Simulating Mobility and DTNs with the ONE

Ari Keränen, Teemu Kärkkäinen, and Jörg Ott
Aalto University School of Science and Technology
Department of Communications and Networking
{firstname.lastname@tkk.fi}

(Invited Paper)

***Abstract*—Delay-tolerant Networking (DTN) enables communication in sparse mobile ad-hoc networks and other challenged environments where traditional networking fails and new routing and application protocols are required. Past experience with DTN routing and application protocols has shown that their performance is highly dependent on the underlying mobility and node characteristics. Evaluating DTN protocols across many scenarios requires suitable simulation tools. This paper presents the Opportunistic Networking Environment (ONE) simulator specifically designed for evaluating DTN routing and application protocols. It allows users to create scenarios based upon different synthetic movement models and real-world traces and offers a framework for implementing routing and application protocols (already including six well-known routing protocols). Interactive visualization and post-processing tools support evaluating experiments and an emulation mode allows the ONE simulator to become part of a real-world DTN testbed. We examine a range of published simulation studies which demonstrate the simulator's flexible support for DTN protocol evaluation.**

***Index Terms*—Delay-tolerant Networking, Simulation, Performance Evaluation, Mobility**

I. Introduction

Personal communication devices, such as cellular phones, have enabled voice and data communications to mobile users, achieving global connectivity via infrastructure networks (cellular, WLAN). Local connectivity among the devices may additionally be obtained by forming ad-hoc networks since the mobile devices are virtually always turned on and have the necessary radio interfaces, processing power, storage capacity, and battery lifetime to act as routers. However, such usually sparse ad-hoc networks generally cannot support the type of end-to-end connectivity required by the classic TCP/IP-based communications due to frequent topology changes, disruptions, and network partitions caused by the node movement. Instead, asynchronous message passing (also referred to as *store-carry-forward* networking) has been suggested to enable communication over the space-time paths that exist in these types of networks (e.g., Delay-tolerant Networking, DTN [10], Haggle [27]).

Manuscript received September 30, 2009; accepted January 20, 2010.
This work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT) and by the Teknologiateollisuus ry in the REDI project.

The performance of such *opportunistic networks* may vary significantly, depending on how the mobile nodes move, how dense the node population is, and how far apart the sender and the receiver are. Delivery latency may vary from a few minutes to hours or days, and a significant fraction of the messages may not be delivered at all. The key factors are the routing and forwarding algorithms used and how well their design assumptions match the actual mobility patterns. No ideal routing scheme has been found so far.

Simulations play an important role in analyzing the behavior of DTN routing and application protocols. With typically sparsely distributed nodes, DTN simulations abstract from the details of the wireless link characteristics and simply assume that two nodes can communicate when they are in range of one another. This allows focusing on the evaluation of the DTN protocols—an approach we follow in this paper. Instead of fully modeling the lower layers we make simplifying assumptions about the data rates, the radio ranges, and thus the resulting transfer volumes.

In sparse node populations, the space-time paths, which are exploited by the store-carry-forward communications, are composed of the encounters between the nodes. The frequency, duration, and other characteristics of these encounters are largely dependent on the underlying mobility patterns. Evaluations of DTN protocols have used a large variety of synthetic mobility models as well as real-world mobility traces (which we review in section II). While synthetically generated node mobility allows for fine-tuning in many respects, this usually covers only limited mobility characteristics. In contrast, real-world traces often have only coarse temporal (e.g., scanning intervals in the order of several minutes) or spatial resolution (e.g., location determined from WLAN access point attachment) and coverage (e.g., only covering a campus area) and may exhibit biases due to the user group chosen for sampling.

All these approaches may provide complementary data points when assessing the performance of DTN protocols. What is important is that protocols are evaluated under different settings and that these settings can be fine-tuned to match the intended application scenario(s) as closely as possible. In this paper we extend our previous work [19] on the Java-based ONE simulator. We present new mechanisms for modeling multiple interfaces on a node, support for interference-limited links and a framework for modeling complex applications running on the nodes. Furthermore, we provide an overview of concrete use cases where the ONE has been successfully exploited to study a variety of aspects related to opportunistic, message-based communications. Finally we present a performance study of the simulator.

Our contributions are twofold: 1) The ONE simulator offers an extensible simulation framework itself supporting mobility and event generation, message exchange, DTN routing and application protocols, a basic notion of energy consumption, visualization and analysis, interfaces for importing and exporting mobility traces, events, and entire messages. 2) Using this framework, we implemented an extensive set of ready-to-use modules: six synthetic mobility models that can be parameterized and combined to approximate real-world mobility scenarios, six configurable well-known DTN routing schemes, a set of base primitives to design application protocols, a basic battery and energy consumption model, several input/output filters for interacting with other simulators, and a mechanism for the integration with real-world testbeds. The ONE simulator is designed in a modular fashion, allowing extensions of virtually all functions to be implemented using well-defined interfaces.

This paper is structured as follows: In section II, we review related work on DTNs and (related) simulations for mobility. We introduce the architecture and different features of the ONE simulator in depth in section III and describe how ONE is used in emulation setups in section IV. We show example use cases from published simulation studies and comment on the simulator's performance in section V. Section VI concludes this paper with a summary and points out future work.

II. Related Work

In this paper, we focus on communication performance in delay-tolerant ad-hoc networks comprising mobile nodes. Delay-tolerant Networking [10] is increasingly applied to enable communication in challenging networking environments, including sparse sensornets and opportunistic mobile ad-hoc networks. The DTNRG architecture [5] proposes a *bundle* layer as an overlay to bridge different (inter)networks. Nodes communicate via asynchronous messages of arbitrary size that are exchanged using the store-carry-and-forward paradigm. Messages have a finite TTL and are discarded when the TTL expires. They may also get dropped by a node due to congestion, yielding a best-effort service. Application protocols need to tolerate the delays resulting from the challenged environment and the risk that messages are not delivered in time or not at all. Typical performance metrics for evaluating DTN protocol performance are hence message delivery probability and latency.

Numerous routing and forwarding schemes have been proposed over the past years (refer to [35] and [23] for overviews). Different mechanisms are usually applied depending on whether the network is primarily of mobile ad-hoc nature (e.g., mobile devices carried by humans) or is based upon a (fixed or mobile) infrastructure (e.g., space networks, bus networks). Obviously, mixed networks exist as well, for example, with mobile users supported by infrastructure nodes.

The primary difference between various DTN routing protocols is the amount of information they have available to make forwarding decisions [13]. Ad-hoc DTNs usually apply variants of reactive protocols. Flooding protocols such as epidemic routing [33] do not use any information. Predictive protocols such as PRoPHET [21] use past encounters of nodes to predict their future suitability to deliver messages to a certain target whereas other protocols also exploit further (explicitly configured) schedule and context information per node [20]. Furthermore, they differ in their *replication strategies*, i.e., how many copies of a message they create which, in turn, has a direct impact on the load incurred on the network. Some protocols generate just a single copy [30] (e.g., First Contact [13], Direct Transmission/Delivery [30]), others a fixed number limited by the sender [31] [29] while epidemic [33] and probabilistic [21] routing potentially create an "infinite" number of messages. Scheduling strategies govern in which order messages are passed when a communication opportunity occurs between two nodes. Finally, queue management strategies define when and which messages are deleted, e.g., if congestion occurs.

For evaluating the performance of DTN routing protocols, manifold settings have been used, mostly including some type of node mobility. Mobility has been created (a) from synthetic mobility models, (b) taken from traces obtained from real-world measurements, and (c) by evaluating code in the real-world. While a few testbeds for (c) exist (such as DieselNet [3]) their flexibility is usually limited, large-scale operation "expensive", and their use is typically limited to those running the testbed. Such testbeds may also be used to obtain real-world traces (b) which can then be made available to other researchers.

Various projects have collected traces of contacts (peers, times, durations, and possibly positions) between Bluetooth devices [11], between users and/or wireless access points [8], among others. The CRAWDAD project[1] provides a repository where numerous real-world traces are available.[2] These traces offer insights into real-world interactions between mobile users from different angles and constitute a valuable data source for validating the mobility and connectivity characteristics obtained from synthetic models.

But also real-world traces have their limitations as—so far—the population analyzed in these traces is naturally very limited and may thus bias the results. Furthermore, the time granularity is often limited in order not to drain mobile device batteries too quickly: e.g., the Haggle iMotes uses sensing intervals of 5 min so that many contact opportunities may easily go undetected and contact durations can only be assessed equally coarsely. While this can be seen to reflect energy constraints, the scanning

[1]http://crawdad.cs.dartmouth.edu/

[2]The DieselNet traces are available at http://traces.cs.umass.edu.

interval cannot be adjusted afterwards. Finally, the results cannot be arbitrarily scaled, thus limiting what can be evaluated.

The only option for flexible and scalable simulations is thus (a) model-based synthetic mobility generation.[3] Mobility models range from simple entity models such as *Random Waypoint* to complex ones such as *Random Trip* [2] to group mobility to community models with major points of interest [4] to vehicular ones taking street maps into account (e.g., [6]). Node velocity and pause times may be adjusted to match pedestrians, vehicles, or other node types and smooth turns, acceleration and deceleration may be added to obtain more realistic behavior [1]. Specific models for vehicular networking furthermore consider additional constraints from simple road setups to real-world maps on one hand and simple non-interfering vehicles to vehicular interaction (distance, speed) based upon traffic flow models on the other. Approximations for footpath construction in and around buildings are used to make motion more realistic and transmission range and performance is adapted to model walls between mobile nodes [14].

In other areas (e.g., for epidemic spreading studies or traffic planning), more complex simulation models have been created mimicking the behavior of the population of an entire city [22]. Depending on the precise setting, the latter may not have the proper focus for evaluating ad-hoc interpersonal communications: TRANSIMS, for example, allows modeling a population and their interaction *at* certain locations or *in* vehicles, but does not include details on the way *between* such locations, which limits the suitability of the generated mobility data of pedestrians. In the case of TRANSIMS, detailed vehicle information could be made available and has been used for investigating MANET protocols [22].

Mobility generators for simple models are available for ns-2 and ns-3, as part of their respective toolsets or as specific extensions (e.g., [2]); both ns-2 and ns-3 accept suitably converted traces as input.[4] The latter also holds for various openly available DTN simulators (*dtnsim* [13] and *dtnsim2*[5]) and numerous ones tailored to specific research needs, based upon OMNet++, OPNET, or entirely newly developed[6], all of which have rather limited support for DTN routing protocols readily available. While ns-2 (and now ns-3) and OMNet++ offer sound generic open simulation platforms for packet-based communications and tools such as JANE [7] provide specific support for MANETs, generic support for DTN simulation is overall fairly limited. The ONE simulator contributes an environment for DTN protocol evaluation, embedding internal and external mobility models, different DTN routing schemes, and interactive inspection (similar to *nsnam* for ns-2) as well as post-processing.

[3]For an overview, see, e.g., [1], [4], [9] and the references therein.
[4]http://www.nsnam.org
[5]http://watwire.uwaterloo.ca/DTN/sim/
[6]E.g., Pydtn at http://www.umiacs.umd.edu/˜mmarsh/pydtn/ and http://www-net.cs.umass.edu/˜ellenz/software.html.

III. The ONE Simulator

At its core, ONE is an agent-based discrete event simulation engine. At each simulation step the engine updates a number of modules that implement the main simulation functions.

The main functions of the ONE simulator are the modeling of *node movement*, *inter-node contacts* using various *interfaces*, *routing*, *message handling* and *application interactions*. Result collection and analysis are done through *visualization*, *reports* and *post-processing tools*. The elements and their interactions are shown in Figure 1. A detailed description of the simulator is available in [16] and the ONE simulator project page [32] where the source code is also available.

Node movement is implemented by movement models. These are either synthetic models or existing movement traces. Connectivity between the nodes is based on their location, communication range and the bit-rate. The routing function is implemented by routing modules that decide which messages to forward over existing contacts. Finally, the messages themselves are generated either through event generators that generate random traffic between the nodes, or through applications that generate traffic based on application interactions. The messages are always unicast, having a single source and destination host inside the simulation world.

Simulation results are collected primarily through reports generated by report modules during the simulation run. Report modules receive events (e.g., message or connectivity events) from the simulation engine and generate results based on them. The results generated may be logs of events that are then further processed by the external post-processing tools, or they may be aggregate statistics calculated in the simulator. Secondarily, the graphical user interface (GUI) displays a visualization of the simulation state showing the locations, active contacts and messages carried by the nodes.

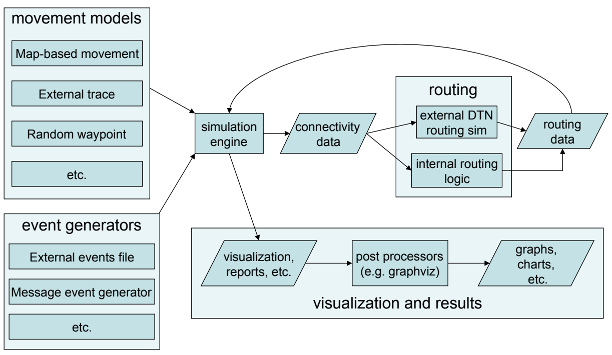

Fig. 1. Overview of the ONE simulation environment

A. Node Capabilities

The basic agents in the simulator are called nodes. A node models a mobile endpoint capable of acting as a store-carry-forward router (e.g., a pedestrian, car or tram with the required hardware). Simulation scenarios are built from groups of nodes in a simulation world. Each group is configured with different capabilities.

Each node has a set of basic capabilities that are modeled. These are *radio interfaces*, *persistent storage*, *movement*, *energy consumption*, *message routing* and *application interactions*. Node capabilities such as persistent storage that involve only simple modeling are configured through parametrization (e.g., peer scanning interval and storage capacity). More complex capabilities such as movement, routing and network interfaces are configured through specialized modules that implement a particular behavior for the capability (e.g., different mobility models or interference-limited radio interfaces).

Modules in each node have access to the node's basic simulation parameters and state, including the position, current movement path, and current neighbors. This allows implementation of, e.g., geographic routing and other context-specific algorithms. In addition, modules can make any of their parameters available for other modules in the same node through an intermodule communication bus. This way, for example, a movement module can change its behavior depending on the router module's state or a router module can adjust the radio parameters based on the node inter-contact times.

The focus of the simulator is on modeling the behavior of store-carry-forward networking, and hence we deliberately refrain from detailed modeling of the lower layer mechanisms such as media access control (MAC) algorithms or retransmissions due to corrupted link layer frames. Instead, the radio link is abstracted to a communication range and bit-rate. The bit-rate is dependent on the interface model and can be time-varying. Furthermore, the context awareness and dynamic link configuration mechanisms can be used to adjust both range and bit-rate depending on the surroundings, the distance between peers and the number of (active) nodes nearby as suggested, e.g., in [14].

The node energy consumption model is based on an energy budget approach. Each node is given an energy budget which is spent by energy consuming activities such as transmission or scanning and can be filled by charging in certain locations (e.g., at home). An inquiry mechanism allows other modules to obtain energy level readings and adjust their actions (e.g., scanning frequency as in [34], forwarding activity, or transmission power) accordingly.

Node movement capabilities are explained below in section III-B and the message routing capabilities in section III-C.

B. Mobility Modeling

Node movement capabilities are implemented through mobility models. Mobility models define the algorithms and rules that generate the node movement paths. Three types of synthetic movement models are included: 1) random movement, 2) map-constrained random movement, and 3) human behavior based movement.

The simulator includes a framework for creating movement models as well as interfaces for loading external movement data (see III-F). Implementations of popular *Random Walk (RW)* and *Random Waypoint (RWP)* are included (see Figure 2, top right). While these models are popular due to their simplicity, they have various known shortcomings [4]. It is also possible to completely omit mobility modeling and construct topologies based on static nodes (see Figure 2, top left).

To better model real-world mobility, *map-based mobility* constrains node movement to predefined paths and routes derived from real map data. Further realism is added by the *Working Day Movement (WDM)* model [9] that attempts to model typical human movement patters during working weeks.

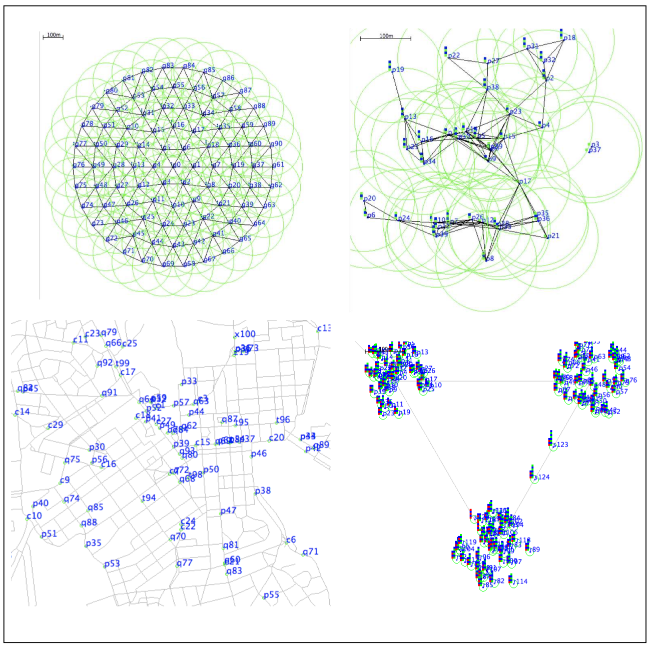

Fig. 2. Various mobility models in the ONE.

1) Map-Based Mobility: Map-based movement models constrain the node movement to paths defined in map data (see Figure 2, bottom left). The ONE simulator release includes three map-based movement models: 1) Random Map-Based Movement (MBM), 2) Shortest Path Map-Based Movement (SPMBM), and 3) Routed Map-Based Movement (RMBM). Furthermore, the release contains map data of the Helsinki downtown area (roads and pedestrian walkways) that the map-based movement models can use. However, the movement models understand arbitrary map data defined in (a subset of) Well Known Text (WKT). Such data is typically converted from real-world map data or created manually using Geographic Information System (GIS) programs such as OpenJUMP.[7]

In the simplest map-based model, *MBM*, nodes move randomly but always follow the paths defined by the map data. This results in a random walk of the network defined by the map data and thus may not be a very accurate

[7]http://openjump.org

approximation of real human mobility. A more realistic model is the *SPMBM* where, instead of a completely random walk, the nodes choose a random point on the map and then follow the shortest route to that point from their current location. The points may be chosen completely randomly or from a list of Points of Interest (POI). These POIs may be chosen to match popular real-world destinations such as tourist attractions, shops or restaurants. Finally, nodes may have pre-determined routes that they follow, resulting in the *RMBM* model. Such routes may be constructed to match, e.g., bus, tram or train routes.

2) Working Day Movement Model (WDM): While high-level movement models such as RWP, MBM, and SPMBM are simple to understand and efficient to use in simulations they do not generate inter-contact time and contact time distributions that match real-world traces, especially when the number of nodes in the simulation is small. In order to increase the reality of (human) node mobility, we have developed the Working Day Movement (WDM) model [9] for ONE.

The WDM model brings more reality to the node movement by modeling three major activities typically performed by humans during a working week: 1) sleeping at home, 2) working at the office, and 3) going out with friends in the evening. These three activities are divided into corresponding sub-models between which the simulated nodes transition depending on the node type and the time of the day.

Beyond the activities themselves, the WDM model includes three different transport models. The nodes can move alone or in groups by walking, driving or riding a bus. The ability to move alone or in groups at different speeds increases the heterogeneity of movement which has impact on the performance of, e.g., routing protocols.

Finally, WDM introduces communities and social relationships which are not captured by simpler models such as RWP. The communities are composed from nodes which work in the same office, spend time in the same evening activity spots or live together.

We have shown that the inter-contact time and contact time distributions generated by the WDM model follow closely the ones found in the traces from real-world measurements [9].

3) Composite Movement Models: Since movement models can be configured on per node basis, it is possible to combine multiple different types of mobility models in one simulation. This allows composite movement models to be created where, for example, some nodes follow map based movement along roads and others who walk around randomly within, e.g., a shopping center or a park.

Figure 2 (bottom right) shows a composite movement model made up of three clusters of nodes with a random-walk movement model constrained to a specific area and one group of nodes following a map based movement model between the clusters. This results in a message ferry scenario where nodes within separate clusters can communicate with each other through the message ferries.

C. Routing

The message routing capability is implemented similarly to the movement capability: the simulator includes a framework for defining the algorithms and rules used in routing and comes with ready implementations of well known DTN routing protocols.

There are six included routing protocols: 1) *Direct Delivery (DD)*, 2) *First Contact (FC)*, 3) *Spray-and-Wait*, 4) *PRoPHET*, 5) *MaxProp*, and 6) *Epidemic*. This selection covers the most important classes of DTN routing protocols: single-copy, n-copy and unlimited-copy protocols, as well as estimation based protocols.

Direct Delivery and First Contact are single-copy routing protocols where only one copy of each message exists in the network. In Direct Delivery, the node carries messages until it meets their final destination. In First Contact routing the nodes forward messages to the first node they encounter, which results in a “random walk” search for the destination node.

Spray-and-Wait [31] is an n-copy routing protocol that limits the number of message copies created to a configurable maximum and distributes (“sprays”) these copies to contacts until the number of copies is exhausted. Both variants of Spray-and-Wait suggested by its authors are included: in normal mode, a node gives one copy to a contact, in binary mode half of the copies are forwarded. Once only a single copy is left, it is forwarded only to the final recipient.

Three routing protocols perform variants of flooding. *Epidemic* [33] replicates messages to all encountered peers, while *PRoPHET* [21] tries to estimate which node has the highest “likelihood” of being able to deliver a message to the final destination based on node encounter history. *MaxProp* [3] floods the messages but explicitly clears them once a copy gets delivered to the destination. In addition, MaxProp sends messages to other hosts in specific order that takes into account message hop counts and message delivery probabilities based on previous encounters.

Routing capabilities of simulators such as ns-2 or *dtnsim2* can also be used in conjunction with ONE. Report modules can export mobility and connectivity data to other programs (modules for ns-2 and dtnsim2 are included) and external scripts are then used to import the results of routing simulation back into ONE (script for dtnsim2 is included).

If the external routing simulation was run with a contact schedule created by the ONE simulator, as described in section III-G, the whole process from node movement to external simulator’s routing decisions can be visualized and inspected using ONE.

Adding Routing Protocols: To evaluate new routing protocols in the ONE simulator, a new routing module needs to be created for the respective protocol. All routing modules inherit basic functionality, such as simple buffer management and callbacks for various message-related events, from the *MessageRouter* module. These callbacks are invoked by the simulator engine for all kinds of events, e.g., when a new message is created or a message is sent to the node. A router module needs to handle these events and also define actions to be carried out at every time step and the behavior when a new node comes into or leaves the node's radio range.

The basic functionality for all these events is common for the all currently implemented routing modules with internal routing logic. It is simply re-used for new routing protocols by extending the *ActiveRouter* module. This module provides functions for checking if any of the currently buffered messages are destined to a neighboring node, offering sets of messages to neighboring nodes, and dealing with successfully transferred and aborted message transfers, and it implements FIFO and random-ordering buffer management. For Epidemic, DD and FC routers no functionality beyond this is needed, making their implementations straightforward.

The following pseudocode listing shows how whole Epidemic router's logic is implemented in the callback that is called on every simulation update round.

```
update()
  if (isTransferring() OR
      nrofMessages = 0 OR
      nrofConnections = 0)
    return

  startedTransfer := exchangeDeliverableMessages()
  if (startedTransfer)
    return

  tryAllMessagesToAllConnections()
end
```

Since the implementation of Epidemic router transfers only a single message at a time, the update method does not do anything if there is an ongoing transfer. Also, if the routing module's message buffer is empty, or the node does not have any connections to other nodes, nothing needs to be done. Next, the routing module checks if any of the messages it has is for one of the nodes it is currently connected to, and if so, it starts to transfer such a message and returns from the method. Finally, all other messages are offered on all connections and a transfer is started if any of the connected nodes accept any of the messages.

More advanced routing modules may need to track node contacts and therefore implement the node discovery callback; e.g., PRoPHET and MaxProp perform their own book-keeping on past encounters this way. Below is shown the logic of the PRoPHET routing module's connection tracking callback.

```
changedConnection(Connection c)
  if (NOT c.isUp())
    return

  peer := con.getOtherHost()
  oldValue := predictions.get(peer)
  predictions.put(peer, oldValue + (1 - oldValue) * P_INIT)

  pForPeer := predictions.get(peer)
  foreach (node, p in peer.getRouter().getPredictions())
    pOld := predictions.get(node)
    pNew := pOld + (1 - pOld) * pForPeer * p * BETA
    predictions.put(node, pNew)
  end
end
```

PRoPHET router updates the delivery predictions every time a new connection comes up, so disconnection events are ignored. The delivery prediction for the contacted peer is updated as defined in the original PRoPHET paper [21]. Then, also transitive predictions are updated by asking peer's delivery predictions from its routing module, and iterating through them while updating the values with the transitive prediction formula. The update method of PRoPHET is similar to Epidemic router module's update, but instead of trying all messages to all connections, the messages are ordered by the delivery predictions and only messages for which the peer has a higher probability of delivery are forwarded. The message sending order is routing module specific and hence implemented in the PRoPHET module, but the routing framework provides methods for trying a set of messages for a set of connections, which also the PRoPHET module utilizes.

State may also be attached to messages using a tagging mechanism and thereby routing information may be forwarded hop-by-hop across the network. For example, the Spray-and-Wait router uses this mechanism to include a copy count in every message. A message can be tagged with multiple independent values, or even data structures, that allow arbitrary routing, or any other, data to be transferred with them.

When a simulation is run with the new routing module, the generic report modules gather the same performance data of the routing process as they do with the existing modules, so that comparing the performance of the new module to the existing ones is straightforward. In addition, it is possible to create routing module specific report modules that, e.g., read and interpret the data stored in messages with the tagging mechanism or query for parameters directly from the routing modules.

D. Application Modeling

In addition to mobility and routing modeling, ONE includes support for modeling application layer interactions. Two approaches are supported: high-level modeling of unicast and request-response interactions through *message generators*, and more detailed modeling through an *application layer framework*.

The message generator can be used to create traffic by drawing the source, destination, size and interval from a random distribution, or by reading them from an external message event file. The message event files can be created with the included tools or created based on real-world traces. Simple, interactive request-response interactions can be modeled by tagging the created messages to expect a response. When such a message is received, the receiver will generate a response message with a given size to the originator.

While the message generator based application modeling enables modeling of simple unidirectional or request-response interactions, more complex interactions can be modeled by building an application module using the application layer framework. The framework allows arbitrary application layer logic to be implemented and attached to the simulated nodes.

The conventional sockets API is built around a send-receive model with well-known source and destination addresses. However, DTN applications often require more complex interactions between the application layer and the bundle transport layer. For example, application gateway nodes may wish to pick up all messages of a particular application and forward them to the classic Internet (e.g., an email gateway [12]). Other applications may wish to terminate the spread of a message based on some application layer condition (e.g., search termination [24]). In general, application layer logic may wish to receive, inspect, modify or terminate any message regardless of whether the host is the destination of the message.

The application layer framework allows the application modules to: 1) receive messages (all messages or only messages with a specific application identifier), 2) modify messages by altering, appending or removing any values or properties, 3) signal to the router that the message should be dropped, 4) generate messages, and 5) execute arbitrary logic and actions every time step. Any number of application instances can be attached to a single host in the simulation.

The messages inside ONE do not carry any application payloads. Instead, generic name-value pairs, called *properties*, can be attached to the messages. Applications can model different types of messages by adding specific properties, e.g., a ping message might have a property named “type” with a value “ping” and the response message may have the type “pong”. This allows arbitrary protocol interactions to be implemented.

The application layer framework also includes support for application specific report modules. The application instances can raise application events with specific types and parameters. Report modules can listen to these events and create application-specific reports based on them.

Adding Application Protocols: To simulate a new application with complex interactions, a new application module and a report module must be constructed. The application module can then be attached to any number of hosts in the simulation scenario.

Every new application module inherits basic functionality from the base *Application* module. This includes a callback for handling incoming messages, an update callback for every simulation time step, and the ability to raise application events that are communicated to all the registered report modules. The new application implements the callbacks to provide the custom application behavior, and sends application events to the report module.

To create a simple ping application, the application module would implement the update callback to generate ping messages at regular intervals. These messages are handed to the router and forwarded in the simulation similarly to all the other messages. The handle callback is implemented to receive the ping messages and generate a pong message in response. Furthermore, the application module will generate events every time ping or pong messages are created or received. The report module can then calculate delivery probabilities based on these events.

The following listings demonstrate an implementation of a simple ping application module.

```
handle(Message msg, DTNHost host)
  if (msg.to = host AND msg.property("type") = "ping")
    pongMessage.to := msg.from
    pongMessage.property("type") := "pong"
    host.sendMessage(pongMessage)
    raiseEvent("RECEIVED_PING")
  else if (msg.to = host AND msg.property("type") = "pong")
    raiseEvent("RECEIVED_PONG")
  end if
end

update(DTNHost host)
  if (getSimulationTime() > nextSendTime)
    pongMessage.to := getRandomHost()
    pongMessage.property("type") := "ping"
    host.sendMessage(pongMessage)
    nextSendTime = getSimulationTime() + randomInterval()
    raiseEvent("SENT_PING")
  end if
end
```

E. Link Layer Modeling

The focus of ONE is in modeling the network layer store-carry-forward interactions and the link layer is abstracted to bit-rate and range. However, the simulator supports generic *interfaces* which can be used to model nodes with multiple radios (e.g., Bluetooth, WiFi) or to create links with time-variant characteristics (e.g., interference limited links).

Each node within the simulator can be configured with an arbitrary number of named interfaces, with each interface having a different type and/or different parameters. Nodes configured with the same interface can create connections between each other. This allows nodes to be configured with short range but high bit-rate interfaces (e.g., Bluetooth) and with longer range but lower bit-rate interfaces (e.g., cellular). Furthermore, it allows a subset of nodes to create a backbone network by using a long range interface for communication within the subset, while using a short range interface to communicate with other nodes.

Beyond a simple constant bit-rate interface, ONE includes an interference-limited interface. This interface has time-variable bit-rate that is calculated based on the number of other transmitting nodes within the vicinity of the node. This leads to a more realistic radio model where the interference from multiple simultaneously transmitting nodes decreases the bit-rates for nearby nodes. The interference modeling could be extended to include, for example, location dependent variable bit-rate to model the effects of fading due to buildings or other factors.

F. External Interfaces

An important feature of ONE is its ability to interact with other programs and data sources. The simulator has interfaces, e.g., for node movement, connectivity and message routing traces.

It is possible to generate node movement using an external program, such as TRANSIMS or BonnMotion[8],

[8]www.cs.uni-bonn.de/IV/BonnMotion

or from a real-world GPS trace such as the ones available from CRAWDAD. Such a trace file needs to be converted to a suitable form for the External Movement module. The distribution package contains a simple script that can convert TRANSIMS output to this format.

Instead of node locations, many real-world traces contain only information about connections between nodes. This kind of traces can also be imported to ONE and used for routing simulations. For this purpose we have created conversion scripts, e.g., for the DieselNet traces. We have also generated connectivity traces from the real-time location data of trams in the Helsinki area.[9]

Like node movement and connection traces, also message traces can be imported to ONE. These may include message creation and deletion events, and starting and cancellation of message transfers. This functionality is especially useful if ONE is used for analyzing traces generated by other DTN routing simulators or even real-world traces.

In addition to reading output of other programs, ONE can also generate input traces for them. It has report modules whose output is compatible with dtnsim and dtnsim2 connectivity trace input. In a similar fashion, it is also possible to create mobility traces using a mobility report module. If properly formatted, these traces are usable in, e.g., ns-2. This way ONE can function as a general purpose mobility simulator.

While report files are an easy way to interact with other programs, a report module can also communicate in real time with them. This approach was used with real world DTN integration, described in section IV.

G. *Reporting and Visualization*

ONE is able to visualize results of the simulation in two ways: via an interactive Graphical User Interface (GUI) and by generating images from the information gathered during the simulation.

Figure 3 shows the GUI displaying the simulation in real-time. Node locations, current paths, connections between nodes, number of messages carried by a node, etc. are all visualized in the main window. If a map-based movement model is used, also all the map paths are shown. An additional background image (e.g., a raster map or a satellite image of the simulation area) is shown below the map paths if available. The view allows zooming and interactive adjusting of the simulation speed.

The GUI produces a filtered log of simulation events, such as contacts and message transfers. Filters are used to show only interesting events, or to pause the simulation when a particular type of event occurs. Selecting a node from a list or a log message opens it for closer inspection. This allows retrieving further information about the messages a node is carrying and about the routing module's state.

While the GUI is good for getting an intuitive overall picture of what is happening during the simulation,

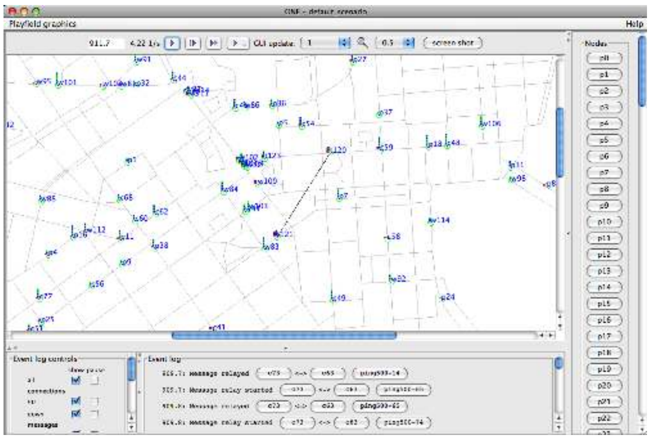

Fig. 3. Screenshot of the ONE simulator's GUI

more rigorous ways to visualize node relations, message paths and performance summaries are provided by post processed report files.

ONE includes report modules that can create Graphviz[10] compatible graph files. Figure 4 shows how these graphs visualize node connections and the paths that the messages have traveled in the network. Likewise, for visualizing how messages are spread in the network as a function of time, a message location report module can provide this data and an animator script will turn the data into a GIF animation.

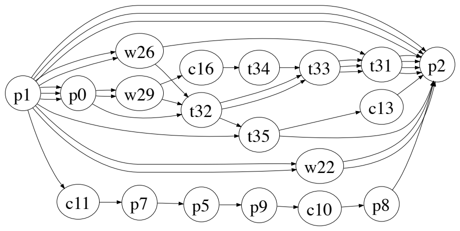

Fig. 4. Example message paths from node p1 to p2

The simulator includes a message statistics report module that gathers statistics of overall performance (number of created messages, message delivery ratio, how long messages stay in node buffers, etc.). A post processing script that plots the report module's output is also included.

H. *Creating Simulation Scenarios*

Simulation scenarios are built by defining the simulated nodes and their capabilities. This includes defining the basic parameters such as storage capacity, transmit range and bit-rates, as well as selecting and parameterizing the specific movement and routing models to use. Some simulation settings such as simulation duration and time granularity also need to be defined.

The simulator is configured using simple text-based configuration files that contain the simulation, user interface, event generation, and reporting parameters. All

[9]http://netlab.hut.fi/tutkimus/dtn/theone/trace_1week_30m.txt.zip

[10]http:// www.graphviz.org/

modules have their high-level behavior defined by their Java code implementation, but the details of their behavior is adjustable using the configuration subsystem. Many of the simulation parameters are configurable separately for each node group but groups can also share a set of parameters and only alter the parameters that are specific for the group. The configuration system also allows defining of an array of values for each parameter hence enabling easy sensitivity analysis: in batch runs, a different value is chosen for each run so that large numbers of permutations can be explored.

If configuring existing implementations of different modules is insufficient for creating a specific scenario, ONE can also be extended with new code. We have introduced several hooks for extensions without a need for any changes in other parts of the simulator code. This allows sharing new modules as plugins and using them with different versions without needing to patch rest of the simulator. Routing modules, movement models, event generators and report modules are all dynamically loaded when the simulator is started. Hence, when creating a new module, user only needs to create and compile a new class, define its name in the configuration file, and the simulator automatically loads it when the scenario is started. All these modules can also have any number of settings defined in the configuration files and these settings are accessible to the module when it is loaded.

IV. Real-world DTN Integration

The ONE simulator has been designed to be used in conjunction with DTN2[11] in order to provide a realistic environment for testing and evaluating real-world DTN applications. The DTN2 bundle router (*dtnd*) is the reference implementation of the DTNRG bundle protocol [28]. DTN2 implements *convergence layers* such as TCP, UDP and Bluetooth, *routing algorithms* such as epidemic and PRoPHET, and *neighborhood discovery* mechanisms such as Bonjour. Applications can use the DTN2 API to take advantage of the bundle delivery services provided by the bundle router.

There are two interaction models between ONE and the DTN2 reference implementation: 1) Controlling *dtnds* through their console interface based upon connectivity data exported by the ONE simulator, and 2) real-time integration using the simulator to emulate all or parts of a DTN. In the first approach, ONE is only used for providing realistic, mobility model based connectivity characteristics to a network of DTN2 nodes. In the second approach, ONE is used to emulate all aspects of a DTN network including mobility, routing, radio link characteristics and node storage constraints while DTN2 is used mainly for providing the application interface.

A. DTN Controller

Traces generated by ONE's connectivity report modules are suitable to control the link status between *dtnd* instances. This requires an external DTN Controller that reads the contact trace files produced by the ONE simulator and controls the *dtnds* through their console interfaces. The connectivity traces report each event of a link between two nodes going up or down and the time instance when it occurred. The controller reads these events sequentially and instructs the corresponding *dtnd* instances to open or close the specified link. Real-time operation is achieved by scheduling the control commands according to the trace file's time-stamps.

This approach allows the network of bundle routers to run independently of the ONE simulator instance. This is practical for creating long lived, robust testbeds. We have experimented with a DTN Controller (not part of the ONE release) to create a testbed network that simulates buses running between Ruoholahti in downtown Helsinki and the Helsinki University of Technology in Espoo roughly eight kilometers away. This was done by generating a simulation scenario for ONE which modeled buses traveling between the two locations over a 24 hour period and then running a simulation to produce the connectivity trace. Our DTN2 controller uses the trace to open and close links between *dtnd* instances resulting in a connectivity pattern that resembles having real bundle routers in the buses. The testbed is used for experimenting with various applications, such as a DTN camera application that takes and returns pictures upon receiving a corresponding request.

B. DTN Emulation Support

In order to take advantage of all of the ONE simulator's features when creating an emulated environment for real DTN applications, real-time integration with the DTN2 reference implementation is required. For this purpose ONE implements the External Convergence Layer Interface of DTN2. This allows the simulator to connect to *dtnd* instances as an external convergence layer adapter, appearing as a link in the DTN2 link table. Any bundles passed onto this link by DTN2 will appear as new messages in ONE. The simulator also controls *dtnds* via the console interface to automatically set up the routing to pass bundles to and from the link.

It is possible to connect any number of DTN2 instances at the same time. Each instance is configured to match a specific node inside the simulation with a mapping from the Endpoint Identifiers (EIDs) used in the bundle protocol to the node IDs used by the ONE simulator. The EID mapping uses regular expressions allowing one node to have any number of matching EIDs as well as allowing one EID to match any number of nodes. When a bundle arrives from *dtnd*, ONE matches the destination EID against the configured EID to node ID mappings and generates a bundle message to each matching destination. After this the bundle messages are treated identically to all other messages inside the simulation. Once a bundle message reaches its destination inside the simulation it is delivered to the *dtnd* instance connected to the destination node.

[11]http://www.dtnrg.org/wiki/Code

The DTN Emulation Support has been used to provide a realistic scenario for demonstrating an implementation of mail over DTN [12]. A simple scenario for the ONE simulator was constructed that mimicked the layout of an exhibition hall with a number of DTN-capable nodes moving around the area. Multiple devices, such as Internet tablets, running *dtnd* and DTN email applications were connected to the simulator. As messages were sent from these devices they appeared in the ONE simulation, traveled around until they found their destination, and then appeared in the real device they were destined to. ONE was run in the GUI mode showing in real-time the messages as they traveled between the nodes.

V. Simulator Usage

ONE has been used to evaluate a number of mobility models, routing protocols, applications and other DTN mechanisms. We do not show new simulation results here, but refer to a number of simulations and results from previous studies. New results for the simulator performance are presented in Section V-B.

A. Use cases

ONE can be used to study the *basic store-carry-forward behavior* by recording message delivery statistics such as latencies and delivery probabilities and their distributions in various scenarios. Such simulations help to build understanding of the performance of a complete DTN message delivery service under different routing algorithms, mobility models and node characteristics.

Beyond the basic analysis of message delivery in different scenarios, more detailed studies of *mobility models*, *routing*, *applications*, and *other DTN mechanisms* can be done by extending various aspects of the simulator and statistics reporting.

Basic Store-Carry-Forward Behavior: A previous technical report [17] describes the basic feature set of ONE. The report presents simulations based on a scenario of interpersonal communication between mobile users in Helsinki downtown area using modern mobile devices. The simulation scenarios use existing modules and no custom extensions to the simulator are created.

The simulations include *pedestrians*, *cars*, *trams* and stationary *throw-boxes*. A sparse scenario with 100 nodes and a more dense scenario with 500 nodes is simulated, both with six or less throw-boxes. Both personal area networking (PAN) and wireless local area networking (WLAN) with communication ranges of 10 and 30 meters respectively are studied. The mobile nodes have buffer sizes of 5 and 20 MB while the stationary throw-boxes have buffer sizes of 50 and 200 MB.

Each node generates one message per hour on average. The message sizes are randomly distributed between 100 KB and 2 MB. Both one-way and simple request-response interactions are simulated.

Four of the movement models described in Section III-B are studied: RWP, MBM, SPMBM, and RMBM. The map based models use a map of downtown Helsinki with an approximate size of 4.5 by 3.4 km. RWP movement uses the same area but is not constrained by the map. Epidemic, Direct-Delivery, PRoPHET, and multiple variations of Spray-and-Wait algorithms are simulated.

The simulations are run over a number of combinations of the above settings. The results show relative distributions of inter-contact times and contact durations. This revealed, among other things, that the addition of higher velocity nodes such as cars and trams and motionless throw-boxes did not alter the contact duration distribution significantly.

The results also show message delivery probabilities and latencies for the various routing algorithms and movement models. These reveal that delivery latencies are in the order of half an hour to couple of hours, while delivery probabilities range from less than 5% to over 35% depending on the configuration. Multiple parameters, such as the mix of different node types, message size and message lifetime, have observable impact on the message delivery.

Further work by the authors [19] extend the above simulations by introducing a new energy model to the simulator along with explicit intervals for scanning for neighbors. The simulations are also run with a larger map (8.3 by 7.3 km) and use WDM mobility in addition to the movement models used in the above report. The energy model assumes each node has a battery with a limited energy budget. Energy is subtracted from the budged every time a node transmits or scans the area for other nodes. Nodes that have exhausted their budged can no longer communicate with other nodes.

The results of the simulations show that as the realism of the mobility increases, the contact durations increase as well. This is explained by the fact that in reality nodes tend to stay close to each other for extended periods of time when, e.g., traveling on a bus, working in an office or spending time in a restaurant. The same effect accounts for more realistic models being less susceptible to adverse effects due to increasing scanning intervals. Another observation is that in more realistic mobility models some nodes tend to be more central than others with more social interactions. This causes the batteries of those nodes to drain more quickly than other nodes.

Mobility Modeling: ONE has been used both for modeling increasingly realistic movement of mobile nodes and as a generator of mobility traces for other simulators.

A study by F. Ekman et al. [9] introduces the WDM (see Section III-B2 for explanation of the model) with a set of ONE simulations that analyzes the impact of lowering the abstraction level of the mobility model, bringing in more realism to the node movement.

The simulations divide the Helsinki area into four *districts*, each with a set number of homes, offices and meeting spots. Three of these districts model primarily residential areas while one models the downtown area. There are bus lines both within and between the districts. In total 1000 nodes, 200 offices and 24 meeting spots within 7 by 8.5 km area are simulated. 50% of the nodes are able to travel by car, while the rest will travel by foot or by bus. In addition, 10 nodes following the SPMBM movement were used as background traffic.

The simulations are focused purely on the movement of the nodes and the resulting inter-contact times, contact durations, and contacts per hour. No routing or message passing is considered. Comparisons are made against an RWP model and real world traces.

The results of the simulations show that the new model produces contact characteristics that are similar to those observed in real-world traces, while the RWP model used as a comparison could not produce such characteristics. This implies that a synthetic mobility model with a low abstraction level can create more realistic mobility than models with higher abstraction levels. By changing the map from Helsinki map to a Manhattan-like grid, the results show that the underlying map had no significant effect on the contact patterns.

Contact traces generated by the synthetic mobility models in ONE have also been used by M. Pitkänen and J. Ott [26] with another simulator to study caching in opportunistic networks. The mobility model used was SPMBM with 100 nodes roaming on the downtown Helsinki map for 30 days.

Routing: J. Karvo and J. Ott use ONE simulations in conjunction with theoretical considerations to study the impact of *timescales* in the parametrization of estimation based routing algorithms [15]. The study implements two modified versions of existing ONE routing protocols, PRoPHET and MaxPROP, which allows parametrization by predefined values or by local estimation.

The simulations use the WDM scenario described above, but scale down the number of nodes to 544. The nodes generate messages with sizes uniformly distributed between 1 KB and 1 MB and buffer sizes of 100 MB. Simulations are run with different offered loads, ranging from each node generating one message per day to eight messages per day. The simulation time is one day (plus warmup and cool-down periods), which captures one full cycle of the daily WDM movement.

The simulation results show that for routing algorithms that update and maintain historical delivery probability estimations, proper parametrization based on the timescales inherent in the underlying mobility and connectivity is important.

A. Keränen and J. Ott studied routing in a network composed from commercial aircraft traveling between international airports [18]. The simulations use data for up to 248,469 flights between 3,879 airports. Since the airline schedules are well known in advance a shortest-path algorithm can be used to route messages within the network. The results show that 90% of messages sent from the Helsinki airport towards other international airports will reach their destination within 24 hours, while practically all messages will reach their destination within 40 hours. Furthermore, the study also considers the delivery of messages between the airport and the city center using the WDM and concludes that over 99% of messages could be delivered from the city to the airport, while some 67% of messages could be delivered from the airport to the city.

Application Modeling: As explained in Section III-D, applications in DTN networks may require more complex interactions than the simple send/receive interactions modeled by the basic message generators in the simulator. One example of such an application is a content search service, which has been studied through ONE simulations [24].

The simulator is modified by adding application layer logic for evaluating search queries and match them against content items stored in the node. Content items are then distributed to the nodes according to the Zipf distribution. The nodes generate queries which are spread in the network and responses are generated by nodes that have content items matching the query in their content stores. Multiple criteria for terminating the spread of the search query are simulated.

The simulations include four distinct scenarios based on a static disc topology with 92 nodes, RWP with 125 nodes, SPMBM with 126 nodes on a Helsinki map, and WDM with 544 nodes on a Helsinki map.

The simulations show that regardless of the search termination mechanism or scenario employed, only popular content items a few hops away from the query originator are likely to be found. Furthermore, the routing algorithm has a significant role in determining the spread of the query message.

Other DTN Mechanisms: Fragmentation of DTN messages has been studied using ONE simulations [25]. The study includes custom modifications to the ONE message delivery mechanism which allow multiple fragmentation methods such as proactive fragmentation, reactive fragmentation along arbitrary boundaries and reactive fragmentation along predefined boundaries.

The simulations use SPMBM on a Helsinki map and RWP movement in 1 by 1 km area as a comparison. 80 mobile nodes, 40 cars and 6 trams are simulated. The nodes have 2 Mbit/s links and 100 MB buffers. Nodes send messages of equal size (500 KB to 5 MB) every 30 seconds on average. The interval is adjusted so that the offered load remains the same regardless of the message size. All six routing protocols described in Section III-C are used.

The results show that reactive fragmentation consistently increases delivery ratio for large messages while the delay stays the same. Fragmentation along predefined boundaries performs slightly better than fragmentation along arbitrary boundaries; most likely due to better duplicate detection and the absence of trivially small fragments. Proactive fragmentation performs worse than no fragmentation at all in all scenarios.

B. Performance Observations

The ONE simulator offers a framework for evaluating DTN protocols and, as such, its performance primarily depends on the evaluated protocols and their computational and memory requirements. Naturally, the performance depends on the size of the simulation area, the number of nodes, their communication range, the mobility model, and the scanning intervals which together govern the frequency of connection events.

Simulations usually run much faster than real-time, but complex simulation setups and large state space may cause significant slowdown. The simulator continuously reports the ratio of simulation time per second of real-time elapsed, which gives some performance indication.

In our previous work [19] we used three rack-mounted Linux PCs with multi-core Intel x86 CPUs (2.9–3.7 GHz) and 8–128 GB of RAM (but most simulations also run on commodity PCs or laptops). For a 1029 node scenario, we observed mean simulation speeds ranging from 40:1 to well beyond 1000:1 depending on the PC, the mobility model, and the routing protocols; only MaxProp was notably slower (as low as 10:1 and less). Increasing the scanning intervals impacts the number of encounters; using 60 s reduced the simulation time by up to one fourth in HCS and WDM. Finally, increasing the radio range leads to more encounters and thus generates more events to process, slowing down simulations depending on the scenario (we observed a factor of 5–10 when moving from 10 m to 100 m radio range).

The routing protocols influence performance by the number of message copies they create and thus FC, DD, and SnW run faster than Epidemic and PRoPHET. Sophisticated routing protocols, such as MaxProp, that require a lot of state information per node and connection and perform complex processing, may slow down simulations with a large number of nodes and frequent encounters, also requiring a lot of RAM for the Java Virtual Machine (JVM).

The simulation speed also depends on the simulation time resolution, i.e., the intervals at which the simulation time is advanced. This interval is adjustable and doubling the interval may often make the simulation run almost two times faster. We ran the above simulations with a time resolution of 1.0 s, noting that earlier experiments yielded similar simulation results for 1.0 s and 0.1 s [17]. For scenarios with larger radio ranges and/or slowly moving nodes, even coarser granularity may be sufficient.

To further test how the number of simulated hosts effects the simulation speed, we ran a set of simulations with SPMBM movement model using 10×8km Helsinki downtown area map, First Contact router, and the population divided evenly into two groups: pedestrians and cars. Simulation time resolution was 0.1 seconds and message generation interval 20–30 minutes (uniform random distribution). We used a regular desktop PC with AMD x86 CPU running at 2.81 GHz, 4 GB of RAM, and a 32-bit Microsoft Windows operating system.

The simulation time was set to only 3 hours but we observed that longer simulation times resulted in similar results with the FC router. The total number of nodes was varied from 100 to 3000 and the effect of node count to simulation speed was measured to be roughly logarithmic, as shown in Figure 5. The simulation with 100 nodes run at almost $500\frac{1}{s}$, while a 500 node simulation had average speed of $70\frac{1}{s}$. Increasing the node count to 1000 and 1500 decreased the speed respectively to 32 and 20 $\frac{1}{s}$. The 3000 node simulation was still running at approximately $6\frac{1}{s}$.

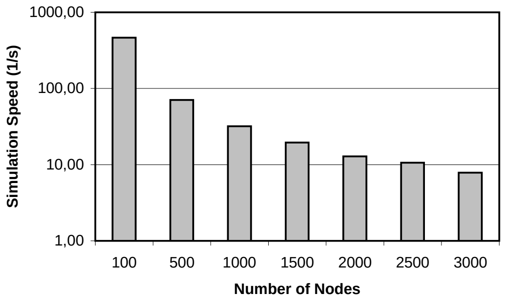

Fig. 5. Simulation speed with different number of nodes

Most of the computational complexity of these simulations came from finding out if two nodes are within radio communication range from each other and moving the nodes in the simulation area. For example, in the 500 node simulation, according to the NetBeans 6.5 Java IDE's profiler[12], about 40% of the CPU cycles were spent for checking the connectivity, 15% for updating routing modules and 25% for movement simulation (mostly for calculating the shortest paths). Also the JVM version may have significant impact on the simulation speed: for example, the simulation with 500 nodes run 14% faster with Sun JVM version 1.6.0_13-b03 compared to version 1.6.0_01-b06.

The upper limit of reasonably simulated node count, at least with currently available commodity hardware, lies somewhere below half a million nodes. A simulation on our test desktop PC with 500,000 nodes, using Random Waypoint movement in 20×20km area, First Contact router and without connectivity checking or messages, was able to process around one update round every second, resulting at speed of $1\frac{1}{s}$ with 1.0 second time resolution. With connectivity checking enabled, same speed is reached with around 200,000 nodes. Simulations of this size require over a gigabyte of JVM heap, and up to 1.5 GB of RAM in total, so 32-bit operating system's memory addressing capabilities, especially with Windows[13], start to limit simulation scenario sizes. We also tested a 1 million node simulation on a 64-bit Linux platform with 3.7 GHz Intel x86 CPU and 16 GB RAM. This required over 3 GB of RAM for the JVM and resulted in simulation speeds just below $0.1\frac{1}{s}$.

If the node count is kept constant and we change only the used routing protocol, the simulation speed will give an indication on the relative computational load the different routing modules cause. We tested this with 200 nodes and the SPMBM movement model. The simulation time was increased to 18 hours since, especially for

[12]http://www.netbeans.org/features/java/profiler.html
[13]http://msdn.microsoft.com/en-us/library/aa366778.aspx

some routing modules, the amount of required computing increases when more messages are created during the simulation. The results of this test are shown in Figure 6.

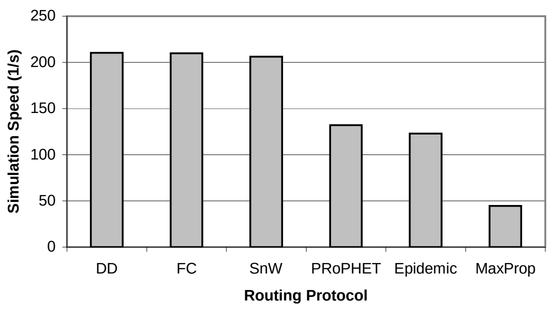

Fig. 6. Simulation speed with different routing protocols

The SnW (binary mode with 6 copies), FC and DD protocols are all approximately equally efficient, while PRoPHET and Epidemic make the simulation take roughly two times longer and MaxProp up to five times longer. PRoPHET and Epidemic simulations become slower over time since they generate a large number of message replicates. PRoPHET requires more advanced operations for to the probability calculations and message sorting, but it compensates this by a smaller amount (some 25% in this test) of started transfers and created message replicas. MaxProp is clearly the heaviest of the routing protocols and its speed also degrades most over time. MaxProp's shortest path calculation for messages took some 20% of the CPU cycles but even more (over 50%) was spent on exchanging the delivery predictions of different nodes. The amount of state MaxProp keeps, especially the delivery prediction mappings, grow over time when more nodes meet each other. Eventually all nodes have full knowledge of each others' prediction mappings requiring in the order of the number of nodes squared entries on each node.

Finally, we tested how much impact different movement models have on the simulation speed. Figure 7 shows that there is not much difference between RWP and MBM, but SPMBM and WDM make the simulation around 20% slower because of the shortest path calculations. WDM, even though notably more complex model than SPMBM, is slightly lighter to simulate since in WDM nodes are stationary for long periods of time and do not thus require complex calculations that often.

All in all, the present version of the ONE simulator is capable of supporting sizable simulation setups of some thousand nodes with fairly complex movement models and routing modules and still simulate their behavior faster than real time. Even larger simulation scenarios are possible if the simulation time resolution can be decreased and/or only simpler routing modules with small amount of state information are used.

VI. Conclusion

In this paper, we have presented the ONE simulator, an opportunistic networking evaluation system that offers a variety of tools to create complex mobility scenarios that come closer to reality than many other synthetic mobility models. GPS map data provides the scenario setting and node groups with numerous different parameters are used to model a wide variety of independent node activities and capabilities. The Working Day Movement model allows recreating complex social structures and features such as scanning intervals add further aspects of reality and heterogeneity to the modeling. All these aspects may matter as our simple examples have shown. With its flexible input and output interfaces, the ONE simulator can incorporate real-world traces and feeds from other mobility generators as well as generate mobility traces for use by other simulators. Its DTN framework currently includes six parameterizable DTN routing protocols and two types of application messaging. Its visualization component is used for instant sanity checks, deeper inspection, or simply to observe node movements in real-time—which broadens its applicability beyond DTN studies. Particularly the integration with the DTN reference implementation allows creating testbeds and emulations.

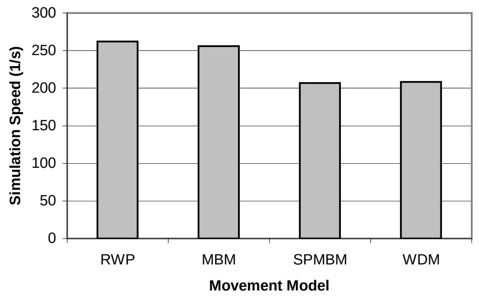

Fig. 7. Simulation speed with different movement models

The ONE simulator still has numerous limitations and will hardly ever be complete. In the short-term we intend to examine support for broadcast and multicast addressing so that the same message can be addressed and delivered to multiple nodes. We also intend to increase support for importing location data from external sources in standard formats. The dependency on DTN2 for supporting real-life DTN integration will be removed through a native implementation of the relevant protocols. Furthermore, message generation activity will need to take into account group relationships and context information. Finally, further refinements to the Working Day Mobility model are needed to provide better modeling of buildings and of traffic; we are interested in learning and importing from other simulation environments.

We are using and continuously advancing the ONE simulator in our ongoing DTN research, as a mobility generator and as a full simulator, e.g., for our work on opportunistic content caching in DTNs [26] and we apply it in our DTN graduate course.[14] We maintain an open source distribution of the simulator; the current release (1.4.0) includes all features described in this paper [32].

[14]TKK S-38.3151, https://noppa.tkk.fi/noppa/kurssi/s-38.3151/

REFERENCES

[1] BETTSTETTER, C. Smooth is better than sharp: A random mobility model for simulation of wireless networks. In *Proc. of ACM MSWiM* (July 2001).

[2] BOUDEC, J.-Y. L., AND VOJNOVIC, M. Perfect Simulation and Stationarity of a Class of Mobility Models. In *Proc. of IEEE Infocom* (2005).

[3] BURGESS, J., GALLAGHER, B., JENSEN, D., AND LEVINE, B. N. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *Proceedings of IEEE Infocom* (April 2006).

[4] CAMP, T., BOLENG, J., AND DAVIES, V. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications 2*, 5 (2002), 483–502.

[5] CERF, V., BURLEIGH, S., HOOKE, A., TORGERSON, L., DURST, R., SCOTT, K., FALL, K., AND H.WEISS. Delay-Tolerant Network Architecture. RFC 4838, 2007.

[6] CHOFFNES, D. R., AND BUSTAMANTE, F. E. An integrated mobility and traffic model for vehicular wireless networks. In *Proc. of the 2nd ACM International Workshop on Vehicular Adhoc Networks* (2005).

[7] DANIEL GRGEN, H. F., AND HIEDELS, C. JANE – The Java Ad Hoc Network Development Environment. In *Proc. of the 40th Annual Simulation Symposium (ANSS)* (2007).

[8] EAGLE, N., AND PENTLAND, A. S. Reality mining: sensing complex social systems. *Personal Ubiquitous Computing 10*, 4 (2006), 255–268.

[9] EKMAN, F., KERÄNEN, A., KARVO, J., AND OTT, J. Working Day Movement Model. In *Proc. 1st ACM/SIGMOBILE Workshop on Mobility Models for Networking Research* (May 2008).

[10] FALL, K. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proc. of ACM SIGCOMM* (2003).

[11] HUI, P., CHAINTREAU, A., SCOTT, J., GASS, R., CROWCROFT, J., AND DIOT, C. Pocket Switched Networks and Human Mobility in Conference Environments. In *Prof. of the ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)* (2005).

[12] HYYRYLÄINEN, T., KÄRKKINEN, T., LUO, C., JASPERTAS, V., KARVO, J., AND OTT, J. Opportunistic Email Distribution and Access in Challenged Heterogeneous Environments (demo. In *Proc. of the ACM MobiCom Workshop on Challenged Networks (CHANTS)* (2007).

[13] JAIN, S., FALL, K., AND PATRA, R. Routing in a Delay Tolerant Network. In *Proc. of ACM SIGCOMM* (2004).

[14] JARDOSH, A. P., BELDING-ROYER, E. M., ALMEROTH, K. C., AND SURI, S. Real-world environment models for mobile network evaluation. *IEEE Journal on Selected Areas in Communications 23*, 3 (March 2005), 99–105.

[15] KARVO, J., AND OTT, J. Time scales and delay-tolerant routing protocols. In *CHANTS '08: Proceedings of the third ACM workshop on Challenged networks* (2008), ACM, pp. 33–40.

[16] KERÄNEN, A. Opportunistic Network Environment Simulator. Special Assignment report, Helsinki University of Technology, Department of Communications and Networking, May 2008.

[17] KERÄNEN, A., AND OTT, J. Increasing Reality for DTN Protocol Simulations. Tech. rep., Helsinki University of Technology, Networking Laboratory, July 2007.

[18] KERÄNEN, A., AND OTT, J. DTN over Aerial Carriers. In *ACM MobiCom 2009 Workshop on Challenged Networks* (Beijing, P.R. China, 9 2009).

[19] KERÄNEN, A., OTT, J., AND KÄRKKÄINEN, T. The ONE simulator for DTN protocol evaluation. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques* (2009), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–10.

[20] LEGUAY, J., FRIEDMAN, T., AND CONAN, V. Evaluating Mobility Pattern Space Routing for DTNs. In *Proc. of IEEE Infocom* (2006).

[21] LINDGREN, A., DORIA, A., AND SCHELEN, O. Probabilistic routing in intermittently connected networks. In *The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR)* (2004).

[22] MARFIA, G., PAU, G., SENA, E. D., GIORDANO, E., AND GERLA, M. Evaluating vehicle network strategies for downtown Portland: opportunistic infrastructure and the importance of realistic mobility models. In *Proc. of the 1st International MobiSys Workshop on Mobile Opportunistic Networking* (2007).

[23] PELUSI, L., PASSARELLA, A., AND CONTI, M. Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks. *IEEE Comm. Magazine* (Nov 2006).

[24] PITKÄNEN, M., KÄRKKÄINEN, T., GREIFENBERG, J., AND OTT, J. Searching for Content in Mobile DTNs. *Pervasive Computing and Communications, IEEE International Conference on 0* (2009), 1–10.

[25] PITKÄNEN, M., KERÄNEN, A., AND OTT, J. Message fragmentation in opportunistic DTNs. In *WOWMOM '08: Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 1–7.

[26] PITKÄNEN, M., AND OTT, J. Enabling Opportunistic Storage for Mobile DTNs. *Journal on Pervasive and Mobile Computing 4*, 5 (Oct 2008), 579–594.

[27] SCOTT, J., HUI, P., CROWCROFT, J., AND DIOT, C. Haggle: A Networking Architecture Designed Around Mobile Users. In *Proceedings of IFIP WONS* (2006).

[28] SCOTT, K., AND BURLEIGH, S. Bundle Protocol Specification. RFC 5050, November 2007.

[29] SPYROPOULOS, T., PSOUNIS, K., AND RAGHAVENDRA, C. Efficient Routing in Intermittently Connected Mobile Networks: The Multiple-copy Case. *ACM/IEEE Transactions on Networking* (Feb. 2008).

[30] SPYROPOULOS, T., PSOUNIS, K., AND RAGHAVENDRA, C. S. Single-copy routing in intermittently connected mobile networks. In *Proc. Sensor and Ad Hoc Communications and Networks SECON* (October 2004), pp. 235–244.

[31] SPYROPOULOS, T., PSOUNIS, K., AND RAGHAVENDRA, C. S. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *Proc. of the ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)* (2005).

[32] TKK/COMNET. Project page of the ONE simulator. http://www.netlab.tkk.fi/tutkimus/dtn/theone, 2009.

[33] VAHDAT, A., AND BECKER, D. Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.

[34] WANG, W., SRINIVASAN, V., AND MOTANI, M. Adaptive contact probing mechanisms for delay tolerant applications. In *Proc. of ACM MobiCom* (September 2007).

[35] ZHANG, Z. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys and Tutorials 8*, 4 (January 2006), 24–37.

Ari Keränen received his M.Sc. in communications engineering from Helsinki University of Technology (TKK), Finland, in 2008 with major in networking technology and minor in software systems. He has worked as a Researcher at the Department of Communications and Networking at TKK since January 2007 and at Ericsson Research Finland since June 2007. His research interests include Delay-tolerant Networking, Peer-to-Peer networks, and Host Identity Protocol.

Teemu Kärkkäinen studies communication engineering at the Aalto University School of Science and Technology. He has worked as a Researcher at the Department of Communications and Networking since 2006, focusing on applications and services for Delay-tolerant Networking.

Jörg Ott is Professor for Networking Technologies at the Department of Communications and Networking at the Aalto University School of Science and Technology. He received his Diploma in Computer Science in 1991 and his Doctor in Engineering (Dr.-Ing.) in 1997 from TU Berlin and also holds a Diploma in Industrial Engineering from TFH Berlin (1995). His research interests are in Internet technologies, transport and application protocols for reliable as well as real-time communications, protocol design, and future network architectures, with one focus on Delay-tolerant Networking. In this context, he is member of the steering committee of the ACM Workshops for Challenged Networks (CHANTS) and Mobility in the Evolving Internet Architecture (MobiArch), which he co-chaired in 2007 and 2009, respectively.