ABSTRACT
This work describes 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 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, and reproducing the environment between individual experiments is almost always impossible because of interference, fading, mobility patterns, randomness in the MAC layer contention, weather etc. This is one of the main reasons for why researchers and developers turn to simulation in evaluating protocols and mechanisms for wireless mobile networks.

OMNeT++ [19] is a public-source, modular, simulation platform that has primarily been used for simulating communication networks. The Mobility framework [8] 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 [7]. 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 traces, 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 simulate the mobility of nodes but contact events are used directly to drive the simulation.

A common design with 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 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 a supplement 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 [12].

Our contributions to the OMNeT++ simulator and Mobility Framework are threefold:

- We present XML tracefile formats for node mobility and node contacts. The structure of the XML files is specified by XML-schemas. Tracefiles can therefore be verified against the format schema using standard XML tools.
- 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 is available for download from our project website at http://www.ee.kth.se/lcn/opponet/.

The rest of this paper is organized as follows. In section 2 we describe the design of our mechanisms 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 mobile nodes can enter and depart the scenario during the course of the simulation. Additionally, multiple mobile node classes with diverse capabilities and mobility patterns are supported.

Key components of our mechanisms are the NodeFactory module and the TraceMobility and ContactNotifier modules. The NodeFactory dynamically manages nodes in a simulation, in accordance to a mobility tracefile which specifies the create and destroy times of individual nodes. Mobility patterns can in addition be specified in the form of a waypoint list in the mobility tracefile. We use the same structure for nodes as is done in the MF. A node is essentially a collection of modules where each module is responsible for a certain task. Every mobile node has a dedicated mobility module for managing its own mobility locally. For a mobility-driven simulation, the TraceMobility module implements node mobility in accordance to the waypoint specification in the mobility tracefile. For a contact-driven simulation, a ContactNotifier module manages peer contact-establish and contact-break events of a node as specified by a contact tracefile.

Figure 1 shows an Omnet++ simulation scenario with a single global factory object of type NodeFactory. The scenario contains a single stationary gateway (gw) node and several dynamically created mobile hosts. ChannelControl is a module from MF for managing connections when nodes are in communication range and the Observer module is a monitoring object specific to the simulation model depicted.

### 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. The XML format imposes strict syntact and semantics on the file structure and allows parsing with open-source software libraries and tools. Our implementation uses the multi-platform libxml2 library for XML support. 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 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

\[1\]http://xmlsoft.org/
Moving towards position \( x = 10.0, y = 10.0 \) at time \( t = 10.0 \). The node is destroyed regardless of any remaining waypoint events.

**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 analyzer, in that it does not associate any location (nor mobility) with the peering nodes. Traces of this kind have recently been generated by experiments with opportunistic networking system such as [6, 17].

**Figure 2:** A simple mobility tracefile.

We point out that contact and mobility traces cannot be used concurrently, since contact traces 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 \( \text{nodeid} \), of a peer with id \( \text{peerid} \) at a given time. The contact event follows the contact timeline.
- **Break** events signify a broken or lost contact between a node, specified by a \( \text{nodeid} \), and a peer with id \( \text{peerid} \) at a given time. The break event follows the contact timeline.

Figure 3 shows a simple example of a contact trace. Node 1 discovers a peer with id 2 at time 0.0. The contact is broken at time 10.0.

**2.2 NodeFactory**

The NodeFactory module is responsible for dynamically creating and destroying nodes according a trace file, which is read and parsed at the initiation of the simulation. For every create and destroy event in the file, the NodeFactory schedules a corresponding OMNeT++ event on the simulator event queue. Upon initialization, the NodeFactory also reads all waypoint update events in a mobility tracefile, or contact events in a contact tracefile, and stores in an associated container, implemented as a C++ STL map of lists, keyed by node id.

The NodeFactory can create multiple types of nodes, mobile or stationary, of the node type specified in the tracefile. When a create event occurs during a simulation run, the event is passed to the event handler routine in the NodeFactory, which calls the necessary OMNeT++ routines to create and initialize a node of the given type. If the node is mobile, using TraceMobility as its mobility module, the factory passes the node its list of waypoints previously stored. Each mobile node is henceforward autonomous in the sense that it is locally responsible for managing its own mobility. Similarly, when a destroy event occurs during a simulation run, the event handling routine of the NodeFactory is called. The NodeFactory invokes the appropriate OMNeT++ node deletion routines which results in the cancelling of all mobility events that are scheduled for the node. Note that the NodeFactory can be utilized to instantiate and destroy any type of node, regardless of its mobility model. 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 model or any of the models created for MF.

The NodeFactory is implemented as an OMNeT++ simple module, residing at the scenario level, as shown in Figure 1.
2.3 TraceMobility module

The TraceMobility module manages the mobility of a node according to a tracefile as described in section 2.1. When the NodeFactory creates a new node with a TraceMobility module it passes the newly created node its corresponding list of waypoint events. Thereafter, the TraceMobility module schedules its hosts mobility autonomously and interpolates its position between waypoints. A periodic event scheduled locally by each mobility 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 other modules of the node compound module that need to know of the position change are notified through the Blackboard module from the MF. The granularity of the intermediate mobility steps is controlled by the length of the update period, which is a configurable simulation parameter.

Our OMNeT++ implementation of the TraceMobility module 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.

Figure 4 shows a typical compound node object from one of our simulations. 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 node mobility. In this case, can take on the role of any BasicMobility derived object at creation time, including that of node mobility. Therefore, nodes in a contact-driven simulation do not have a mobility module but instead the ContactNotifier contains a list of contact events, which it publishes to interested submodules of the the mobile node through the Blackboard. Submodules interested in the contact-establish and contact-break events must subscribe to this information and listen for it by overriding receiveBBitem, which is the callback function utilized by the Blackboard. Applications include simplification of node discovery algorithms at the node level, essentially abstracting it from the simulation model. Although not a mobility module as such, the ContactNotifier serves a comparable purpose of notifying submodules of status change. It can thus be viewed as replacing the TraceMobility module in the Navigator role when using the contact driven approach to opportunistic networking simulation.

2.4 ContactNotifier module

The ContactNotifier is a node module that manages the contact events of a node when performing contact-driven simulations. The contact events are specified in a contact tracefile as described in section 2.1.

In a contact-driven simulation there is no actual mobility of nodes at the simulation level. The contact trace only specifies the contact events which are a consequence of node mobility. Therefore, nodes in a contact-driven simulation do not have a mobility module but instead the ContactNotifier contains a list of contact events, which it publishes to interested submodules of the mobile node through the Blackboard. Submodules interested in the contact-establish and contact-break events must subscribe to this information and listen for it by overriding receiveBBitem, which is the callback function utilized by the Blackboard. Applications include simplification of node discovery algorithms at the node level, essentially abstracting it from the simulation model. Although not a mobility module as such, the ContactNotifier serves a comparable purpose of notifying submodules of status change. It 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 us to separate mobility generation from the core OMNeT++ protocol simulator and instead generate mobility of nodes in a more flexible, high-level scripting languages or to import them 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 of nodes, given a running time \( t \). 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.

u2tr

u2tr converts UDel [2] 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 later also include support for propagation. The u2tr converter supports UDel’s node types, given equivalently named and, properly initialized, compound node models in the OMNeT++ simulation.
SimpleMob 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} {details}
```

- **command** is create, destroy or waypoint. **time** is OMNeT++ decimal time from the beginning of the scenario. **node** is an integer uniquely identifying the node. **details** are defined for the create and waypoint events as described below.

- **create** {time} {node} {x y z} {type}

  **create** specifies creation of a mobile node at a specified time and location. **Type** is an optional parameter, specifying the type of node to be created. This string must correspond to an existing OMNeT++ module in the simulation.

- **destroy** {time} {node}

  **destroy** specifies the destruction of a node. Time is optional; a negative time will simply destroy the 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 setdest events.

- **waypoint** {time} {node} {x y z} {velocity} {pause}

  **waypoint** specifies the next location of the node, its velocity and pause time at the destination. Normally distributed variations of velocity and pause times are supported. Time is optional as with the destroy command; if less than zero, then the time of travel is deduced from the distance and 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[14].

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[12]. We use the simulations to verify and investigate the impact of assumptions that we make for analytical studies of the system[16], 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 Östermalm area in central Stockholm, shown in figure 5. The area is approximately 350 × 380 m² 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 \( \lambda_i \), \( i = 1, ..., 12 \). 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 \( \Delta = 20 m \) and the contact setup time is \( t_{\text{setup}} = 20s \). 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[16]. The position update period of the TraceMobility module is set to 0.1s.

Many of the measures 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.

In simulating the alarm scenario we first run the simulator for a warm-up period of length \( t \) 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 \( t_{\text{setup}} \). 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 \( T > t_{\text{setup}} \).

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 \( \lambda_{\text{tot}} = 12 \cdot \lambda \). Table 1 shows the relationship between the per-entry arrival rate \( \lambda \), total arrival rate \( \lambda_{\text{tot}} \), inter-arrival times \( 1/\lambda_{\text{tot}} \) and node density \( \rho \). The node density \( \rho \) in nodes/meter is calculated as \( \rho = \lambda_{\text{tot}} \cdot t \).
Figure 5: A part of a downtown Stockholm 5(a) and the corresponding network of street segments 5(b) used in our simulations.

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

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

where $D$ 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 area. For $\lambda = 0.05$s$^{-1}$ 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 $\lambda = 0.01$s$^{-1}$ 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 $t = 2000$s. 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.

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) [8] library for OMNeT++ [19]. MF supports a variety of mobility models, including BonnMotionMobility which uses BonnMotion [3] 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 in use in the field of communication networks. Traced mobility in ns-2 is supported through flat text files using set and setdest commands. GloMoSim [10] 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 SimpleMob 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 simulator, so this simple measure undoubtedly adds to the value of our system. ONE: Opportunistic Network Environment [7] 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 [2] 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 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) [9] 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.

Important [1] is a framework that aims to evaluate the impact of different mobility models on the performance of MANET routing protocols. The important project includes a mobility generation tool and a mobility analysis tool. The mobility generator generates traces according to some common mobility models such as Random Waypoint, Freeway mobility, Manhattan mobility model and more.

ANsim (Ad-Hoc Network Simulator) [11] is a tool to analyze the connectivity between node pairs for some common mobility models. It can also read mobility traces from an XML file similar in format to what we propose here. A major difference is however that we allow for node arrivals and departures during a simulation run.

CamuMobiSim [5] is a Java-based mobility generator. A meta-mobility model integrates a variety of variables, creating mobility traces suitable for e.g. ns-2, GloMoSim and QualNet.

The Crawdad [20] project collects data for various wireless networking experiments. It contains data from some

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 $t = 0$. 
interesting recent experiments in Delay-tolerant networking where contact traces of people[6] and vehicles[4] are collected and used to evaluate routing algorithms for DTN. Our contact-driven simulation approach has the goal of utilizing contact traces like this to simulate protocols and mechanisms for these types of 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 traces 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 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