Opportunistic Networking in OMNeT++

Ólafur Ragnar Helgason
Laboratory for Communication Networks
KTH, Royal Inst. of Technology
10044, Stockholm, Sweden
olafur.helgason@ee.kth.se

Kristján Valur Jónsson
Network Systems and Services Laboratory
Reykjavik University
103, Reykjavik, Iceland
kristjanvj04@ru.is

ABSTRACT

We describe mechanisms for simulating opportunistic and delay-tolerant networks in the OMNeT++ discrete event simulator. The mechanisms allow for simulating open systems of wireless mobile nodes where mobility- or contact traces are used to drive the simulations. This way mobility generation is separated from the core OMNeT++ protocol simulations which facilitates importing synthetic or real data from external mobility generators, real mobility tracking data or real contact traces. The paper describes the design and implementation of our mechanisms for OMNeT++ and gives an example of how we have used these to simulate opportunistic wireless content distribution in an urban environment.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event*

General Terms

Simulation, performance

Keywords

Simulation, Mobility, Opportunistic Networking, DTN, OMNeT++

1. INTRODUCTION

Performing experimentation on mobile wireless networks is a difficult task. It is non-trivial to capture meaningful measurement results because of the number of external factors influencing the measurements. Further, reproducing the environment between individual experiments is almost always impossible because of interference, fading, mobility patterns, randomness in the MAC layer contention, weather etc. Moreover, performing experimentation with a large number of mobile nodes is expensive and difficult to manage. These are some of the main reasons for why researchers and developers turn to simulation in evaluating protocols and mechanisms for wireless mobile networks.

OMNeT++ [1] is a public-source simulation platform that has primarily been used for simulating communication networks. Some of the main benefits of OMNeT++ are its modular design, clear structure and strong GUI support. The Mobility framework [2] for OMNeT++ provides extensions to the core simulator for supporting mobile wireless network simulations. However, the Mobility Framework lacks support for some of the features that are characteristic for the class of Opportunistic- and Delay-Tolerant Networks [3]. These are highly heterogeneous networks of mobile nodes, which are characterized by sporadic node contacts, where end-to-end connectivity cannot be assumed. Commonly, these sparse ad-hoc networks therefore use some form of mobility assisted forwarding (also known as store-carry-forwarding) to deliver messages.

In this paper we describe our design and implementation of mechanisms for OMNeT++ for simulating opportunistic networks of wireless mobile nodes. We propose simulations which are driven by tracefiles, and our mechanisms allow for simulation of open systems where mobile nodes can dynamically arrive and depart from a simulation scenario. Our design supports two separate approaches for simulating opportunistic networking, a *mobility driven* and a *contact driven* approach. With the mobility driven approach, mobility patterns of the nodes are specified in a mobility tracefile and during a simulation, node contacts arise when two or more nodes are within communication range. With the contact driven approach, the time of node contacts and their durations are specified in a contact tracefile. This approach does therefore not directly simulate the mobility of nodes but contact events, which are a consequence of node mobility, are used to drive the simulation.

A common design for these approaches is that we separate mobility generation of nodes from the core protocol simulations. There are various benefits associated with this. First, it facilitates importing both synthetic and real mobility or contact traces. In particular, it allows for importing traces from external mobility or contact generators and mobility patterns from GPS tracking methods or real contact traces. Second, it allows mobility patterns to be generated in more flexible high-level programming environments while the protocol simulator focuses on efficiency and short execution time. Also, by having a single mobility module that handles traces, instead of a special module for each type of mobility, simulator core code can be simplified. Third, with

OMNET++ 2008, March 03, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.3050

mobility traces it becomes easier to re-execute the same mobility scenario on different protocol parameters and it facilitates running same mobility scenarios on different simulators. Properties of the mobility process can also be analyzed more easily offline. Finally, a standardized trace file format for the mobile network simulation community would increase the inter-operability between different tools for generating mobility patterns and performing simulations.

Currently the Mobility Framework is mainly targeted at simulating closed systems of wireless mobile nodes. In Opportunistic networks, the intermittent connectivity of nodes arises because of node mobility and delivering messages is a non-trivial task because nodes carrying them may leave the area under consideration. Simulating networks of this kind needs, in many cases, to be done with an open system approach. The modules we have implemented for our mechanisms make use of some of the functionality provided by the OMNeT++ Mobility Framework and can therefore be seen as an extension to it. As an application of our work we describe how we have used our modules to simulate a wireless content distribution system that utilizes both opportunistic contacts between mobile nodes and wireless Access Points to distribute content to mobile nodes [4].

Our contributions to the OMNeT++ community are threefold:

- We present XML formats for node mobility and contact traces. The structure of the XML files is specified by XML-schemas.
- We provide OMNeT++ modules for dynamically creating and destroying nodes during a simulation run and for implementing our mobility- and contact-driven approaches.
- We have implemented a set of tools for generating node mobility patterns and for converting output from external mobility generators to our XML trace format.

Our code will be made available to other users of OMNeT++ free of charge.

The rest of this paper is organized as follows. In section 2 we describe the design of our mechansims and discuss some implementation issues. Section 3 describes how we have used our mechanisms to simulate an opportunistic wireless content distribution system. Section 4 discusses related work and in section 5 we conclude.

2. DESIGN AND IMPLEMENTATION

Our design allows highly dynamic mobile simulation scenarios to be created, where nodes can enter and exit the scenario during the course of a simulation run. The key components of our mechanism are the `NodeFactory`, `TraceMobility` and `ContactNotifier` modules. The `NodeFactory` dynamically manages nodes in the simulation, using scripted events in a *tracefile*, which specifies the create and destroy times of individual nodes, as well as waypoint updates and contact events. We use the same structure for generated nodes as is commonly done in modules employing the MF. A node is a compound module of OMNeT++ modules, one of which is a *navigator* module, derived from the `BasicMobility` class of MF. The `TraceMobility` module may be used, along with other `BasicMobility`-derived modules, when using the *mobility driven* approach. For a *contact driven* simulation, a `ContactNotifier` module assumes the role of the navigator and manages peer contact-establish and break events as specified in a contact tracefile.

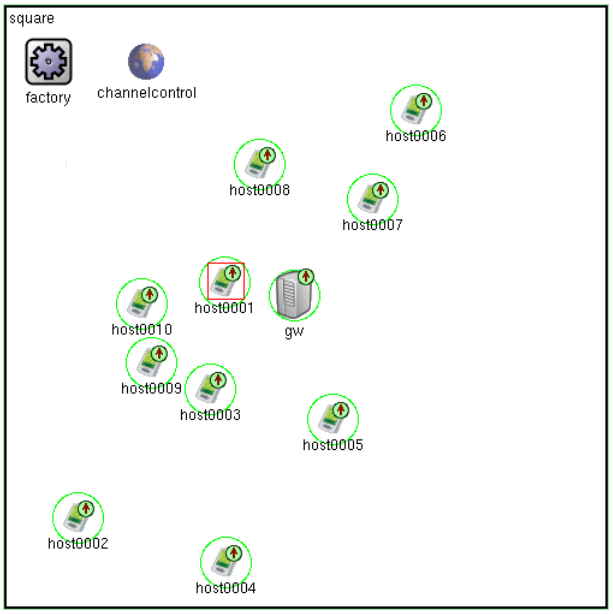


Figure 1: Simulation scenario with a `NodeFactory` object, mobile hosts and a single wireless gateway.

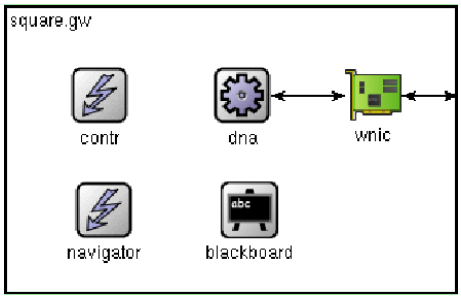


Figure 2: An example mobile node with a generic mobility module

Figure 1 shows an OMNeT++ simulation scenario with a single global *factory* object of type `NodeFactory`. The scenario contains a single stationary gateway node and several dynamically created mobile hosts. The `ChannelControl` from MF is required when working with `BasicMobility`-derived modules. Figure 2 shows a typical mobile node object. A generic mobility block, named *navigator* in this case, can take on the role of any `BasicMobility`-derived object at creation time, including that of `TraceMobility`. The `Blackboard` is utilized for notifying submodules within the node of location updates. The other modules making up the node are a wireless interface card, a protocol module and controller, all of which are specific to the depicted application. This example is further discussed in [5].

2.1 Tracefile Format

Two distinct types of tracefiles are supported: *Mobility traces* define creation time and position for each dynamically

created node in the scenario. Additionally, destroy events and waypoint updates may be associated with each created node. *Contact traces* define a set of contact establishment and break events for a population of nodes.

Both the mobility and contact tracefiles are in XML format, which imposes strict syntax and semantics on the file structure and allows parsing with open-source software libraries and tools. The structure of the tracefiles is defined by XML schemas, allowing tracefiles to be validated using commonly available schema validators.

Mobility trace

A *mobility trace* includes a series of node *create*, *destroy* and *waypoint* events, providing an arbitrarily fine-grained control over the lifetime and mobility of a population of nodes. A mobility trace can be created by mobility generators (e.g. [6]) or constructed from measurements, e.g. tracking the movement of a population of nodes equipped with GPS locators.

`Create` events specify the arrival of a node with a given *id* into the scenario at a given *time* and *location*. A node *type* designation enables nodes of various types and capabilities to be instantiated. A simulation could e.g. include pedestrians, vehicles and access points, all differing in capabilities and resources. A *mobility model* designation can optionally be included. The default is `TraceMobility`, but any `BasicMobility`-derived mobility modules can be specified here, allowing mobility models to be mixed in the same simulation. A optional *name* can be used to specify a string for display in a GUI environment or output traces. Similarily, an *icon* can be specified for easier visualization of different node roles or classes. Nodes are created at rest, and a `waypoint` command is required for a `TraceMobility` enabled node to start its journey. The parameters in effect for nodes with other mobility models, e.g. *random waypoint* (RWP), take effect at create time and must be specified in the initialization file for the scenario.

`Waypoint` events specify a waypoint change, i.e. the next destination on a nodes journey. The waypoint event specifies a node *id* and activation *time*. A node moves with a fixed *speed* on the leg between the current and destination waypoints. Note that pauses can be implemented by letting the journey between two successive waypoints be shorter in duration than the difference in their activation times. Waypoint events are only required when nodes change direction or velocity. Relatively compact tracefiles are thus feasible, even with a large number of generated nodes.

`Destroy` events specify the departure of a node with a given *id* from the scenario at a given *time*. The node is destroyed regardless of any remaining waypoint events.

Figure 3 shows an example of a simple mobility tracefile. A single node of type `SimpleNode` is created at $t = 0.0$ at $(x, y) = (0.0, 0.0)$ and starts moving towards $(x, y) = (10.0, 10.0)$ at $t = 10.0\ s$ with $v = 2.0\ m/s$. It then remains stationary at the final destination until it is destroyed at $t = 60.0\ s$.

```
<mobility-trace>
  <create>
    <time>0.0</time>
    <nodeid>1</nodeid>
    <type>SimpleNode</type>
    <name>mobile node 1</name>
    <icon>device/palm_s</icon>
    <mobilityModel>TraceMobility</mobilityModel>
    <location>
      <xpos>0.0</xpos>
      <ypos>0.0</ypos>
    </location>
  </create>
  <destroy>
    <time>60.0</time>
    <nodeid>1</nodeid>
  </destroy>
  <waypoint>
    <nodeid>1</nodeid>
    <time>10.0</time>
    <destination>
      <xpos>10.0</xpos>
      <ypos>10.0</ypos>
    </destination>
    <speed>2.0</speed>
  </waypoint>
</mobility-trace>
```

Figure 3: A simple mobility tracefile.

Contact trace

A contact trace contains peer contact establishment and break events for a population of nodes. Such a trace provides one further level of abstraction for a protocol simulator, in that it does not associate any location (nor mobility) with the nodes. Traces of this kind have recently been generated in opportunistic networking experiments, e.g. [7, 8, 9]. We point out that contact traces cannot be used concurrently with any mobility model, since contact information cannot, in general, be assumed to have any associated location data.

A contact trace consists of sequences of *contact* and *break* events:

`Contact` events signify a discovery event by a node, specified by a *nodeid*, of a peer with id *peerid* at a given *time*.

`Break` events signify a broken or lost contact between a node, specified by a *nodeid*, and a peer with id *peerid* at a given *time*.

`Create` and `destroy` events can optionally be employed to instantiate and remove nodes from the simulation. If omitted, the nodes are simply created when first encountered in the trace file.

Figure 4 shows a simple example of a contact trace. Two nodes, Node 1 and Node 2, are created at startup. Node 1 discovers its peer Node 2 at $t = 5.0\ s$. The contact is broken at $t = 10.0\ s$.

2.2 NodeFactory

The `NodeFactory` module dynamically creates and destroys nodes during the course of a simulation. A tracefile, as described in Section 2.1, is parsed at the initiation of the simulation, and an event created on the simulator event queue for each create and destroy event. The waypoint and contact events are also read at initialization and stored in an associated container, implemented as a STL map of lists, keyed by node id. No such events are however scheduled by the factory; the created nodes are initialized with this information upon creation and each node is then locally responsible for scheduling its own mobility/contact events.

When a scheduled *create* event is executed during a simulation run, a node of the specified type is instantiated using the appropriate OMNeT++ methods for dynamic node creation. If the node uses `TraceMobility` as its mobility

```
<contact-trace>
  <create>
    <time>0.0</time>
    <nodeid>1</nodeid>
    <type>SimpleNode</type>
  </create>
  <create>
    <time>0.0</time>
    <nodeid>2</nodeid>
    <type>SimpleNode</type>
  </create>
  <contact>
    <time>5.0</time>
    <nodeid>1</nodeid>
    <peerid>2</peerid>
  </contact>
  <break>
    <time>10.0</time>
    <nodeid>1</nodeid>
    <peerid>2</peerid>
  </break>
</contact-trace>
```

Figure 4: A simple contact tracefile.

module, it is initialized with the previously parsed *waypoint* update information. Similarly, a node employing a `ContactNotifier` module is initialized with the associated contact and break events at startup. Each node is henceforth autonomous in the sense that it is locally responsible for managing its own mobility or contacts. When a *destroy* event occurs during a simulation run, the event handling routine of the `NodeFactory` is called. It invokes the appropriate OMNeT++ node deletion methods. Note that the `NodeFactory` can be utilized to instantiate and destroy any type of node, regardless of its mobility model, when a *mobility trace* model is employed. A trace consisting solely of create and destroy events could thus be used to manage a population of nodes using e.g. a RWP mobility module or any of the existing ones derived from the `BasicMobility` class of the MF. The `NodeFactory` is implemented as a OMNeT++ simple module, and resides at the scenario level as shown in Figure 1.

Dynamically creating and destroying nodes on demand admittedly adds somewhat to the overhead of the simulation, since dynamic memory allocation can be a computationally expensive operation. However, this approach is in our opinion superior to the current practice in simulation of dynamic scenarios, where all nodes are created at startup and "flown" into the active region on demand and not destroyed until the simulation commences. This places great demands on the simulator, both in terms of memory and computing power, and significantly limits simulation scenarios with a large number of nodes. Another related performance issues with our current implementation is that the full simulation tracefile is read and parsed at startup. For large-scale simulations the tracefile can become somewhat big and thus the stored create, destroy and waypoint/contact events can add unduely to the memory requirements. We intend to solve this issue in our future work such that large simulation trace files are read in chunks as the simulation progresses.

2.3 TraceMobility Module

The `TraceMobility` module manages the mobility of a node according to a tracefile as described in Section 2.1. It is based on, and extends, the MF by deriving from the `BasicMobility` base class. A `TraceMobility` module can thus be used to enable trace controlled motion in any simulations which currently use the MF. When the `NodeFactory` creates a new node with a `TraceMobility` module, it passes the newly created node its corresponding list of *waypoint* events. The `TraceMobility` module manages its hosts mobility autonomously thereafter. A periodic event scheduled locally by the module triggers location updates, and the mobile node thus moves in a number of small steps to its next waypoint. After each step, the position of the node is updated and subscriber modules are notified through the `Blackboard` module of the MF. The granularity of the motion is controlled by the length of the update period, which is a configurable simulation parameter.

2.4 ContactNotifier Module

The `ContactNotifier` manages the contact events of a node when performing contact-driven simulations. The events are specified in a tracefile, as described in Section 2.1.

There is no actual mobility of nodes at the simulation level when employing the contact-driven approach. The contact trace simply lists the contact events, which may be a consequence of node mobility. Therefore, nodes in a contact-driven simulation do not have a mobility module, but instead employ the `ContactNotifier` to publish contact events to subscribing modules of the the node through the `Blackboard`. Although not a mobility module as such, the `ContactNotifier` serves a comparable purpose of notifying submodules of status change, and is indeed derived from the `BasicMobility` class of the MF. `ContactNotifier` can thus be viewed as replacing the `TraceMobility` module in the *navigator* role when using the *contact driven* approach to opportunistic networking simulation.

2.5 MobiTrace Toolbox

We have created a set of tools for mobility generation and for converting the output of external mobility generators to our XML mobility trace format. The `MobiTrace` toolbox consists of a set of scripts, implemented in the Python scripting language. Our design of trace-driven simulation allows separation of mobility generation from the protocol simulator, thus allowing node mobility to be scripted in a more flexible, high-level languages or imported from external generators. We will now briefly describe some of our tools.

UrbanMobility

The `UrbanMobility` tool generates mobility patterns of nodes (pedestrians, vehicles etc.) in an urban area. It takes as input a *map*, *routing probabilities* and a set of *generators*. The *map* specifies a grid of streets and intersections in the form of a graph of nodes and vertices, along with the node positions. Node *generators* can be attached to positions on the map where each generator is essentially an arrival process of nodes, with inter-arrival time and node speed given by some probability distributions. The *routing probabilities* specify node behavior at intersections, i.e. with which probability each street at the intersection is next selected by the node. We note that intersections can also have exits where nodes can leave the area (for underground transportation, area boundary etc.). The `UrbanMobility` tool generates a mobility trace of the format described in Section 2.1.

rwpy

`rwpy` is a simple implementation of the Random Waypoint mobility model (RWP), which generates mobility traces for a fixed number n of nodes, given a running time t_r. This application is essentially a proof-of-concept, as RWP mobility can easily be implemented in a MF-derived mobility module. A scripting approach to this simple mobility model is thus not strictly necessary. Like `UrbanMobility`, `rwpy` generates a XML tracefile of create, destroy and waypoint events, as described in Section 2.1.

u2tr

`u2tr` converts *UDel* [6] mobility traces to the tracefile format specified in Section 2.1. The UDel models include both simulation of mobility and propagation in an urban area. Currently we only support the mobility part of UDel models but we plan to include support for propagation modeling at a later time. The `u2tr` converter supports UDel's node types, given equivalently named compound node models in the OMNeT++ simulation.

mobgen

`mobgen` is a utility for converting a mobility specification in a flat text file to the XML format of Section 2.1. This utility has proven useful to create simpler scenarios and is thus described here in some detail. One event is specified per line in the input file, whose format is as follows:

```
{command} {time} {node} [...]
```

`command` is *create*, *destroy* or *waypoint*. `time` is in seconds from the beginning of the scenario. `node` is an integer uniquely identifying the node.

```
create {time} {node} {x y} [type]
```

`create` specifies creation of a mobile node at a specified `time` and `(x,y)` coordinates. `Type` is an optional parameter, specifying the type of node to be created. This string must correspond to an `BasicMobility`-derived module in the simulation.

```
destroy {time} {node}
```

`destroy` specifies the destruction of a node. A negative or zero `time` means that the node will be destroyed node after the last leg of its journey is travelled and its final pause is done. A destroy event with a specified time will destroy the node at that exact time, regardless of any remaining waypoint events.

```
waypoint {time} {node} {x y z} {velocity} [pause]
```

`waypoint` specifies the `(x,y)` coordinates of next waypoint, its `velocity` and, optionally, a `pause` time at the destination. Normally distributed variations of velocity and pause times are supported. An activation `time` less than zero means that the time of travel is deduced from the distance to the next waypoint and the specified velocity. If however the next consecutive event specifies a time, the velocity is deduced from the distance and travel time.

3. OPPORTUNISTIC WIRELESS CONTENT DISTRIBUTION

This section gives an example of how we have used the `UrbanMobility` generation tool and the trace mobility mechanism for simulating Opportunistic Wireless Content Distribution in an urban area[10].

In the Wireless Content Distribution system that we simulate, pedestrians in an urban area carrying a wireless communication device can exchange content while in communication range with another mobile node or a fixed access point[4]. We use the simulations to verify and investigate the impact of assumptions that we make for analytical studies of the system[11], to explore cases which are analytically non-tractable and to study transient behavior. The simulation part we describe here studies how well content spreads in an urban area where pedestrians are coming and going. In particular, we are interested in how content spreading is influenced by the arrival process of the nodes into the given area, the effect of the contact setup time, speed distribution of the nodes, communication range etc.

Simulation Setup

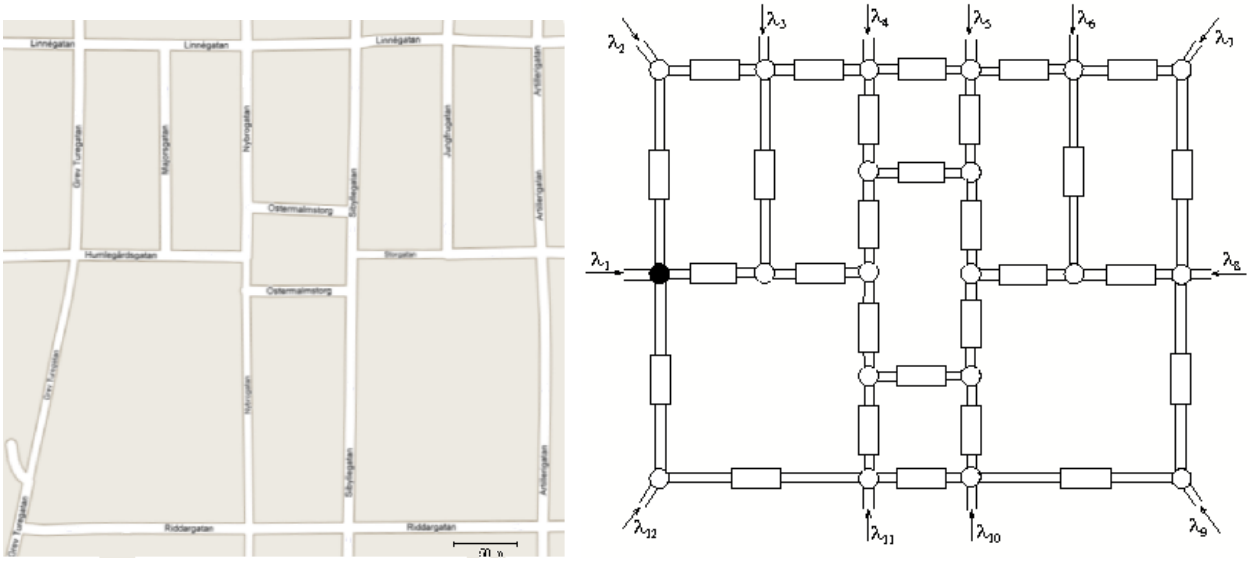
For simulating content distribution in an urban area we have modelled a part of the *stermalm* area in central Stockholm, shown in figure 5. The area is approximately $350 \times 380m^2$ and consists of 28 street segments whose lengths vary between 20 m and 200 m. To generate the pedestrian traffic, `UrbanMobility` takes as input the topology description of the area in the form of a connected graph. Then we attach Poisson arrival processes to streets that are entry points into the area and specify routing probabilities for the intersections. There are 12 intersections that connect this area to the outside world and we assume that the arrival rates to the entry points are $\lambda_i = \lambda, i = 1, ..., 12$. The intersection routing probabilities are configured as follows: Upon arriving at an intersection, nodes continue to move on the same street (if possible) with probability 0.5 or turn to other adjoining streets with equal probabilities.

In the simulations we set the node transmission range to $\Delta = 20m$ and the contact setup time is $t_{setup} = 20s$. Nodes choose their speed from a Uniform(1.00,1.86) distribution and thus the mean pedestrian speed is 1.43 m/s which has recently been measured as the average walking speed of pedestrians in Stockholm[12]. The position update period of the `TraceMobility` module is set to 0.1s.

Many of the measure we are interested in are steady-state averages of the stochastic process under inspection. When the stationary distribution of the system is known, the simulation model can be initialized according to it and the system starts in steady state. In our case, the stationary distribution is however not known, and we employ Welch's graphical procedure[13] to estimate the length of the initial transient.

Content Spread - Virtual Storage

How fast and how well content spreads in our system are two of its fundamental properties. Ideally we want content to spread as fast as possible to all those who are interested in obtaining it. To assess these two system properties we have simulated the spreading of an *alarm* signal. An alarm signal is a short message that is of interest to all nodes of the system and all nodes will help in spreading it by redistributing it to its peers.



(a) (b)

Figure 5: A part of a downtown Stockholm 5(a) and the corresponding network of street segments 5(b) used in our simulations.

$\lambda\,(s^{-1})$	$\lambda_{tot}\,(s^{-1})$	$1/\lambda_{tot}\,(s)$	$\rho\,(m^{-1})$
0.01	0.12	8.33	0.009
0.02	0.24	4.17	0.017
0.03	0.36	2.78	0.026
0.04	0.48	2.08	0.034
0.05	0.60	1.67	0.043

Table 1: Relationship between per-entry arrival rate λ, total arrival rate λ_{tot}, inter-arrival times $1/\lambda_{tot}$ and node density ρ.

In simulating the alarm scenario we first run the simulator for a warm-up period of length l time until the steady state has been reached and the total arrival- and departure rates for the area have converged. For convenience we say that the simulator is started at time $t = -l$ and steady state has been reached at time $t = 0$. At $t = 0$ a single stationary node (Access Point) at the entry 1 intersection (black in figure 5(b)) releases the alarm. Nodes will start obtaining the alarm, either from directly from the access point or from peers that already have the alarm. We assume that the alarm message is small and its transfer time after a contact has been set up is negligible compared to t_{setup}. A node will thus obtain the alarm whenever it makes contact with a peer or Access Point that has the alarm and this contact is of duration $T > t_{setup}$.

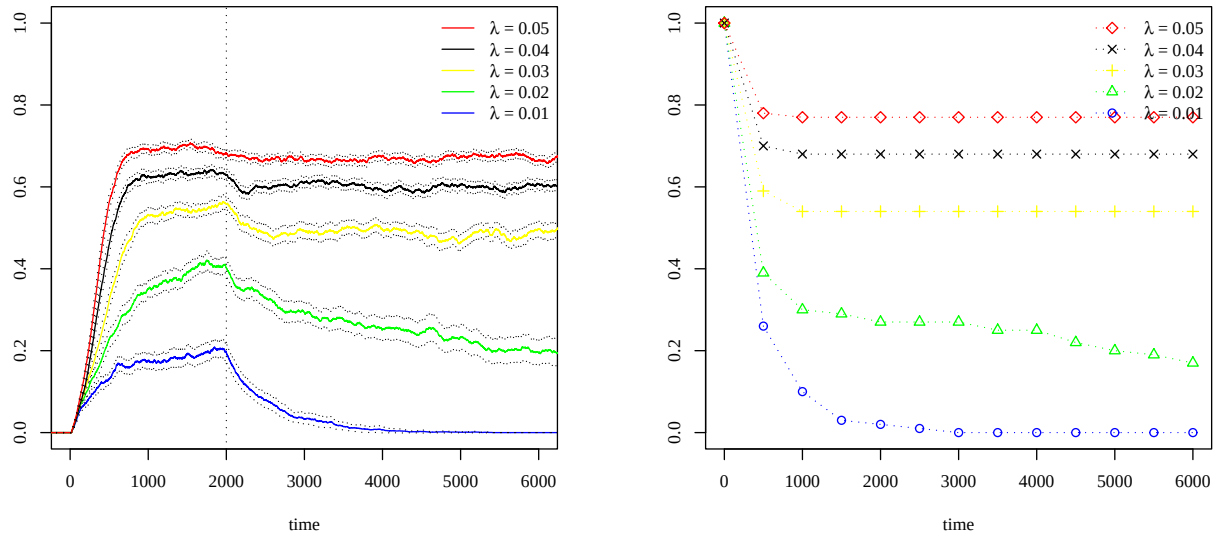
We have simulated for five different per-entry arrival rates. There are 12 entries into the area so the total arrival rate into the area is $\lambda_{tot} = 12\lambda$. Table 1 shows the relationship between the per-entry arrival rate λ, total arrival rate λ_{tot}, inter-arrival times $1/\lambda_{tot}$ and node density ρ. The node density ρ in nodes/meter is calculated as $\rho = \lambda_{tot} \cdot \overline{D} / \sum_i l_i$ where $\overline{D}$ is the average time a node spends in the area and l_i is the length of street i, $i = 1, ...28$.

For each arrival rate we have conducted 100 simulation runs and in each run we collect the time-series in 1 s intervals of the fraction of nodes carrying the alarm. In figure 6(a) we have plotted the average time-series for each of the arrival rates. The results confirm that the spreading of the alarm is strongly dependent on the density of the nodes in the are. For $\lambda = 0.05s^{-1}$ approximately 70% of the nodes in the area are carrying the content in steady state and it takes approximately 800s to reach the steady state average. For $\lambda = 0.01s^{-1}$ the average fraction of nodes carrying the alarm is much lower, or just below 20%, and it takes at least 2000s to reach this steady state average.

It is interesting to study what happens if we turn off the access point node that initially provides the alarm message. In figure 6(a) we turn off the access point when the alarm distribution has reached its steady state. For all the arrival rates we consider steady state has been reached at at $t = 2000s$. We note that when the arrival rate is low the alarm message disappears from the area because the density of nodes in the area is not high enough to facilitate the ad hoc spreading. This becomes evident at an arrival rate of $\lambda = 0.01$. At higher arrival rates, we see however that the spreading is not dependent on the access point support and the content becomes residential in the area as long as there are new nodes to which it can be passed. In other words, the spreading process exhibits a virtual storage effect: the content resides in the area even though there is no infrastructure support and nodes are coming and going.

To further strengthen our assertion of a virtual storage effect, we have studied a scenario with one single mobile node bringing an alarm into the area. It enters at $t = 0$ (after the mobility has reached steady state) and then we study the evolution of the availability of this message. In particular we are interested in determining if and when the alarm dies out from the area.

For each arrival rate under consideration, we have performed 100 simulation runs and in figure 6(b) we have plotted the fraction of runs where the alarm remains in the area as a function of time.



(a) (b)

Figure 6: 6(a): Fraction of nodes that have the content as a function of time. The access point is switched off at $t = 2000$ s. Dotted lines indicate 95% confidence intervals. 6(b): Fraction of runs where the alarm is resident in the area, plotted as a function of time. A single mobile node with the episode enters at $t = 0$.

For $\lambda = 0.05$, we see that in roughly 20% of the runs, the alarm disappears within 500 s. Interestingly for $\lambda = 0.05$, in all runs where the episode remains in the system after 1000 s, it will also be there when the simulation ends. This further strengthens our position that there is a virtual storage effect in the area: if the episode manages to spread initially to a critical number of other nodes, then it will be resident in the area. We see the same behavior for $\lambda = 0.04$ and $\lambda = 0.03$, although fewer runs achieve this effect (68% when $\lambda = 0.04$ and 54% for $\lambda = 0.03$). For $\lambda = 0.02$, the effect starts fading and the virtual storage effect is not visible anymore for $\lambda = 0.01$.

As a conclusion we have seen from our simulations that content will spread and remain in the area even if only a single node brings the content at these arrival rates of nodes. This indicates that content distribution between peer nodes is highly successful in this type of urban areas with pedestrian nodes.

4. RELATED WORK

Our work described in this paper is an extension to the Mobility Framework (MF) [2] library for OMNeT++ [1]. MF supports a variety of mobility models, including `BonnMotionMobility` which uses *BonnMotion* [14] generated mobility traces. BonnMotion is a mobility simulator, able to create mobility data from random waypoint, Gauss-Markow, Manhattan Grid and Reference Point Group Mobility model. However, the BonnMotion implementation in MF supports only simple destination update events. In contrast, we describe a solution for dynamic creation of a predefined number of nodes, in which a node can enter and depart the scenario, in addition to having a richer set of features, like multiple node classes.

ns-2 [15] is perhaps the best known simulator currently used in the field of communication networks. Traced mobility in ns-2 is supported through flat text files using `set` and `setdest` commands. GloMoSim [16] is a simulator for wired and wireless networks and uses the parallel discrete-event simulation language *Parsec*. A trace file of mobility events is supported, similar to the format supported by ns-2. This format is similar to the simple text file format proposed for our `mobgen` tool in Section 2.5. Both ns-2 and GloMoSim mobility traces can thus be supported in our system by conversion scripts. The vast majority of currently available mobility generation tools support those simulators, so this simple measure undoubtedly adds to the value of our system. ONE: Opportunistic Network Environment [17] is a Java-based simulation framework, intended for simulating opportunistic networks. It can generate node movement using a variety of mobility models, as well as importing traces from external mobility generators.

The UDel Models [6] are a suite of tools for simulating mobility and propagation in an urban environment. The mobility simulator is based on information from labor statistics, urban planning and traffic engineering communities and it has rich features for simulating pedestrian dynamics, arrival times at work, diurnal variations, vehicle traffic etc. Our `u2tr` tool can convert UDel mobility traces to the XML trace mobility format. We also plan to implement mechanisms for supporting the UDel propagation models in an OMNeT++ simulation.

Generic Mobility Simulation Framework (GMSF) [18] can generate ns-2 compatible mobility traces using a GIS-based model or a variety of mobility models. In addition, a generic XML format, similar to the one here proposed, can be exported.

There have been developed many tools for generating [19] [21] or analyzing [20] mobility for various other network simulators than OMNeT++, particluarly for ns-2, GloMoSim and QualNet. Our MobiTrace toolbox contains scripts for converting ns-2 tracefiles to our proposed XML format and these traces can therefore be used in OMNeT++ as well.

The *Crawdad* [22] project collects data for various wireless networking experiments. It contains data from some interesting recent experiments in Delay-tolerant networking where contact traces of people and vehicles are collected and used to evaluate routing algorithms for DTN. The goal of our contact-driven simulation approach is to utilize contact traces like these [7, 8, 9] to simulate protocols and mechanisms for opportunistic and delay-tolerant networks.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described the design and implementation of our mechanisms for simulating Opportunistic Networks in the OMNeT++ discrete event simulator. We advocate simulations driven by traces where mobility is separated from the core protocol simulations in OMNeT++. This approach facilitates importing synthetic or real data from external mobility generators, real mobility tracking data or real contact traces. We have described our design and implementation of mechanisms for conducting simulations driven by mobility or contact traces.

Our extensions to OMNeT++ and the Mobility framework consist of the specification of mobility- and contact traces, a module for dynamically creating and destroying nodes during the course of a simulation, modules that implement node mobility or node contacts from tracefiles and a toolbox of scripts for mobility generation and conversion of output from external mobility generators.

We have showed how our mechanisms can be used to simulate opportunistic content distribution in an urban environment. We use tools from our `MobiTrace` toolbox to model a real urban area and to generate pedestrian traffic in this area. Then we import the mobility traces and run simulations in OMNeT++ to evaluate the protocol behavior and the feasibility of the system.

This paper describes work in progress and enhancements and revisions are thus due to continue onwards. Future work involves extending our mechanism to not only capture node dynamics and mobility, but also propagation, fading and other properties of wireless communication. We also plan to increase compatibility with other simulators and mobility generators than those already implemented.

6. REFERENCES

[1] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *Proc. of European Simulation Multiconference (ESM2001)*, June 2001.

[2] W. Drytkiewicz, S. Sroka, V. Handziski, A. Köpke, and H. Karl, "A Mobility Framework for OMNeT++," in *Proc. of 3rd International OMNeT++ Workshop*, Jan. 2003.

[3] "Delay Tolerant Networking Research Group." http://www.dtnrg.org.

[4] G. Karlsson, V. Lenders, and M. May, "Delay-tolerant Broadcasting," *IEEE Transactions on Broadcasting*, vol. 53, pp. 369 – 381, Mar. 2007.

[5] K. V. Jónsson, Ó. R. Helgason, and G. Karlsson, "A Gateway for Wireless Broadcasting," in *Proc. CoNEXT 2007, Student Workshop*, (New York, NY, U.S.A.), December 10 2007.

[6] S. Bohacek, V. Sridhara, G. Singh, and A. Ilic, "The UDel Models - MANET Mobility and Path Loss in an Urban/Suburban Environment," tech. rep., University of Delaware, 2004.

[7] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms," in *Proc. IEEE Infocom*, (Barcelona, Spain), 2006.

[8] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks," in *Proc. IEEE INFOCOM*, April 2006.

[9] A.-K. Pietilainen and C. Diot, "Experimenting with Real-life Opportunistic Communications using Windows Mobile Devices," in *Proc. CoNEXT 2007, Student Workshop*, (New York, NY, U.S.A.), December 10 2007.

[10] M. May, G. Karlsson, Ó. R. Helgason, and V. Lenders, "A System Architecture for Delay-Tolerant Content Distribution (invited paper)," in *Proc. IEEE WRECOM*, Oct. 2007.

[11] Ó. R. Helgason and G. Karlsson, "On the Effect of Cooperation in Wireless Content Distribution," in *Proc. IEEE/IFIP WONS*, Jan. 2008.

[12] R. Wiseman, "The pace of life." www.britishcouncil.org/paceoflife.pdf, 23.10.2007.

[13] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. Reading, MA: McGraw-Hill, 3rd edt., 2000.

[14] "BonnMotion." http://www.cs.uni-bonn.de/IV/BonnMotion.

[15] "The NS-2 Network Simulator." http://www.isi.edu/nsnam/ns.

[16] "GloMoSim." http://pcl.cs.ucla.edu/projects/glomosim.

[17] "The ONE: Opportunistic Network Environment simulator." http://www.netlab.tkk.fi/tutkimus/dtn/theone.

[18] "Generic Mobility Simulation Framework." http://polar9.ethz.ch/gmsf.

[19] F. Bai, N. Sadagopan, and A. Helmy, "The IMPORTANT Framework for Analyzing the Impact of Mobility on Performance of Routing for Ad Hoc Networks," *AdHoc Networks Journal - Elsevier Science*, vol. 1, pp. 383 – 403, Nov. 2003.

[20] "ANsim." http://www.ansim.info.

[21] "CanuMobiSim." http://canu.informatik.uni-stuttgart.de/mobisim.

[22] J. Yeo, D. Kotz, and T. Henderson, "CRAWDAD: a community resource for archiving wireless data at Dartmouth," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 21–22, 2006.