Filling in the missing link between simulation and application in opportunistic networking

Adrián Sánchez-Carmona, Frédéric Guidec, Pascale Launay, Yves Mahéo, Sergi Robles

Filling in the Missing Link between Simulation and Application in Opportunistic Networking

Adrián Sánchez-Carmona[1], Frédéric Guidec[2], Pascale Launay[2], Yves Mahéo[2], Sergi Robles[1]

1: Universitat Autònoma de Barcelona (UAB),
2: IRISA/Université Bretagne Sud

***Abstract*—In the domain of Opportunistic Networking, just like in any other domain of computer science, the engineering process should span all stages between an original idea and the validation of its implementation in real conditions. Yet most researchers often stop halfway along this process: they rely on simulation to validate the protocols and distributed applications they design, and neglect to go further. Their algorithms are thus only rarely implemented for real, and when they are, the validation of the resulting code is usually performed at a very small scale. Therefore, the results obtained are hardly repeatable or comparable with others.**

LEPTON is an emulation platform that can help bridge the gap between pure simulation and fully operational implementation, thus allowing developers to observe how the software they develop (instead of a pseudo-code that simulates its behavior) performs in controlled, repeatable conditions.

In this paper we present LEPTON, and we show how existing opportunistic networking systems can be adapted to run with this emulator. Taking two existing middleware systems as use cases, we also demonstrate that running demanding scenarios with LEPTON constitute an excellent stress test and a powerful tool to improve the opportunistic systems under test.

I. Introduction

Opportunistic networks constitute a category of mobile ad hoc networks in which the sparse or irregular distribution of mobile devices (or nodes) yield frequent link disruptions and network partitions [1]. In such conditions, the store, carry and forward principle of Delay Tolerant Networking (DTN [2], [3]) helps bridge the gap between non-connected parts of the network. Whenever a transient contact occurs between two nodes, this contact can be exploited opportunistically by these nodes to exchange messages. The messages received by a node during such a contact are stored in a local cache, so they can be carried physically as the node is moving, and forwarded later to other mobile nodes.

Developing middleware and applications for opportunistic networks is a challenge, because message delivery is often not guaranteed, and because this delivery can be delayed by minutes, hours, or days, as it depends on the wanderings of benevolent mobile carriers. In order to meet this challenge developers must follow a rigorous procedure, that ideally should involve all the steps shown in Figure 1.

First comes the initial idea, the conception of the mechanisms. After this initial phase, the idea must be reified into a particular model, which can then be analyzed formally. During this analysis, the mechanisms and procedures can be checked, some theoretical results can be obtained, and limitations can be identified. The next stage is typically simulation. In a simulator, the system can be tested in a given set of scenarios. Even if these scenarios involve datasets coming from the real world (e.g., traces of real taxis, or positions of real people evacuating a stadium), or even if the simulator is assumed to simulate very accurately all the layers of the protocol stack, the system under evaluation is executed as some sort of pseudo-code, and this gives little proof that the system being designed can eventually be deployed and used for real. Results obtained through simulation may be deceptive, creating a misleading impression of scientific correctness. Indeed, as observed in [4], the credibility of simulation results tends to decrease as the use of simulation increases. The final validation of an opportunistic networking system should thus always be based on real full-featured code (accounting for example for memory management or concurrency issues), rather than on the pseudo-code used in simulations.

Testing real code in real conditions can be painstaking, or even impossible, especially when these real conditions involve mobility [5] and hundreds of nodes or thousands of hours. Emulation is an approach that can help with this respect, as it makes it possible to run real code in tightly controlled (and repeatable) conditions. The emulation stage can be seen as the missing link in the engineering process of most existing opportunistic networking systems. Yet, this stage is crucial to make sure that the code under test —with its bugs and limitations— is scalable, and can correctly integrate and interact with the rest of the components of the system (such as users, for instance). Using an emulator, these properties can be verified under the desired conditions, and results can be reproduced and compared [6] to others at convenience.

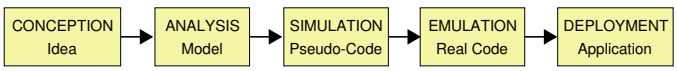

Figure 1. Engineering process of an opportunistic networking system.

Briefly put, we could say that emulation is more apt than simulation to evaluate real code, and easier and cheaper than experimentation. Therefore, we propose to complement (not to substitute) simulation and experimentation with emulation, hence putting the emphasis on an engineering step in the development of an opportunistic system that could reveal itself especially useful as the complexity of this system grows.

In this paper we present LEPTON (Lightweight Emulation PlatTform for Opportunistic Networking), an emulation platform that has been primarily designed to allow the developers of real opportunistic networking software (i.e., middleware and/or applications) to run their real software systems with simulated mobility. With LEPTON, an implementation can run in real time, either on a real device (e.g. smartphone, tablet, laptop) or on a virtual one.

LEPTON also constitutes an interesting demonstration tool, since participants in a demo session can use an opportunistic application deployed on smartphones or tablets, while a display screen shows the simulated mobility of all devices.

In this paper our main contribution is LEPTON, a lightweight emulation platform for opportunistic networking. In order to show its usefulness, we also use it to compare two existing DTN middleware systems using the same scenario and mobility traces.

The remainder of this paper is organized as follows. Related work is discussed in Section II. In Section III we provide an overview of LEPTON and of its salient features. In Section IV we present briefly the two DTN systems we considered as use-cases for our proof of concept, and we explain how each system was adapted to be used with LEPTON. Experimental results are presented in Section V, and Section VI concludes this paper.

II. Related work

Simulation is the lightest approach to observe how opportunistic networking protocols or applications can perform at runtime. General-purpose discrete event network simulators such as ns-2 [7], ns-3 [8], OMNeT++ [9], QualNet [10] or Riverbed Modeler [11] include modules that can simulate the mobility of nodes in a wireless network. Most of these simulators implement standard wireless MAC layers (e.g., IEEE 802.11, 802.15.1, 802.15.4), and they can optionally simulate physical phenomena observed on the wireless medium, such as shadowing, free space path loss, fading, co-channel interference, etc.

Because of its ease of use, the ONE simulator [12], [13] has become the tool of choice for simulating opportunistic networks. It supports a variety of mobility models, and contact or mobility traces such as those available in the CRAWDAD database [14] can be imported with little effort. Unlike many other simulators, ONE does not attempt to simulate the PHY and MAC protocol layers accurately. Communication is message-based, rather than packet-based or frame-based. A message is transferred between two nodes if these nodes are considered as neighbors at the time the message is sent. Optionally, the delivery of a message can be delayed so as to account for a set transmission bitrate. A commonly praised feature of ONE is that several of the major DTN routing protocols (e.g., First Contact, Epidemic dissemination, Spray and Wait, MaxProp, Prophet) have already been implemented for this simulator, so they are immediately available for running simulations. Yet these protocols are implemented in such a way that no control messages are ever exchanged between nodes during a simulation run. Comparing the results obtained with these implementations therefore makes little sense, as the overhead induced by control traffic is simply ignored.

A simulator is indeed a convenient tool for the developer of a new protocol or distributed algorithm, as simulation makes it possible to observe how this protocol or algorithm performs in a virtual setup in which everything is fully repeatable and controllable. This setup can include hundreds or thousands of nodes, which would hardly be practical in real settings. Yet the validity of results obtained with a simulator is always debatable, for every single part of a simulated system can be deemed as being *not realistic enough*. For example, the mobility models used in simulators can hardly reproduce the diversity of real mobility patterns. Usually, this is mitigated by the usage of contact and mobility traces instead of pure algorithmic models, but these traces often been captured in very specific settings (e.g., people moving around in a conference building, taxi cabs roaming city streets, etc.). Radio channel modeling is another example of this, as it cannot mimic all the complexity of real wireless medium characteristics, such as radio wave reflection on obstacles (e.g., walls, furniture, etc.) or interferences due to neighboring electronic devices. However, the main drawback is that the protocols or distributed algorithms tested in simulators are often coded as pseudo-code (it is not possible to execute the real code in a discrete event driven simulation), and are thus significantly simpler than the real code that could be deployed on real mobile devices. Indeed, when using discrete event simulators, the time required to react to an event is neglected, for event processing is performed atomically. When developing code for real execution, though, attention must be paid to ensuring that concurrent events can be processed as smoothly and efficiently as possible.

Since simulation results can only provide an indication of how a system *should* perform in real life, testing this system in real conditions is the ultimate way to confirm that it indeed performs as expected. Yet running experiments in real conditions requires deploying testbeds, possibly at a large scale. While everything is virtual in a simulation, everything is –or at least should be– real in a testbed. Indeed, a testbed is simply a perfectly normal instance of the system that is under study in a particular experiment [15]. Running experiments in a testbed offers the greatest degree of realism (since everything is running "for real"), but deploying and managing hardware and software in a testbed is a costly and time-consuming endeavor. To the best of our knowledge no large testbed has ever been deployed specifically for opportunistic networking, besides DieselNet, which itself was part of the DOME testbed [16]. Some general-purpose large-scale network testbeds such as ORBIT [17] can support mobile nodes, though.

Simulation and testbeds lie on opposite ends of the experimentation spectrum. Simulation allows repeatability, tight control, large scale, and cost-effective tests. But because of the high level of abstraction it offers, most results it produces should only be considered as qualitative assessments [18]. In contrast, the drawbacks of experiments conducted in testbeds are the lack of repeatability and tight control, as well as limited scalability.

Emulation fits between simulation and testbeds, as it in-

volves real elements used along with simulators. Several emulation systems for mobile ad hoc networks have been proposed during the last decade. As a general rule, the mobility of network nodes and transmissions on the wireless medium are simulated, and network nodes are either physical nodes ([19], [20]) or virtual nodes ([21], [22]), or a mix of both kinds of nodes.

Using node virtualization makes it possible to run experiments involving large populations of nodes. For this reason, virtualization is also used in many testbeds (which thus cease to be *pure testbeds*) as a means to provide scalability.

With node virtualization, the code that should normally be executed on a pool of real distinct mobile devices is instead executed in virtual (fixed) machines. The TUNIE emulation tested [21] thus uses a XEN virtual machine hypervisor to coordinate virtualized DTN nodes, whereas EmuStack [22] uses the Docker container technology, MoViT [23] uses KVM (Kernel-based Virtual Machine), and HYDRA [24] uses VirtualBox.

Whether using real or virtual nodes, the code running on each node includes application code, possibly some middleware implementing high-level protocols and services, and the OS with its entire protocol stack. Instead of sending and receiving frames directly through a wireless interface, though, a virtual interface is created (using for example the TUN/TAP driver) in order to intercept traffic at either packet or frame level. This traffic is then tunneled to a centralized controller whose role is to simulate the wireless medium. This controller drives the communication between all network nodes according to their simulated mobility. In EmuStack, standard Unix utilities such as Netfilter *(iptables)* and Traffic Control *(tc)* are used to control and shape the traffic between pairs of nodes, thus simulating network connectivity and radio channel conditions. In MoViT and HYDRA, traffic shaping is not performed by a centralized controller, but each virtual node processes its own outgoing traffic locally, based on directives received from a coordination system [23]. In TUNIE, traffic flows through OpenFlow-enabled Ethernet switches and wireless access points, that delegate forwarding decisions to a remote controller [21], [24]. In TWINE [25], which is focused on studying cross-layer techniques, an emulation layer is inserted directly in the Linux kernel, between the network layer and the device driver. This additional layer includes several components that simulate for example mobility and wireless propagation. In [26], a static-grid testbed is used, and it is application code that moves from node to node in the grid in order to simulate the mobility of devices.

In TROWA [20], APIs are provided at application, transport and network layers. Using these APIs, applications or protocols running on real or virtual devices can send all traffic to a mobile network simulator, this traffic being tunneled via TCP. HINT [27] uses a quite similar approach, as it defines a User Link Layer (ULL) that abstracts communication between applications (running on real devices) and a centralized mobile network simulator. In both cases, the code that must be tested must be modified, since instead of using standard communication APIs (such as TCP or UDP sockets) it must use those provided with the emulation system.

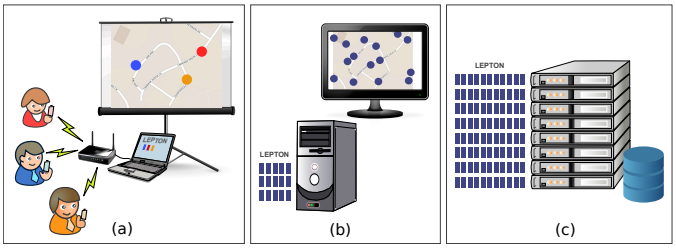

Figure 2. LEPTON's main deployment modes: *a)* Demonstration session using real devices; *b)* Small scale experiment on a single workstation; and *c)* Large scale experiment on a cluster.

All the above-mentioned emulation systems, except TROWA and HINT (that require to use their specific communication API), require either specific hardware (e.g., OpenFlow-enabled devices), a specific software environment (e.g., virtual machines or containers, or a modified OS kernel), and sometimes a combination of specific hardware and software. In fact most of these systems are actually testbeds extended with node and link virtualization. For the developer of an opportunistic networking protocol or algorithm, accessing one of these testbeds or deploying an equivalent platform for testing his/her code is hardly an option, at least not for early experimentations while the code is still under development. A lighter solution is needed, even if using this solution implies trading off a bit of transparency and accuracy, against availability and ease of use.

The LEPTON emulation platform is meant to provide this solution. It is a lightweight, easily deployable emulation platform that does not require exotic networking equipment, and that does not even require deploying virtual machines on one or several hosts. Besides, LEPTON is openly distributed and documented, so it is readily available for any interested user. A simple laptop or desktop workstation running Linux can easily support emulation-based experiments involving up to a couple hundred nodes, and larger experiments can be run on any cluster of Linux machines.

III. The LEPTON emulation platform

LEPTON is implemented as Java code, and it is distributed under the terms of the GNU General Public License. Its prime function is to simulate the mobility of nodes in an opportunistic network, while conducting transmissions between these nodes based on their relative positions. Being an emulator rather than a simulator, LEPTON drives the communication between full-featured instances of an opportunistic system, each instance determining the behavior of one node during the simulation. In the remainder of this paper we will use the term System Node (SN) to refer to an instance of the opportunistic system. SNs are not a part of LEPTON, but the system under evaluation. A SN is typically composed of application code combined with communication middleware, which itself requires to run on top of a standard protocol stack. This SN can be executed on real devices, such as smartphones, tablets, or a wireless sensors (see Fig. 2.a). The ability to run experiments involving real devices makes possible to confirm that the software being tested can indeed run smoothly on

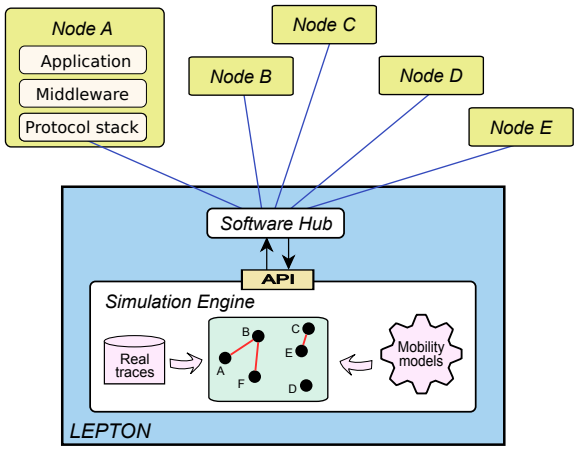

Figure 3. Multiple nodes interacting with LEPTON. Nodes interact between them through the software hub, which interacts with the simulation engine that models the nodes' mobility.

```
...
st 8500                    // Time step (ms)
an N1 x=87.90 y=50.20      // Add N1
an N2 x=87.80 y=51.60      // Add node N2
ae E1-2 N1 N2              // Add edge N1-N2
...
st 13200                   // Time step (ms)
cn N1 x=76.00 y=50.20      // Change node (N1 has moved)
cn N2 x=69.80 y=50.40      // Change node (N2 has moved)
...
st 27600                   // Time step (ms)
cn N1 x=66.40 y=47.20      // Change node (N1 has moved)
cn N2 x=64.20 y=37.40      // Change node (N2 has moved)
de E1-2                    // Delete edge N1-N2
...
st 120600                  // Time step (ms)
dn N1                      // Delete node N1
...
```

Figure 4. Illustration of the DGS file format.

such devices. It also comes in handy in demo sessions, since attendees can then use real devices that implement the system that is being presented, while observing on a display screen what would be the effects of mobility... if they were actually moving.

When an experiment must involve a large number of SNs, running the system under test on real devices is often impractible. Multiple SNs can then be executed concurrently on a single desktop workstation (Fig. 2.b), or on the nodes of a computer cluster (Fig. 2.c). The important point here is that the code implementing each SN is not pseudo-code, but full-featured code that would run equally well on a real mobile device.

In any case, the role of LEPTON is to simulate the mobility of the SNs, and to control the communication between them accordingly.

Since interacting with real instances of an opportunistic system cannot be done in a discrete-event simulator, LEPTON has been designed to emulate the mobile network in real time, moving nodes either according to a given mobility policy, or based on mobility or contact traces.

A. General architecture

Every LEPTON's experiment involves two parts (see Fig. 3): 1) the network emulator and 2) the System Nodes (SNs), although the SNs are not part of LEPTON, they are implementations of the system to be tested (more details in Section IV). The emulator itself is structured in two main parts: a simulation engine and a software hub. The simulation engine determines if any pair of nodes should be considered as neighbors at any specific time of the experiment. The software hub plays approximately the same role as a switched hub in an Ethernet LAN: it relays every message it receives from a SN to its destination, and to that destination only. Yet in order to decide whether this destination is indeed accessible, the hub must consult the simulation engine.

B. The simulation engine

The role of the simulation engine is to model the wireless network internally as a dynamic graph, whose nodes represent SNs, and whose edges represent radio links between SNs. The implementation of this dynamic graph in the simulation engine relies on facilities offered in the GraphStream library [28], which provides elaborate data types and algorithms for dynamic graph modeling, analysis, and visualization. The simulation engine is also responsible for driving the dynamics of the graph according to a given mobility model, or based on mobility and/or contact traces.

Several traditional mobility models (Random Waypoint, Levy Walk, Map-based mobility, etc.) have already been implemented for LEPTON, and new models can be included as and when needed. A manual mobility model is available, that is especially useful in demo sessions. With this model nodes do not move spontaneously, but the dynamic graph that represents the connectivity between nodes can be adjusted manually by dragging and dropping nodes on a display screen.

LEPTON can also run experiments based on pre-defined mobility or contact scenarios. In that case the simulation engine simply reads a DGS file, whose format is described in the next section.

The simulation engine can calculate radio contacts at runtime (i.e., while running an experiment), alternatively, these contacts can be calculated in advance and be inserted in the DGS file that will be played during an experiment. LEPTON includes a very simple radio propagation model that can be used both ways: two mobile nodes are assumed to be in radio contact if they are within a fixed distance of each other. Besides, the calculation of radio contacts is controlled via a user-defined function that allows any interested users to easily implement more accurate models (for example, we have developed one that accounts the obstacles).

A file format for dynamic graphs : The GraphStream library defines an event-oriented file format called DGS (Dynamic Graph Stream) for describing dynamic graphs. With this format the evolution of a graph is described using events such as adding, deleting or changing a node or an edge, as illustrated in Figure 4. Reading a DGS file therefore comes down to reading a stream of such events.

When an experiment is run according to a given mobility model, rather than by reading a pre-defined DGS file, LEPTON can record all graph events in DGS format, thus recording how nodes have appeared or disappeared during the experiment, how they have moved, and how radio contacts have been established and lost between neighbor nodes. The resulting DGS file can then be used to repeat the experiment as and when needed.

C. The software hub

In a real mobile network, neighbor SNs can exchange messages directly with one another. When running an experiment with LEPTON, each message sent by a SN must be directed through the software hub, so, interfacing an existing opportunistic system with LEPTON therefore requires:

1) Making sure that all SNs explicitly redirect their traffic to the software hub, which from their viewpoint will be seen as some kind of a proxy.
2) Developing an appropriate software hub for that system, and making sure that this hub will consult the simulation engine in order to decide what to do about each message received.

The first requirement is usually easy to meet. Instead of sending beacons to a multicast group, and gossiping with peers through unicast communication, each SN must redirect its transmissions explicitly to the software hub. Depending on the opportunistic system considered, this may require changing slightly the code of SNs, or simply changing a parameter in a configuration file.

Developing a dedicated software hub for a system may require a little more work, but since most existing systems rely on UDP and TCP for their transmissions[1], LEPTON comes with Java software hub templates that can be easily adapted to fit the needs of a particular system. These templates are presented briefly below, and illustrated in pseudo-code. LEPTON is compatible with any opportunistic system that uses non-IP transmissions (e.g., Bluetooth, XBee, ZigBee). However, interfacing with it requires developing a specific software hub for the considered technology.

Forwarding UDP traffic: The software hub receives all datagrams (usually, the beacons) produced by SNs. Upon receiving a datagram, the hub learns about the identity, IP address, and port number used by the sender's beaconing service (see lines 1 and 2 in Alg. 1). The information obtained when receiving a datagram from a node is stored in a registry (line 4), so it can later be used to forward messages to that SN. With some opportunistic systems, the datagram may contain an indication of the TCP port number the sender's gossiping service is listening to. In that case this port number must be replaced in the datagram by the port number of the TCP socket the hub is listening to (line 6), so the target SN can later open a gossiping session through the hub rather than directly with a peer SN. These operations may be omitted in systems that do not rely on TCP-based gossiping, or that systematically use the same number for UDP and TCP ports.

Every datagram received from a SN can then be forwarded to all neighbors of the sender, the list of these neighbors being provided by the simulation engine. Note that Alg. 1 allows datagrams to be addressed either in unicast mode to a single neighbor (lines 13-14), or in broadcast (lines 8-9) or multicast (lines 10-12) mode. In all cases, the software hub makes a list of all the SNs that must receive the datagram and sending a copy of it to every one of them (lines 18-21).

Algorithm 1 Processing of UDP datagrams received by the software hub

```
1:  upon receiving beacon from (@ip, udpPort)
2:      extract (id_src, id_tgt, [tcpPort]) header from beacon
3:      // Record (or update) sender's info in registry
4:      peerRegistry.put(id_src, @ip, udpPort, [tcpPort])
5:      if tcpPort != null then
6:          substitute tcpPort by localTcpPort in beacon
7:      // Look for targets
8:      if id_tgt == broadcast then
9:          target = SimulationEngine.getPeers(id_src)
10:     else if id_tgt == multicast then
11:         allPeers = SimulationEngine.getPeers(id_src)
12:         target = filterMulticastPeers(allPeers, id_tgt)
13:     else if SimulationEngine.arePeers(id_src, id_tgt) then
14:         target = {id_tgt}
15:     else
16:         target = {}
17:     // Iteratively send to all targets
18:     for id in target do
19:         if peerRegistry.contains(id) then
20:             (@ip, udpPort) = peerRegistry.get(id)
21:             send beacon to (@ip, udpPort)
```

Forwarding TCP traffic: When the message passing is based on TCP sessions, the situation is a bit more tricky (Alg. 2). Indeed, when a contact is established in "real life" between two SNs, a TCP session is established directly between these nodes, and all subsequent messages are then exchanged via that TCP session. Depending on the characteristics on the DTN system considered, such a TCP session is maintained either as long as possible (i.e., until the contact is disrupted), or only for the duration of the transaction. When running in emulation mode, a SN that wishes to start gossiping with another SN must actually open a TCP session to the software hub, which in turns opens another TCP session to the target SN, provided the simulation engine confirms that the target SN is a neighbor of the initiator. Two TCP sessions must thus be maintained by the hub for each pair of neighbor SNs that are engaged in gossiping, the hub ensuring that everything it receives from one SN is forwarded to its peer (line 18), with the simulation engine's assent (line 16). When one of these SNs decides to close the session opened (through the hub) with its neighbor (line 21), both TCP sessions are closed simultaneously by the hub (line 22). Likewise, if the hub receives something from a SN and learns from the simulation engine that the target peer is not a neighbor of the

[1]Many opportunistic systems implement a neighbor discovery mechanism based on UDP multicast or broadcast, while the exchange of control and data messages between neighbor nodes is based either on UDP or TCP.

sender anymore, then both TCP sessions are closed (line 20). Note that the software hub should never close TCP sessions authoritatively as soon as the contact is lost between two SNs, because a TCP session can be maintained between two hosts even when there is no connectivity between them, as long as no attempt is made to transfer data via that session. Closing TCP sessions authoritatively as soon as the contact is lost between two SNs would introduce a bias, since TCP sockets would be closed earlier during emulation-based experiments than in real life. The initiative to close a TCP session must therefore always be taken by a SN, not by the software hub.

Algorithm 2 "Bridging" between two TCP sessions by the software hub

```
1:  main
2:      upon receiving opening request from (@ip, tcpPort)
3:          iSocket = accept incoming session
4:          extract (id_src, id_tgt) header from input stream
5:          // Check that id_tgt is a neighbor of id_src
6:          if SimulationEngine.arePeers(id_src, id_tgt) then
7:              (@ip, tcpPort) = peerRegistry.get(id_tgt)
8:              oSocket = open session with (@ip, tcpPort)
9:              spawn StreamManager(iSocket, oSocket)
10:             spawn StreamManager(oSocket, iSocket)
11:         else
12:             close incoming session
13: thread StreamManager(iSocket, oSocket)
14:     upon receiving data from iSocket
15:         // Check that id_src and id_tgt are still neighbors
16:         if SimulationEngine.arePeers(id_src, id_tgt) then
17:             (@ip, tcpPort) = peerRegistry.get(id_tgt)
18:             forward data to oSsocket
19:         else
20:             close both sockets
21:     upon receiving close request from iSocket or oSocket
22:         close both sockets
```

Algorithms 1 and 2 describe the general behavior of a software hub that can fit most opportunistic systems, provided these systems rely on UDP for beaconing, and either UDP or TCP for gossiping.

Software hub's considerations: In order to implement a software hub that uses a different communication technology, the only parts of the templates that need to be modified are those that extract meta-information from UDP datagrams (line 2 in Alg. 1), or from the input stream in TCP sessions (line 4 in Alg. 2).

Moreover, since all traffic must flow through the software hub during an experiment, the workload imposed on the node running that hub can be high. It is legitimate to wonder if this node may constitute a bottleneck during the experiments. Fortunately, measurements confirm that both the CPU workload and network throughput observed on that node remain quite low (more details on Section V). Besides, the user can do as in [29] and deploy as many software hubs as needed. This way, the workload would be distributed among all the software hubs, which only need to communicate between them to update the state of their registry (if it is needed).

```
public void addNode(String id);
public void deleteNode(String id);
public boolean arePeers(String id1, String id2);
public Collection<String> getPeers();
```

Figure 5. API that allows the software hub to interact with the simulation engine. The *addNote* and *deleteNode* functions are used to notify the appearance or disappearance of nodes to the simulation engine, while the *arePeers* and *getPeers* functions are used by the software hub to make forwarding decisions.

The simulation engine API: The API used by the software hub to interface with the simulation engine is quite simple (Fig 5). With these functions, the hub can notify the simulation engine that it has detected the presence or the loss of a SN. Upon receiving a message from a SN, the software hub can also consult the simulation engine in order to determine what to do about this message. The software hub can thus either check that the message's destination node is currently a neighbor of the source node, or it can obtain the list of all current neighbors of the sender.

Considerations about the addressing scheme: When developing a software hub for an opportunistic system, special attention should be paid to the addressing scheme used in this system. Indeed, redirecting all messages to the hub implies that identifiers for the source and destination of a message should be included in the message itself. An IP address, for example, cannot be used to identify the actual source or destination of a message, since this message will first be sent to, and then received from, the software hub. Fortunately most opportunistic systems use a high-level addressing scheme, based for example on textual node ids, which provides the required decoupling between "network-level" addresses (such as IP addresses) and "system-level" addresses (such as node ids). LEPTON can thus relate the messages received by the hub to the mobile nodes whose whereabouts are determined by the simulation engine.

D. Modes of operation

LEPTON supports several modes of operation that can be applied to the two following types of experiments:

- *Experiment involving real devices:* when real devices (and thus, possibly, real users) are involved during an experiment (see Fig. 2.a), LEPTON must discover the presence of each device at runtime. Upon receiving a message (typically a beacon) from a new device, the software hub notifies the simulation engine about this discovery. The simulation engine then adds a node to the dynamic graph it maintains, and starts moving this node according to a pre-defined mobility model (Random Waypoint, Levy Walk...). A mobile device is considered as "lost", and is thus removed from the dynamic graph, if the software hub does not receive any message from that device for a while. This mode of operation is very useful during demo sessions, when the dynamic graph is displayed in real time on a screen, so the users can observe how they are assumed to "move", as seen from the simulation engine's viewpoint. This mode is also

useful to obtain energy consumption measurements of the protocols under evaluation while they are executed on a real device.

- *Experiment involving virtual nodes*: in the abscence of real devices, each SN runs as a Unix process containing at least the implementation of an opportunistic networking system and the application code. This application code is the real application code, unless the application is interactive (in this case a simulated user's behavior must obviously be added). As with real devices, the software hub discovers the presence of every node when their first message is sent, and it notifies the simulation engine to add it to the dynamic graph. Hundreds of nodes can be executed in a single desktop workstation or laptop, as shown in Fig. 2.b). When the number of virtual SNs is very large, these SNs can be distributed on the nodes of a computer cluster (Fig. 2.c), with a distribution based on a round-robin policy, so the workload is statistically balanced between all cluster nodes.
  When the experiment involves a stable population of SNs, multiple SNs can be created by LEPTON at the beginning of an experiment. Alternatively, the lifecycle of the SNs can be dictated by a DGS file. SNs are then created and terminated dynamically, according to the events observed in that file. While reading the file the simulation engine can trigger the creation or termination of a SN at the appropriate time.

These modes of operation can be combined when necessary. For example, an experiment may involve virtual nodes (running as concurrent processes on a host platform) as well as real nodes (running on real devices), thus scaling up to a large population of virtual nodes, while creating, terminating and simulating the mobility of the virtual nodes according to a DGS file.

E. Visualization

The GraphStream library provides advanced support for graph visualization. LEPTON can use these features to display (see Figure 6) the evolution of the wireless network. This visualization can be done on-the-fly while an experiment is running (which provides for nice demo sessions), when replaying an experiment, or in order to produce videos. Examples of such videos are available on LEPTON's web page[2].

Replaying a simulation consists in reading the DGS file that has been produced earlier during an experiment. This can be done at any desired speed, including step by step, and it makes it possible to observe what has happened in the network at a specific time during the simulation. The history of the radio contacts of a specific node can for example be retraced that way. Moreover, when the information contained in the DGS file is combined with middleware-level or application-level information gathered from execution logs, advanced analysis can be performed. For example, the multi-hop propagation of a specific message during an experiment can be observed in detail.

[2]http://www-casa.irisa.fr/lepton/videos.html

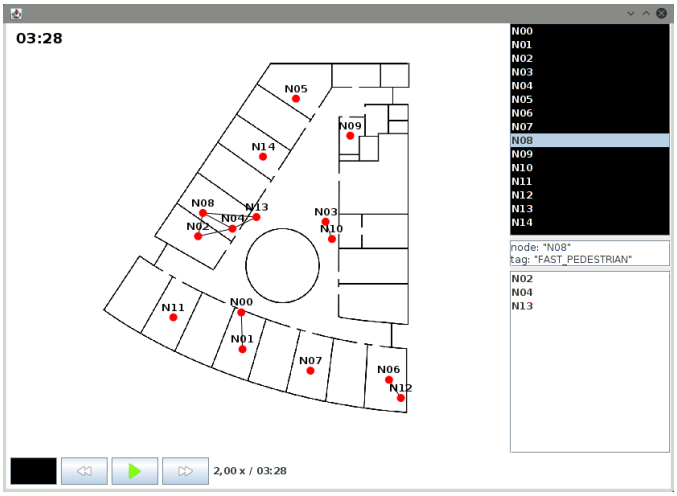

Figure 6. LEPTON's snapshot during an emulation run involving 15 devices moving in an office floor, assuming a transmission technology with very short transmission range (in that case, 5 meters). The SNs are depicted with red dots, and the thin black lines depict radio links.

F. Additional tools

In addition to the simulation engine and the software hub, LEPTON comes with a set of tools that can be used either when preparing an experiment, or after an experiment has been conducted. This toolset notably includes format converters to manipulate traces, as well as programs that can be used to analyze a mobility scenario.

Input and output of tracesets: Mobility or contact tracesets such as those available in CRAWDAD [14] can be converted to the DGS file format used by LEPTON. Converters are notably available for tracesets that use the file formats defined for well-known simulators (e.g., ONE, Legion), but ad hoc converters can also be devised for tracesets that use exotic formats.

Sometimes the traceset considered as an input contains information about both node mobility and radio contacts, but sometimes it only contains information about node mobility. In the latter case programs available in LEPTON's toolset can be used to calculate radio contacts, so information about these contacts gets inserted directly in the output DGS file. When reading such a file, LEPTON's simulation engine will simply drive communication according to these precalculated contacts, rather than calculating them at runtime.

Analysis of mobility/contact scenarios and experimental results: Several programs provided in LEPTON's toolset make it possible to analyze a mobility scenario in details (based on the corresponding DGS file), but also the log files produced during an experiment. With these programs traditional statistical results about the number of nodes, of neighbors, of contacts and inter-contacts, etc. can be produced easily.

The toolset also includes analysis tools that make it possible to investigate a scenario, or the results of an experiment, at a very fine grain. The dissemination of a single message can for example be examined in details, or the evolution of the neighborhood of a single node. With such tools the results produced by different experiments (run with the same

scenario) can be compared, and these results can also be compared with the theoretical behavior that can be expected from the system under test. Predicting the expected behavior of the system under test is indeed an important feature of LEPTON's toolset, as it provides a reference to which actual results can be compared.

Among other tools, LEPTON's toolset thus implements an algorithm that can determine the *propagation horizons* of messages [30]. Basically, the horizon of a message sent at *(S, t_S)* is the set of all potential receivers for that message, considering the evolution of the network graph starting from t_S. A node *R* is a potential receiver for a message sent at *(S, t_S)* if there exists at least one journey (or temporal path) from *(S, t_S)* to *(R, t_R)*, with $t_R \geq t_S$. Computing the horizon of a message therefore provides input about the "ideal" propagation of that message. It also provides a reference with which the actual propagation of the message, as reported in log files, can be compared after an experiment. An illustration of how this algorithm helps analyze experimental results is provided in Section V.

IV. Integrating opportunistic systems with LEPTON

Wireless network emulation with LEPTON is expected to be an excellent way to validate the code of a full-featured opportunistic system, as well as a way to compare the performances observed with different systems in similar conditions. In order to confirm this expectation, two existing systems have been tested using LEPTON, using the same mobility scenario, as well as the same application scenario.

These two systems were chosen for three reasons: 1) because they can both support content-based networking[3], that is, a model where information flows towards interested receivers rather than towards specifically set destinations; 2) because the two systems use a similar communication pattern, as they rely on UDP beaconing for neighbor discovery, and on TCP sessions for the gossiping; and 3) because they were designed and implemented by two different research groups targeting different nodes' hardware, so they may require of slightly different adaptations in order to interface with LEPTON.

These two systems are exemplary in nature to demonstrate the capabilities of LEPTON, how to interface different systems with it[4] and how to use it to compare and evaluate their performance. Therefore, it is not intended to evaluate or assess their performance on a certain scenario, and the focus should be placed the process that leads to the results and on LEPTON's usefulness, but not on the results obtained nor on the compared systems.

In the remainder of this section we present the two systems, and we explain how we developed the SN (and the required software hub) we used to evaluate each system through LEPTON.

[3]This is not the only type of networking compatible with LEPTON. The authors are currently working on adapting IBRDTN [31] and DTN2 [32], which are destination-based networking systems.

[4]Further details about this integration process are available on http://www-casa.irisa.fr/lepton/doc/howtos/emulated_nodes.html.

A. DoDWAN

DoDWAN *(Document Dissemination in mobile Wireless Ad hoc Networks)* is an open-source middleware system that can support content-based networking [33] in partially or intermittently connected wireless networks ***[34], [35]***. It is implemented as Java code, and it is distributed under the terms of the GNU General Public License, together with documentation for users and developers[5]. Binaries of an application suite based on DoDWAN are also available as a demonstrator for Android smartphones and tablets ***[36]***.

Application services running on a mobile device can interact with DoDWAN through a *pub/sub* (publish/subscribe) API. The objects that can be published and subscribed for through this API are called *documents*. A document is basically a payload, and meta-information characterizing this payload (type, author, production date, keywords, etc.).

In order to be able to receive documents, an application service must first *subscribe* with DoDWAN, and provide a *selection pattern* that characterizes the kinds of documents it is interested in. When several application services run on the same host, their selection patterns together define the host's *interest profile*. DoDWAN uses this profile to identify documents that must be exchanged whenever a radio contact is established between two hosts.

A gossiping algorithm drives interactions between neighbor nodes in an opportunistic way, leveraging every radio contact between two nodes to allow these nodes to exchange documents according to their respective interest profiles. This gossiping algorithm takes inspiration from the Autonomous Gossiping (A/G) algorithm [37], which itself defines a selective version of the epidemic routing model proposed in [38]. In DoDWAN's gossiping algorithm, though, special attention has been paid to avoiding any unnecessary transmission between neighbor hosts, while ensuring maximal reactivity to connectivity changes observed on the wireless channel.

DoDWAN is most commonly used with Wi-Fi interfaces running in either managed or ad hoc mode. Yet over the last few years it has also been tested with a variety of alternative wireless technologies, such as Bluetooth, XBee, and VHF battlefield radios used in military tactical networks ***[39]***.

Integration of DoDWAN with LEPTON: Since DoDWAN can rely on different kinds of wireless technologies, no assumption is made about the format of the addresses used to identify mobile hosts at data link or network level. Indeed, assuming that a node is always identified by an IP address would be contrived when running for example with Bluetooth or XBee radio modules. DoDWAN's implementation therefore assumes that each wireless host is identified by a unique identifier, expressed as a character string. This identifier is embedded in the beacons used for neighbor discovery, and in the messages used in gossiping transactions.

Developing a software hub capable of receiving and forwarding traffic for DoDWAN nodes, based on the templates presented in Section III-C, is therefore straightforward. Besides, no modification of DoDWAN's source code is required to redirect all transmissions to this software hub. This is

[5]http://www-casa.irisa.fr/dodwan

because the configuration file that determines how a DoDWAN node should interact with peer nodes is quite flexible. For example the IP addresses and port numbers used for the beaconing and gossiping services are defined in that configuration file (assuming IP is used), so redirecting all transmissions to a software hub only requires changing that configuration file.

B. Active-DTN

The variety of routing protocols defined for mobile and opportunistic networks over the last decades suggests that no general-purpose routing strategy can satisfy the requirements of all applications at once. Active-DTN (or aDTN for short) is a Bundle Protocol [40] compliant open-source middleware system that can help meet the different needs of multiple applications that share the same wireless network, by allowing these applications to specify how their messages should be processed while traversing the network [41]. aDTN extends the structure of bundles so they can carry routing code together with plain data. More specifically, aDTN messages use the Mobile code Metadata Extension Blocks (MMEB, defined in RFC 6258 [42]) to carry their own routing code, lifetime control code and bundle prioritisation code. These codes are executed by any node that receives a bundle, in order to decide to what neighbors should the bundle be forwarded to, if it should be dropped, and the priority it must receive. The system has been implemented in C++, and it is distributed with libraries that allow the development of DTN applications in C, Java, or Python.

Active-DTN was designed to demonstrate the feasibility of this paradigm of bundles that carry executable codes, and to obtain data from low-scale field experiments. Several of such experiments have been run successfully on the Autonomous University of Barcelona (UAB) campus, involving up to 10 Raspberry Pi nodes that were carried either by pedestrians, bus shuttles, cars, and a drone.

Integration of aDTN with LEPTON: By default aDTN uses UDP multicast for neighbor discovery, and TCP sessions for the transfer of bundles between neighbor nodes. The main difference with DoDWAN with that respect is that a transient TCP session is established whenever a bundle must be transferred between two aDTN nodes, while a single TCP session is maintained between two DoDWAN nodes as long as these nodes are neighbors.

The software hub needed to interface aDTN with LEPTON simulation engine was developed by making small adaptations to the templates presented in Section III-C.

Additionally, as aDTN is not as configurable as DoDWAN, the code of the aDTN system had to be slightly modified, so that an aDTN node could redirect the messages to the software hub. First of all, the Neighbor Discovery module was modified so this module could listen to a UDP socket bound to a unicast address, rather than to a multicast address. Conversely the beacons sent through that socket must be addressed to the hub, rather than to a multicast group.

The Bundle I/O module was also modified so that messages sent to the hub carried the identity of their real destination. In the original implementation of aDTN, bundles sent directly to a neighbor node are preceded by a short header that only specifies the sender's ID and the bundle's length. The ID of the receiver node is not specified, since this is by construction the node that receives the message. However, when a bundle is relayed through the software hub, the header that precedes a bundle must be extended so the ID of the receiver is also specified in the message's header. Figure 7 illustrates this process.

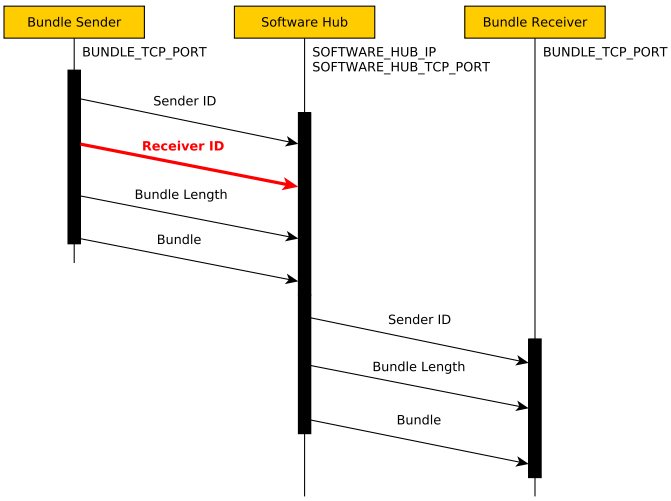

Figure 7. Sequence diagram of the relay of one bundle by the software hub between an aDTN node and a neighbour. The Bundle I/O module was modified to add the receiver ID to every message, because the software hub requires it to know the intended destination.

V. LEPTON IN PRACTICE

As explained in Section I, the engineering process of any DTN system (after the initial design and modeling phases) should ideally include a simulation phase (based on pseudo-code), an emulation phase (based on real code), and ultimately a deployment phase (in a real-life setting, with real devices and, potentially, real users). In order to demonstrate that none of these phases should be neglected, we show in this section that submitting a DTN system to a stress-test in an emulation platform such as LEPTON is an excellent means to pinpoint weaknesses that would not necessarily appear with a discrete-event simulator. We also show that, unlike a simulator, an emulation platform can provide testing conditions that closely mimick real-life conditions.

In Section V-A we first present the scenario we defined to evaluate the two DTN systems (DoDWAN and aDTN) presented in the former section. In Section V-B we present results we obtained after simulating both DoDWAN's and aDTN's behaviors with the ONE simulator. We then present in Section V-C the results obtained after running the implementations of both DTN systems with LEPTON, and show that unlike pure simulation, emulation helped differentiate these systems, and pinpoint and correct an unexpected weakness in aDTN's code. Finally, in Section V-D we describe a small-scale field experiment we performed with mobile devices

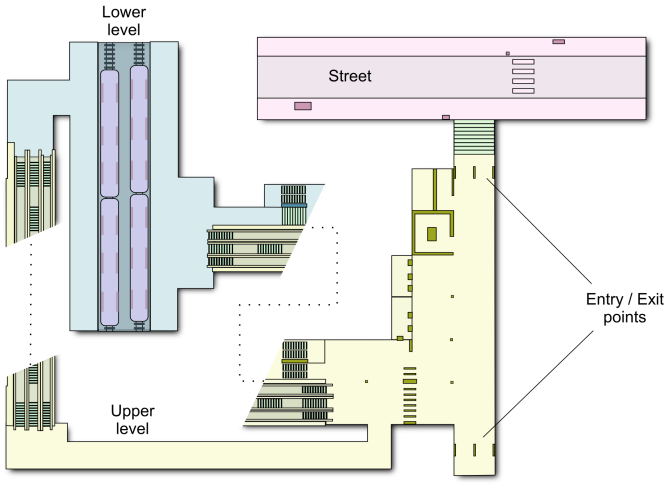

Figure 8. Overview of the subway station. There are two different levels that are connected between them by a pair of stairs (depicted with dotted lines).

running DoDWAN, and show that this scenario could be reproduced very accurately with LEPTON.

A. Scenario considered for both simulation and emulation runs

Mobility scenario: For this evaluation we used mobility traces available in the CRAWDAD database [14], in the *kth/walkers* dataset [43]. This dataset includes mobility traces that have been produced using the Legion Studio simulator [44], for both an outdoor scenario (pedestrians in the Östermalm area of central Stockholm) and an indoor scenario (pedestrians in a two-level subway station) [45]. We selected the subway scenario, that comes pre-defined with Legion Studio, and that models a train platform connected via escalators to the upper entry-level (see Fig. 8). Pedestrians can arrive from several entry points of the subway station, or when trains arrive at the platforms. They can likewise leave the station through an exit point, or by boarding a train. In the traceset the location of each walker is defined every 0.6 second, and the traceset covers exactly one hour.

This scenario is interesting because, unlike many scenarios based on algorithmic mobility models, it involves a continuously changing population of mobile nodes that move in a relatively restricted area (1921 m^2). Indeed, most walkers that enter the station only stay there for a few minutes. The challenge when considering such a scenario for wireless network emulation is that instances of mobile nodes must be created and deleted continuously during the whole duration of an experiment. Besides, the density of mobile nodes in the subway station is high, for each node can be in radio contact with dozens of other nodes at any time, even when considering a short radio range.

The subway station scenario is analyzed in detail in [45] (assuming 10 m and 30 m radio range) and in [46] (assuming 10 m and 50 m radio range). For our evaluation, we only considered the 10 meter range. Since the *kth/walkers* dataset only provides information about nodes positions, one of the programs available in LEPTON's toolset was used to precalculate radio contacts, thus producing a DGS file containing both nodes positions and radio contacts.

Application scenario: Besides the mobility scenario, we have defined the following simple topic-based application scenario (topic-based messaging can be considered as the simplest form of content-based networking). The messages produced during the experiment will pertain to five different topics, numbered 0 to 4, each message pertaining to a single topic. Each mobile node is assumed to be interested in only one topic, and is thus willing to receive and forward only messages pertaining to that topic. The population of mobile nodes is thus split into five distinct groups, each group including all nodes that share an interest for the same topic. Since nodes in the subway traceset are numbered from 0 to 3299, we decide that node *#i* is interested in topic *#(i mod 5)*. As soon as it starts running, each node sets a subscription in order to receive all messages pertaining to the topic it is interested in. Because of the DTN model, nodes that are interested in topic *#j* will therefore serve as both receivers and carriers for all messages pertaining to that topic.

During the experiment, a number of messages must be published, and we are interested in analyzing how these messages propagate in the network. No message is published during the first five minutes of the experiment. This bootstrapping interval allows for the first mobile nodes to populate the simulation area. Likewise, no message is published during the last 15 minutes of the experiment. This allows for the last published messages to propagate for a while before the experiment completes. Messages are thus published during a 40 minute time window, which goes approximately from time t=5' to time t=45' during the experiment. More specifically, the first node to enter the subway station during the time window is node #200, and the last one is node #2679. Only 20 % of the nodes created in this time window are allowed to publish a message, each publisher node publishing a message pertaining to the topic it is itself interested in. In order to determine which node can act as a publisher, we use the following simple method: *for node #i, if (i mod 25) <= 5, then node #i is a publisher for topic (i mod 5)*.

With this simple application scenario, 500 messages are published during the experiment (100 messages for each topic).

The analysis tools available with LEPTON have been used to examine the DGS file produced from the subway traceset. Some of the figures thus obtained are summarized in Table I.

Besides these statistical data, the horizons of the 500 messages considered in our application scenario have been computed using LEPTON's horizon calculator. Figure 9 shows the number of potential receivers for each of these messages. It can be observed that the later a node is created during an experiment, the fewer the potential receivers for a message published by this node.

As explained in Section III-F, computing the horizon of a message provides input about the "ideal" propagation of that message. Indeed, the actual number of nodes that can be expected to receive a message is a key requirement for calculating delivery ratios, once experimental results are available.

Metric	Values (* = min / max / avg / stdev values)
Duration of the experiment	3,600”
Nb. of nodes created during the experiment	3,300
Nb. of active nodes per experiment step	2 / 325 / 180 / 57 (*)
Activity duration per node	6.000” / 11’53.400” / 3’15.366” / 1’39.301” (*)
Number of radio contacts	384,465
Number of neighbors per node	0.7 / 73.8 / 25.25 / 11.8 (*)
Durations of radio contacts	0.600” / 6’01.200” / 24.486” / 34.776” (*)
Number of inter-contacts	78,591
Durations of inter-contacts	0.600” / 6’11.400” / 27.114” / 44.424” (*)

Table I
STATISTICS ABOUT THE SUBWAY STATION SCENARIO, ASSUMING 10 METER RADIO RANGE.

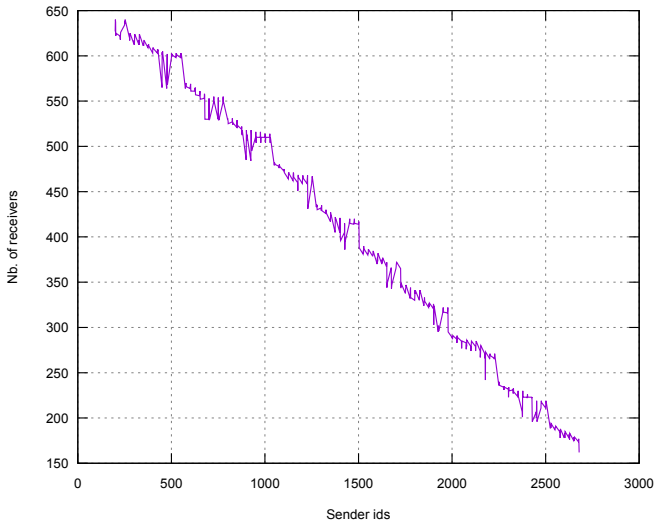

Figure 9. Number of receivers accessible from each message publisher, according to the horizon calculation. As expected, the later a node enters the scenario, the lesser the amount of accessible receivers it can reach.

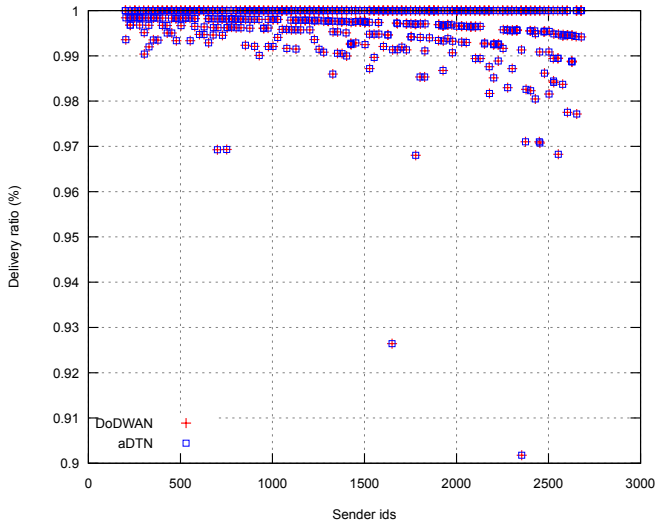

Figure 10. Delivery ratio obtained by DoDWAN (red cross) and aDTN (blue square), relative to the amount of potential receivers. Note that the two systems perfom exactly the same on the simulations.

B. Simulating the subway scenario

In order to observe how the two middleware systems presented in Section IV (i.e., DoDWAN and aDTN) can perform when running the subway scenario, simulations have first been run using the ONE simulator [12], [13].

Simulation setup: We have converted the LEPTON’s DGS files we introduced in the previous paragraphs into the format required by the *ExternalEventQueue* (for the establishment of connections and the creation of messages) and *ExternalMovement* (for the nodes’ movement) ONE’s java classes. We also extended ONE’s *ActiveRouter* to develop a *DoDWANRouter* and an *ADTNRouter* classes that replicate DoDWAN’s and aDTN’s behaviour. This way, we have been able to evaluate how DoDWAN and aDTN should behave on the subway scenario.

As all the simulation events are deterministic (because all movement, connectivity and message creation events are fixed, and none of the two DTN systems introduce any randomness by design), all the experiments we performed have generated the same results, so we present them as the results of only one experiment. Finally, messages’ payloads were set to 200 bytes, and their lifetime was unlimited.

Message delivery ratios: In order to improve the precision of the delivery ratio metric, we have benefited from the horizon computation. This way, for each message and for each middleware system, we calculated de delivery ratio as NR_O/NR_I, where NR_I is the number of potential receivers (or “ideal” number of receivers), as predicted by the horizon computation, and NR_O is the number of effective receivers, as observed from the simulator’s log (or “observed” number of receivers).

As shown in Figure 10, both DoDWAN and aDTN have delivered exactly the same amount of messages to the same receivers (they both delivered a message to 99.7% of the potential receivers). This was not unexpected. The reason is that, although DoDWAN uses a publish/subscribe strategy based on topics of interest, while aDTN uses mobile code to make routing decisions, in the described application scenario, both systems operate using the very same logic. Therefore, both systems take the same routing decisions and, when they face the same situation, they both forward and deliver the messages to the same nodes. Besides, since processing time and buffer management are not a problem in a discrete event-based simulator (because the whole world “stops” while a node has to make a decision) there is nothing that differentiates one from the other from the simulator’s point of view.

Other metrics: One of the advantages of using a simulator is that it makes it easy to collect global statistics of

Metric	Value
Started relay events	204,663
Finished relay events	203,223
Aborted relays	1,440
Average latency	961.49 s
Average hopcount	14.01 hops

Table II
SOME OF THE MAIN METRICS OBTAINED USING THE ONE'S *MessageStatsReport*. SIMULATIONS PERFORMED WITH DODWAN AND ADTN HAVE OBTAINED EXACTLY THE SAME VALUES.

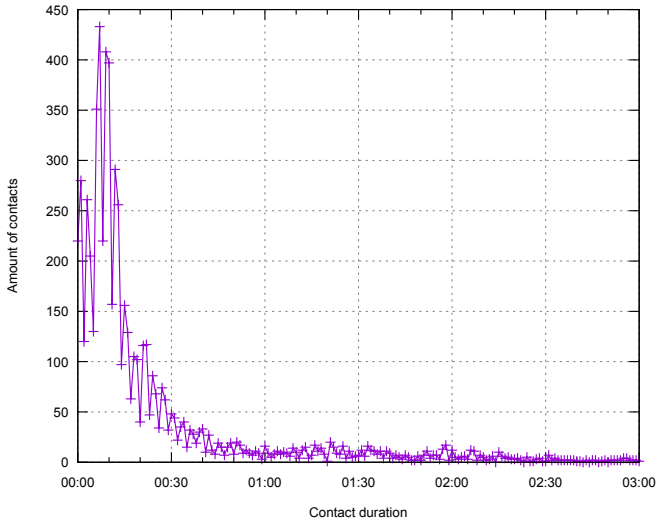

Figure 11. Contact duration time. Two thirds of the contacts last less than 15 seconds, and the vast majority of them last less than 30 seconds.

certain metrics. In this case, we have used some of the *Report* classes that are distributed with ONE, to build Table II, that shows some of the typically most used metric statistics about the experiment, and Figure 11, that shows the distribution of contact durations.

Regarding the high amount of aborted relays, it is very interesting to study contact durations, because all abortions are due to premature disconnections between transferring nodes. Note that the 7.7% of the contacts last less than 2 seconds, so they are very easily to skip if one of the nodes is busy transmitting a bunch of messages. This is what happened with messages 1650 and 2354, that could not be delivered to some early carriers because they where occupied, causing that, respectively, 27 and 22 potential receivers have never received the messages.

These contacts would be exploited (or not) to forward messages depending on the system implementation's performance. Additionally, 16.7% of the contacts last less 5 seconds. This is the beaconing period that both DoDWAN and aDTN use, so all those very short contacts could also be missed by their neighbour discovery modules.

However, these factors (and many others) have not been considered in this experiment, as ONE does not simulates beaconing (it directly notifies a node when a new connection is created) and it treats routing decisions as atomic operations. Therefore, there is no way to know the impact of those factors on the overall performance of the two compared systems using

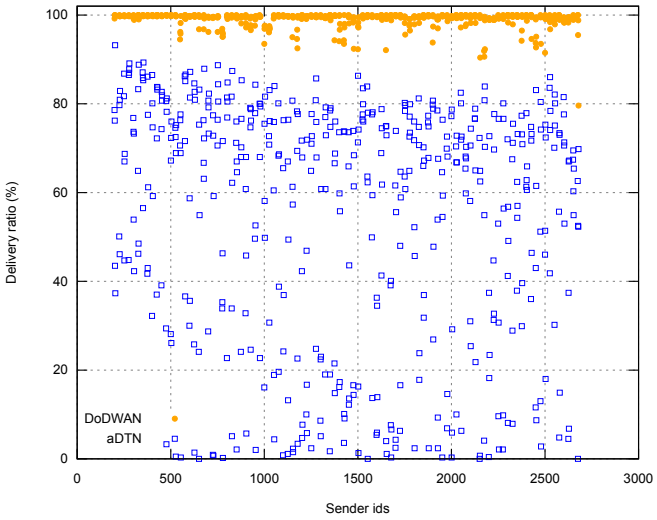

Figure 12. First set of delivery ratio results observed with systems aDTN (blue squares) and DoDWAN (orange dots). For almost all senders, DodWAN has delivered their messages to more than 90% of the possible destinations, while aDTN has performed very inconsistently, delivering messages to between the 0% and the 90% of the possible destinations.

simulation.

Summarizing, what these results show is that simulation is a good tool to know that both aDTN and DoDWAN should be able to perform equally well in this scenario and that their designs do not contain any incompatibilities or flaws. According to the simulation results, there is no difference between the two systems when applied to this scenario.

C. Emulating the subway scenario

Both DoDWAN's and aDTN's real implementations have been used to run the subway scenario, using LEPTON as an emulation platform. Note that our prime objective here is to present how the conclusions thus obtained differ from those obtained with pure simulation. Performing an exhaustive analysis and comparison of DoDWAN and aDTN is out of the scope of this paper.

Emulation setup: Experiments were run using virtual nodes with the two middleware systems. In all experiments the systems were configured so as to use a beaconing period of 5 seconds, and to assume unlimited cache capacity. As in the simulations, messages were published with a 200 byte payload, and unlimited lifetime.

Message delivery ratios: Again, we have calculated the delivery ratio in relation with the amount of potential receivers we obtained through the horizon computation. Figure 12 shows the delivery ratios observed after running the same application scenario with aDTN's and DoDWAN's SNs. Note that these are not statistical results, as they present figures obtained after one experiment run for each of the two systems considered.

In this figure it appears that aDTN performs quite poorly. The average delivery ratio observed with this system is actually 54.6%, while that observed with DoDWAN is 98.9%. These first results came as a surprise, since before running with LEPTON, aDTN had first been extensively tested using

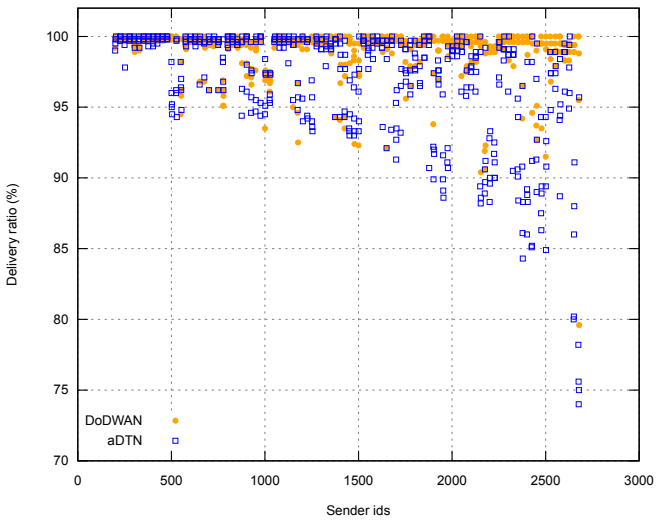

Figure 13. Second set of delivery ratio results observed with systems aDTN (blue squares) and DoDWAN (orange dots), after correcting aDTN's issue.(note than the y-axis starts at 70 instead of 0). Although DoDWAN slightly outperforms aDTN, the overall results are very similar.

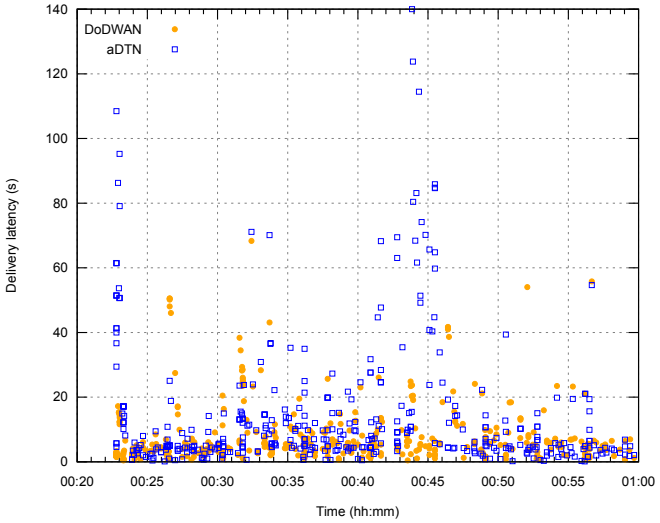

Figure 14. Delivery latencies observed during the dissemination of message #1251, when running either aDTN (blue squares) or DoDWAN (orange dots) virtual nodes using LEPTON. The obtained latencies are very similar, except in two moments: at the start of the message's dissemination, and around 00:43, where DoDWAN outperforms aDTN.

the ONE simulator, as well as using a dozen of Raspberry Pi micro-computers as experimentation platforms. On both occasions aDTN seemed to perform quite satisfactorily. It is only when being tested with LEPTON that aDTN proved unable to support the workload imposed by a demanding scenario such as that of the subway station. After a detailed examination of the implementation of aDTN, it appeared that the buffer management system in aDTN sometimes maintained duplicates of messages on the same node. This error was easily corrected, and another experiment was run with LEPTON and aDTN. The new delivery ratios obtained with aDTN are shown in Figure 13 (together with those obtained earlier with DoDWAN), and this time it can be observed that both systems show roughly similar performances. Indeed, the average delivery ratio observed with aDTN is 97.1%, against 98.9% for DoDWAN.

This experience confirms that running a DTN middleware system in emulation mode is an excellent stress test for the system considered. Bad or weak implementation choices can be revealed in such conditions, while they could remain undetected when running pure simulations, or when running real code on a handful of experimentation platforms. Besides, this has been also useful to confirm that, even when the simulation has shown that both systems could perform equally in this scenario, their implementations play a key role on differentiating them. Even as the delivery ratio obtained by both systems are quite similar, there are obvious differences between them that were not spotted in the previous simulations, and these differences may be crucial in some scenarios.

Interestingly, Figure 13 also shows that the amount of not delivered messages increases with time during an experiment. This is because when the system under test fails to exploit a contact between two SNs, the chance to compensate for this failure by exploiting later contacts gets weaker as the simulation gets closer to completion. This phenomena affects aDTN's results more heavily than DoDWAN's ones because aDTN's latencies (presented on the following paragraphs) are higher than DoDWAN's ones. Therefore, aDTN nodes get even fewer chances to compensate the missed opportunities.

Message delivery latencies: Another interesting metric to consider when analyzing experimental results is the latency of message delivery, as observed for each receiver. In that particular case we are not interested in the delay between the time a message is published and the time it is received. We are interested in the delay between the time a message *could* be received by a node (according to the horizon calculation), and the time it is *actually* received by that node (as recorded in the experiment's log files). Figure 14 shows the latencies observed (with both aDTN and DoDWAN) for message #1251, and for each of the receivers of that message. In this figure, a symbol that shows for example a latency of about 80 seconds at time 00:24 indicates that a receiver that *could* have received message #1251 at a certain time (according to the horizon computation) actually received it 80 seconds later during the experiment.

In Fig. 14 it can be observed that when running the simulation with aDTN nodes, about 20 of these nodes did not receive message #1251 as early as possible during the first minute of the message's dissemination. Note that this did not happen on the simulation (see Figure 15, that shows that only four nodes received the message more than 30 seconds later than the expected). These nodes that received the message a lot later than expected could not serve as early carriers for the message, which compromised the speed of that message's dissemination in the subway station. Further investigations (based on a thorough analysis of the log files) would be required in order to explain this phenomenon.

LEPTON's toolset provides programs that can help perform such fine grain analysis, based on an automatic extraction of

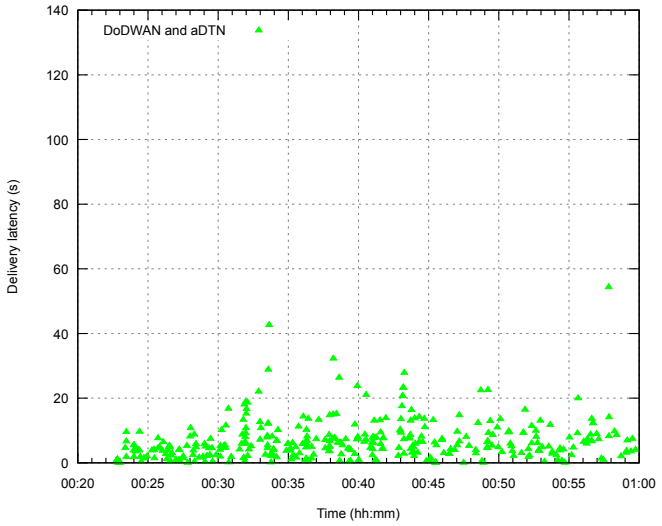

Figure 15. Delivery latencies observed during the dissemination of message #1251 using the ONE simulator, when running either aDTN or DoDWAN (they obtained exactly the same results).

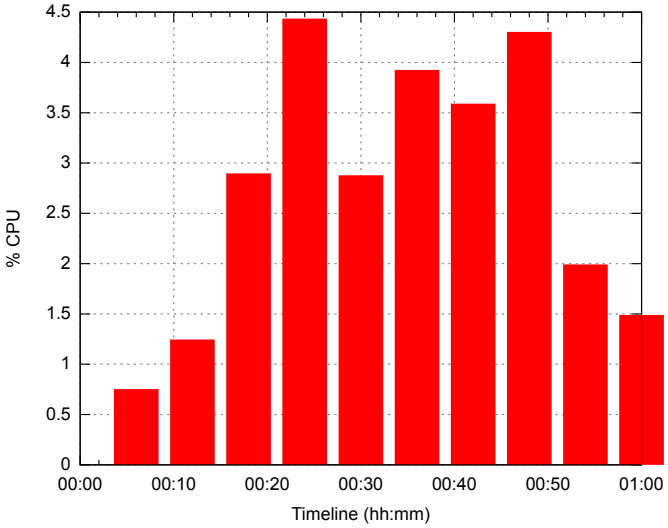

Figure 16. CPU load observed on the host running LEPTON during an experiment (with DoDWAN nodes).

data from log files. This topic remains out of the scope of this paper, though, since our main objective here is simply to demonstrate how fully-implemented opportunistic systems can run in an emulated environment, thanks to the LEPTON platform, producing a wealth of results that can then be analyzed at a very fine grain.

a) Considerations about LEPTON's resource consumption: Experiments based on the subway scenario have been run on a cluster composed of 20 cluster nodes (each node including 2 "6-core" Xeon L5640 processors), interconnected via a Gigabit Ethernet LAN, each cluster node running Linux. In practice, only 5 nodes of the cluster were used during our experiments. One of these nodes ran the LEPTON emulator (i.e., simulation engine plus the appropriate software hub), and the other 4 nodes ran instances of the DTN system considered, which were created dynamically at runtime and interacted via the hub. In our experiments, we deployed only one software hub, and we have studied the workload imposed to the node running it.

Figure 16 shows the evolution of the CPU workload observed on that node during an experiment involving DoDWAN's SNs. Experiments based on aDTN produced roughly similar results. It can be observed that simulating the mobility of mobile nodes (which is basically the task of LEPTON's simulation engine), and controlling communication between these nodes (which is that of the software hub), only use a small fraction of the CPU capacity. The CPU load observed on the cluster nodes hosting SNs was also measured during experiments. This workload was even lower than that shown in Figure 16, even though each cluster node had to run dozens of concurrent SNs at any time (with occasional peaks at about 80 SNs per cluster node).

Figure 17 shows the network throughput observed on the same cluster node during an experiment. The maximal throughput observed is slightly above 16 Mbps. This is only a small fraction of the network capacity in that configuration.

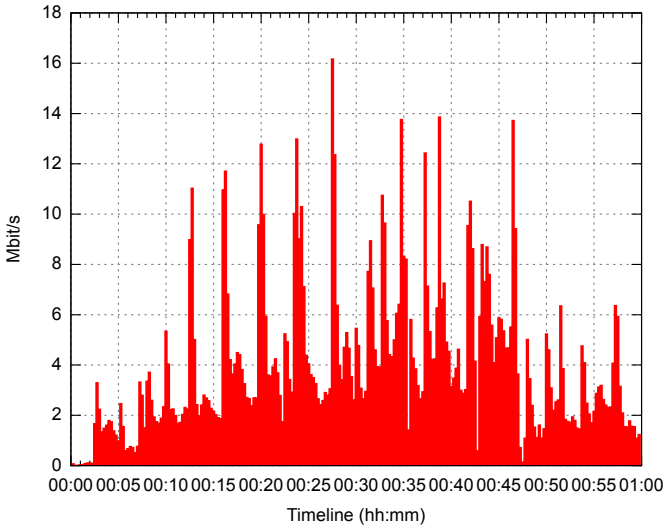

Figure 17. Network throughput observed in the software hub during an experiment (with DoDWAN nodes).

These figures confirm that LEPTON is indeed a lightweight emulation platform, that makes it possible to run emulation-based experiments on standard laboratory equipment. With many simulation scenarios an experiment can be conducted on a single workstation or laptop, and for more demanding scenarios a couple of standard workstations (or part of a cluster) can be used.

D. Field experiment with DoDWAN

Whenever running a certain scenario with a simulation or emulation platform, it is always legitimate to wonder whether the results thus obtained bear any resemblance to what could be observed in real conditions. In order to demonstrate that LEPTON makes it possible to run emulation-based experiments in conditions that closely mimick real-life conditions, we have conducted a small-scale field experiment based on

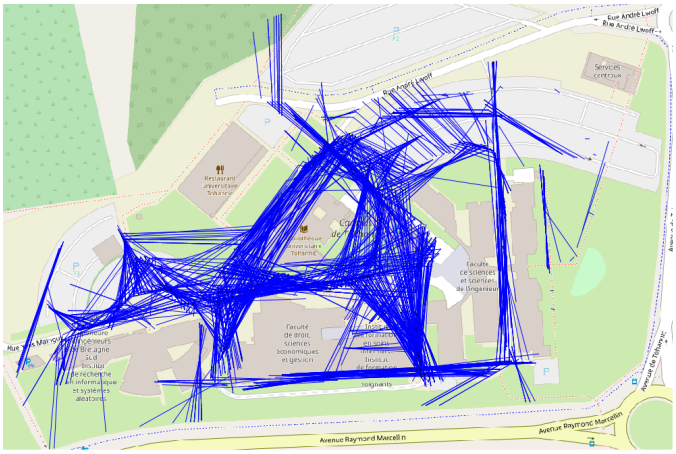

Figure 18. Map of the campus where the field experiment was conducted. Every blue line represents the transmission of a message between two smartphones.

DoDWAN, and then reproduced the same scenario with LEPTON.

Mobility and application scenario: 10 volunteers were equipped with HTC Wildfire-S smartphones, whose Wi-Fi interfaces were configured to operate in ad hoc mode. Each smartphone ran DodwanDroid[6], an Android application based on DoDWAN, as well as another application recording GPS locations and radio contacts.

During the experiment the volunteers were asked to roam a small university campus (covering roughly a 420m x 160m area, as depicted in Figure 18), staying outdoors (because of the GPS recording), while using DodwanDroid to exchange short-text messages on a public channel (meaning every message published by one volunteer was meant to be received by all other volunteers). The experiment lasted about 25 minutes, but the volunteers were requested to start DodwanDroid on their smartphone shortly after they began walking in the campus, and to stop it shortly before the 25-minute deadline. They were also requested to stop publishing messages a couple of minutes before stopping the DodwanDroid application, thus giving the last messages published a chance to disseminate for a while.

Log files were collected and analyzed after this experiment. Some of the statistical figures thus obtained are shown in Table III[7]. The last line in that table provides interesting results about the distances covered during message transfers between neighbor nodes. During the field experiment some messages got transmitted over up to 172 meters, with an average distance of 56 meters. Such rather long transmission distances could of course be observed because the volunteers stayed outdoors while roaming the campus. Regarding delivery ratio, 249 messages were published by the volunteers during this experiment, and 2234 receive events were recorded in the log files. Since each message could be received by at most 9 smartphones, the delivery ratio is 2234/(249*9)=99.6%.

[6]http://www-casa.irisa.fr/dodwan/dodwandroid.html

[7]Further details about this field experiment (including videos) are available on http://www-casa.irisa.fr/dodwan/field_expe_2018_01.html

Field results vs emulation results: A history of node creation, node deletion, publish events, GPS and radio contact data extracted from the log files were used to produce a DGS file. This way, we used LEPTON to replay exactly the same application-level scenario (using virtual nodes instead of the real devices), starting and stopping each node, and issuing publish commands at the appropriate times. Three emulation runs were performed with LEPTON so as to check its stability.

Log files produced during each emulation run were analyzed using the same method as for those obtained after the field experiment. As expected, this analysis shows that the pattern of message dissemination is almost the same in real-world and emulation conditions (note the similitude between the two colums in Table III). One perceptible difference, though, is that the number of contacts observed during emulation runs is lower than that observed during the field experiment. This is because a few very short intercontacts (lasting typically less than 500 ms) occurred during the field experiment. Such short intercontacts can be missed by DoDWAN nodes when running with LEPTON, two adjacent contacts being then perceived as a single, longer one. Another difference is that 2236 receive events occurred during each emulation run, against 2234 in the field experiment. Indeed, a 2.6 second radio contact occurred between two smartphones, shortly before the completion of the field experiment. These two smartphones failed to exploit this very short contact, and thus failed to achieve the last two message transfers. During the emulation runs, though, these message transfers did not fail, hence the slight (less than 0.1%) discrepancy in the number of receive events between field data and emulation data.

Figure 19 shows the average message delivery delays obtained in all four datasets. The similitude between the four curves confirms that LEPTON's behavior is stable, and that it mimicks real conditions satisfactorily. Based on these results, and despite the minor differences discussed above, we consider that the overall similarity between field results and emulation results demonstrates that LEPTON is apt at replaying accurately field scenarios in emulation mode.

VI. Conclusion and Future Work

In this article, we have presented LEPTON, an emulation platform we designed for opportunistic networking development. This platform is meant to constitute a lightweight emulation solution that bridges the gap between pure simulation, and real-world experimentation. With LEPTON, full-featured application and/or middleware code can run in an emulated environment, in which only the mobility of network nodes is simulated. The code running on each network node is therefore the same code that could run on a mobile device. In fact, instances of the system under test can be executed concurrently on real mobile devices (such as smartphones or tablets), on a single workstation, or on a cluster of workstations, while LEPTON simulates their mobility and drives transmissions accordingly.

Initial experiments conducted with two existing opportunistic systems have confirmed that such systems can be easily adapted to run with LEPTON, and that running opportunistic

Metric	Field experiment	Emulation run
Duration of the experiment	25'20"	
Nb. of nodes involved in the experiment	10	
Nb. of active nodes	2.0 / 10.0 / 9.5 / 1.5(* = min / max / avg / stdev)	
Activity duration per node	21'35" / 23'53" / 23'06" / 41" (*)	
Average number of neighbors per node	0.0 / 2.7 / 1.2 / 0.5(*)	0.0 / 2.4 / 1.2 / 0.5
Number of contacts	163	157
Durations of contacts	00" / 10'16" / 48" / 01'11"(*)	01" / 10'19" / 49" / 01'13"(*)
Number of inter-contacts	119	113
Durations of inter-contacts	04" / 18'12" / 04'01" / 03'37"(*)	04" / 15'07" / 04'03" / 03'25"(*)
Number of messages published	249	
Number of receive events	2234 (delivery ratio: 99.6%)	2236 (delivery ratio: 99.8%)
Message delivery delays	00" / 18'16" / 02'30" / 02'08" (*)	00" / 17'59" / 02'36" / 02'02"(*)
Distances covered during msg transfers (m.)	0.0 / 171.8 / 55.7 / 32.6(*)	0.4 / 169.7 / 54.4 / 33.0(*)

Table III
STATISTICS ABOUT A SMALL-SCALE FIELD EXPERIMENT CONDUCTED WITH DODWAN ON A UNIVERSITY CAMPUS.

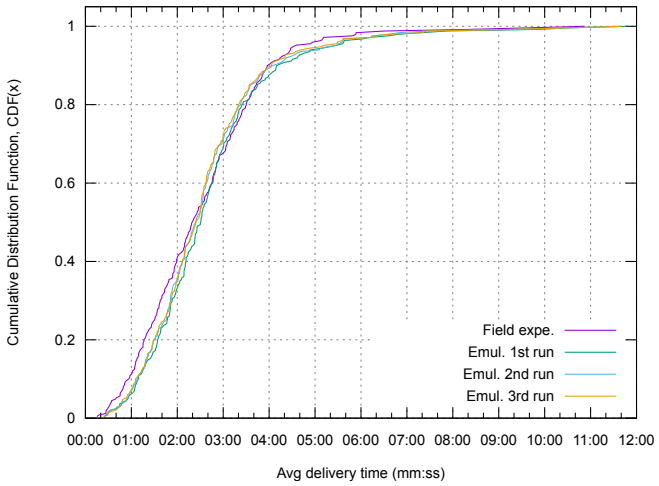

Figure 19. Cumulated distribution functions of message delivery delays observed in the field experiment, and when replaying the same scenario with LEPTON (three emulation runs). Note the similitude between the field experiment results and LEPTON's ones.

networking scenarios in such conditions constitute an excellent stress test for these systems. They have also shown that weaknesses in the system under test can be revealed while running demanding application scenarios in emulation mode, while they could perfectly remain undetected when running pure simulations, or when running on a handful of experimentation platforms.

Future work shall aim at extending the set of mobility models offered in LEPTON, as well as the set of complementary tools that facilitate the analysis of log files after an experiment.

ACKNOWLEDGMENTS

This work is partially supported by the Spanish MINECO grant TIN2014-55243-P and the Catalan AGAUR grant 2014SGR-691.

REFERENCES

[1] C. Boldrini, K. Lee, M. Önen, J. Ott, and E. Pagani, "Opportunistic Networks," *Computer Communications*, pp. 1–4, Mar. 2014.

[2] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," in *ACM Annual conference of the Special Interest Group on Data Communication (SIGCOMM 2003)*, pp. 27–34, ACM, Aug. 2003.

[3] M. M. Qirtas, Y. Faheem, and M. H. Rehmani, "Throwboxes in delay tolerant networks: A survey of placement strategies, buffering capacity, and mobility models," *Journal of Network and Computer Applications*, vol. 91, pp. 89–103, 2017.

[4] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET Simulation Studies: the Incredibles," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, pp. 50–61, Oct. 2005.

[5] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 619–632, 2012.

[6] S. Sharma, A. Hussain, and H. Saran, "Towards repeatability and verifiability in networking experiments: A stochastic framework," *Journal of Network and Computer Applications*, vol. 81, pp. 12 – 23, 2017.

[7] "ns-2 Network Simulator." http://nsnam.sourceforge.net/wiki. Accessed: 2018-02.

[8] "ns-3 Network Simulator." https://www.nsnam.org/. Accessed: 2018-02.

[9] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools'08)*, ICST, Mar. 2008.

[10] "QualNet Network Simulator." http://web.scalable-networks.com/qualnet-network-simulator. Accessed: 2018-02.

[11] "SteelCentral Riverbed Modeler." https://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html. Accessed: 2018-02.

[12] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," in *2nd International Conference on Simulation Tools and Techniques (SIMUTools'09)*, ICST, Mar. 2009.

[13] A. Roy, S. Bose, T. Acharya, and S. DasBit, "Social-based energy-aware multicasting in delay tolerant networks," *Journal of Network and Computer Applications*, vol. 87, pp. 169–184, 2017.

[14] "CRAWDAD: a Community Resource for Archiving Wireless Data At Dartmouth." http://www.crawdad.org. Accessed: 2018-02.

[15] E. Göktürk, "A Stance on Emulation and Testbeds, and a Survey of Network Emulators and Testbeds," in *21st European Conference on Modelling and Simulation (ECMS'07)*, ECMS, June 2007.

[16] H. Soroush, N. Banerjee, M. Corner, B. Levine, and B. Lynn, "A Retrospective Look at the UMass DOME Mobile Testbed (invited)," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 15, pp. 2–15, Oct. 2011.

[17] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liuand, and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-generation Wireless Network Protocols," in *IEEE Wireless Communications and Networking Conference (WCNC'05)*, pp. 1664–1669, IEEE, Mar. 2005.

[18] W. Kiess and M. Mauve, "A Survey on Real-world Implementations of Mobile Ad-hoc Networks," *Ad Hoc Networks*, vol. 5, pp. 324–339, Apr. 2007.

[19] R. Beuran, S. Miwa, and Y. ShinodaRazvan, "Network Emulation Testbed for DTN Applications and Protocols," in *32nd International Conference on Computer Communications (INFOCOM'13)*, pp. 3441–3446, IEEE, Apr. 2013.

[20] K. Maeda, K. Nakata, T. Umedu, H. Yamaguchi, K. Yasumoto, and T. Higashino, "Hybrid Testbed Enabling Run-time Operations for Wireless Applications," in *22nd Workshop on Principles of Advanced and*

Distributed Simulation (PADS'08), pp. 135–143, ACM/IEEE/SCS, June 2008.

[21] J. Li, P. Hui, D. Jin, and S. Chen, "Delay-Tolerant Network Protocol Testing and Evaluation," *IEEE Communications Magazine*, vol. 53, pp. 258–266, Jan. 2015.

[22] H. Li, H. Zhou, H. Zhang, B. F. Feng, and W. Shi, "EmuStack: An OpenStack-Based DTN Network Emulation Platform," *Mobile Information Systems*, vol. 2016, Oct. 2016.

[23] E. Giordano, L. Codecà, B. Geffon, G. Grassi, G. Pau, and M. Gerla, "MoViT: The Mobile Network Virtualized Testbed," in *9th ACM International Workshop on Vehicular Inter-networking, Systems, and Applications (VANET'12)*, pp. 3–12, ACM, 2012.

[24] J. Morgenroth, S. Schildt, and L. Wolf, "HYDRA: Virtualized Distributed Testbed for DTN Simulations," in *5th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH'10)*, pp. 71–78, ACM, 2010.

[25] J. Zhou, Z. Ji, and R. Bagrodia, "TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications," in *25th IEEE International Conference on Computer Communications (INFOCOM'06)*, IEEE, Apr. 2006.

[26] H. Yoon, J. Kim, M. Ott, and T. Rakotoarivelo, "Mobility Emulator for DTN and MANET Applications," in *4th ACM International Workshop on Experimental Evaluation and Characterization (WINTECH'09)*, pp. 51–58, ACM, 2009.

[27] G. Baudic, A. Auger, V. Ramiro, and L. Emmanuel, "HINT: from Network Characterization to Opportunistic Applications," in *11th Workshop on Challenged Networks (CHANTS'2016)*, ACM, Oct. 2016.

[28] "GraphStream: a Dynamic Graph Library." http://www.graphstream-project.org. Accessed: 2018-02.

[29] J. Liu, Y. Li, N. Van Vorst, S. Mann, and K. Hellman, "A real-time network simulation infrastructure based on OpenVPN," *Journal of Systems and Software*, vol. 82, no. 3, pp. 473–485, 2009.

[30] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-Varying Graphs and Dynamic Networks," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 5, pp. 387–408, 2012.

[31] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, "IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation," *Electronic Communications of the ECEASST*, vol. 37, 2011.

[32] G. Zhang, X. Cai, T. Han, and R. Xiao, "An implementation of DTN testbed based on DTN2," in *International Conference on Multimedia Technology (ICMT'11)*, pp. 3393–3396, IEEE, July 2011.

[33] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism," *Journal of Network and Computer Applications*, vol. 39, no. 1, pp. 319–333, 2014.

[34] J. Haillot and F. Guidec, "A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks," *Journal of Mobile Information Systems*, vol. 6, no. 2, pp. 123–154, 2010.

[35] J. Haillot and F. Guidec, "A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks," in *22nd International Conference on Advanced Information Networking and Applications (AINA'08)*, pp. 188–195, IEEE, Mar. 2008.

[36] Y. Mahéo, N. Le Sommer, P. Launay, F. Guidec, and M. Dragone, "Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs," in *4th Extreme Conference on Communication (ExtremeCom'12)*, pp. 1–6, ACM, Mar. 2012.

[37] A. Datta, S. Quarteroni, and K. Aberer, "Autonomous Gossiping: a Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Mobile Ad-Hoc Networks," in *1st International Conference on Semantics of a Networked World (ICSNW'04)*, no. 3226 in LNCS, pp. 126–143, Springer, June 2004.

[38] A. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," tech. rep., Duke University, Apr. 2000.

[39] J. Haillot, F. Guidec, S. Corlay, and J. Turbert, "Disruption-Tolerant Content-Driven Information Dissemination in Partially Connected Military Tactical Radio Networks," in *28th Military Communication Conference (MILCOM'2009)*, pp. 2326–2332, IEEE, Oct. 2009.

[40] K. Scott and S. Burleigh, "Bundle Protocol Specification." IETF RFC 5050, Nov. 2007.

[41] C. Borrego, S. Robles, A. Fabregues, and A. Sánchez-Carmona, "A Mobile Code Bundle Extension for Application-Defined Routing in Delay and Disruption Tolerant Networking," *Computer Networks*, vol. 87, pp. 59–77, 2015.

[42] S. Symington, "Delay-Tolerant Networking Metadata Extension Block." IETF RFC 6258 (Experimental), May 2011.

[43] S. T. Kouyoumdjieva, O. Helgason, and G. Karlsson, "CRAWDAD dataset kth/walkers (v. 2014-05-05)." http://crawdad.org/kth/walkers/20140505, May 2014. Accessed: 2018-02.

[44] "Legion - Science in Motion." http://www.legion.com. Accessed: 2018-02.

[45] O. Helgason, S. T. Kouyoumdjieva, and G. Karlsson, "Opportunistic Communication and Human Mobility," *IEEE Transactions on Mobile Computing*, vol. 13, pp. 1597–1610, July 2014.

[46] O. Helgason, S. T. Kouyoumdjieva, L. Pajević, E. A. Yavuz, and G. Karlsson, "A Middleware for Opportunistic Content Distribution," *Computer Networks*, vol. 107-2, pp. 178–193, Oct. 2016.