The Journal of Systems and Software 142 (2018) 57–72

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

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

Adrián Sánchez-Carmona[a,*], Frédéric Guidec[b], Pascale Launay[b], Yves Mahéo[b], Sergi Robles[a]

[a] Universitat Autònoma de Barcelona (UAB), Spain
[b] IRISA (UMR 6074), Université Bretagne Sud, France

ARTICLE INFO

Article history:
Received 13 September 2017
Revised 9 April 2018
Accepted 14 April 2018
Available online 20 April 2018

Keywords:
Distributed systems
Software evaluation
Opportunistic networking
Emulation systems
Software development process

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 to 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 pseudo-code that simulates its behavior) performs in controlled, repeatable conditions.

In this paper we present LEPTON, an emulation platform we developed, and we show how existing opportunistic networking systems can be adapted to run with this platform. 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.

1. 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 (Boldrini et al., 2014). In such conditions, the store, carry and forward principle of Delay Tolerant Networking (DTN Fall, 2003; Qirtas et al., 2017) 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 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 Fig. 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 that come from the real world (e.g., traces of real taxi cabs, 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 usually executed based on pseudo-code. This does not prove that the system being designed can eventually be deployed and used for real. Results obtained through simulation can be deceptive, creating a misleading feeling of scientific correctness. Indeed, as observed in Kurkowski et al. (2005), 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.

* Corresponding author.
E-mail address: adria.sanchez@deic.uab.cat (A. Sánchez-Carmona).

https://doi.org/10.1016/j.jss.2018.04.025
0164-1212/

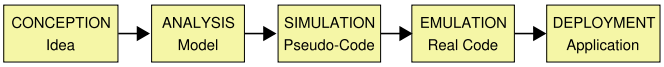

Fig. 1. Engineering process of an opportunistic networking system.

Testing real code in real conditions can be painstaking, or even impossible, especially when these real conditions involve the mobility (Zhu et al., 2012) of hundreds of nodes over hundreds 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. This stage is crucial to make sure that the code under test —with its bugs and limitations— is scalable, and that it can correctly integrate and interact with the other 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 to others at convenience (Sharma et al., 2017).

Briefly put, we could say that emulation is more apt than simulation to evaluate real code, and easier and cheaper than full-scale field 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 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 the description of LEPTON, our lightweight emulation platform for opportunistic networking. In order to show its usefulness, we also use it to compare two existing Opportunistic Networking (OppNet) middleware systems, using the same scenario and mobility traces.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. In Section 3 we provide an overview of LEPTON and of its salient features. In Section 4 we present briefly the two OppNet 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 5, and Section 6 concludes this paper.

2. 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 (ns-2, 2018), ns-3 (ns-3, 2018), OMNeT++ (Varga and Hornig, 2008), QualNet (QualNet, 2018) or Riverbed Modeler (SteelCentral, 2018) 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 simulator ONE (Keränen et al., 2009; Roy et al., 2017) 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 (CRAWDAD, 2018) 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 in such conditions 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 using contact or mobility traces instead of pure algorithmic models, but these traces have often been captured in very specific conditions (e.g., people moving around in a conference building, taxi cabs roaming city streets, etc.) Radio channel modeling is also debatable, as models 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. With 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 (Göktürk, 2007). 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 (Soroush et al., 2011). Some general-purpose large-scale network testbeds such as ORBIT (Raychaudhuri et al., 2005) 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 con-