it would read out the fields from one packet, write it into the other message type and encapsulate/decapsulate the upper layer message.

We also believe that such a layered architecture will enhance interoperability between modules from different sources.

4 Conclusion and Future Work

We have presented a simple to use yet extensible framework to support simulations of mobile networks in OMNeT++. We are confident that this framework will find broad acceptance among research groups dealing with mobile networking. Through the well defined interfaces and clear documentation, we hope to encourage programmers to develop and contribute extensions to the *Mobility Framework*.

During our work we have learned a lot about architectural planning and design. Scalability with respect to memory usage for instance is a big issue when simulating large networks and requires early and wise design decisions. Another lesson learned was that designing a framework with extensibility in mind puts narrow constraints on where to store information. Parametrized module identifiers and our Blackboard concept are direct consequences of that.

In future work we will implement more sophisticated mobility architecture modules to reflect real geographical/topological settings and more complex channel models. Further modules for the different layers of the protocol stack are in preparation.

References

[1] The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`, 2002.

[2] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Rajive Bagrodia, and Mario Gerla. Glomosim: A scalable network simulation environment. Technical Report 990027, UCLA Computer Science Department, 13 1999.

[3] Witold Drytkiewicz, Steffen Sroka, Vlado Handziski, Andreas Köpke, and Holger Karl. Mobilty framework website. `http://www-tkn.ee.tu-berlin.de/research/mobility-omnetpp-sim`.

[4] Dimitri van Heesch. Doxygen. `http://www.stack.nl/~dimitri/doxygen/`, 2002.