ns-3 Module for Routing and Congestion Control Studies in Mobile Opportunistic DTNs

Jani Lakkakorpi and Philip Ginzboorg
Aalto University, Comnet, Finland
Email: {jani.lakkakorpi, philip.ginzboorg}@aalto.fi

***Abstract*—We present a delay-tolerant networking (DTN) module for ns-3 simulator that can be used, e.g., in DTN routing and congestion control studies. We have validated the code by replicating our extensive DTN congestion control simulations that we performed last year with ns-2. New simulation results show that our ns-3 DTN code works as intended. However, the results are not identical, due to implementation differences in the two DTN modules.**

***Keywords*—DTN, congestion control, routing, ns-3, opportunistic network**

I. Introduction

The DTN [2] architecture introduces a bundle protocol [14], which offers transport services for applications. The bundles are usually much larger than IP packets. This allows creation of self-contained messages that enable complete application interactions with a single message exchange. The architecture is designed for networks where an end-to-end path may not exist, i.e., the bundles are transmitted based on hop-by-hop reliability. Any suitable convergence layer (such as TCP or UDP) can be used for transmitting a bundle over a single hop. Bundle has a lifetime, after which the bundle expires and is removed from the network.

A number of different routing protocols have been proposed for DTNs. Most of the routing protocols create multiple copies of the bundle in order to increase the probability of reaching the destination. Epidemic routing [17] does not limit the number of messages: a message is copied to all nodes that do not yet have a copy. This kind of flooding has been shown to cause congestion and message buffer overflows, which decreases message delivery probability. Spray and wait routing, on the other hand, [15] explicitly limits the number of bundle copies to a fixed value. We have implemented epidemic routing and spray and wait for ns-3.

Independent of the DTN routing protocol, the destination node may generate a return receipt to the source node upon bundle reception. These return receipts can also act as antipackets as in VACCINE mechanism [6], which deletes the bundles from the message buffers and gives immunity to nodes upon seeing a return receipt. Antipackets are always sent using epidemic routing [18] and they are removed from the network as they eventually expire. In our code, antipackets are used by default.

The rest of this paper is organized as follows. Section 2 presents our DTN module for ns-3. Section 3 shows the results of simulating the same DTN scenarios obtained with ns-3 and ns-2, highlighting their differences. Finally, Section 4 concludes the paper with a discussion.

II. Simulation Model

We have added several extensions to the *ns-3.16* simulator [3] that allows us to study DTN routing and congestion control together with accurate models for radio links. Our simulations build on top of (1) synthetically generated random waypoint (RWP) model, (2) real-world vehicular traces, and (3) more realistic yet synthetic pedestrian mobility model.

A. Traffic and Mobility Models

We apply a simple traffic model in which each node sends a variable size message at a random time with 200 second intervals $[t, t+200s]$ to another, randomly selected, node. Before sending to MAC layer, the bundles are fragmented into 1500-byte datagrams. A retransmission mechanism, providing reliable delivery of datagrams is implemented, so that messages will not be lost due to transmission errors. In our code, this can be accomplished by using UDP with retransmissions or having TCP as bundle transport protocol. However, TCP may consume a lot of memory in the simulating machine, because there could be hundreds of simultaneous TCP connections. Their number (and thus the memory consumed) depends on the routing algorithm, traffic patterns, etc.

(1) RWP is our first mobility model. This model is well understood and it is easy to generate scenarios with different network densities and node velocities. In our ns-2 [13] DTN model [11], we generated random waypoint node mobility using the *setdest* program. In ns-3, no additional programs are needed for this. We have 40 mobile nodes that select a random direction and a random (uniformly distributed) speed at random times. Maximum speed is 20 m/s and pause length is two seconds. We always pause before choosing a new direction and a new speed. In our RWP scenarios, area size ranges from 10 m $\times$ 10 m to 2000 m $\times$ 2000 m, and the simulation time is always 5000 seconds.

(2) ns-3 is able to read ns-2 trace files as such and thus we can use exactly the same trace files as in [12]. In both of our trace-based mobility models the number of mobile nodes is 116 and the simulation time is 3600 seconds. In the first mobility trace, the San Francisco taxi cab trace [1], the area size is 5700 m $\times$ 6600 m. The whole data set contains GPS coordinates of approximately 500 taxis collected over 30 days in the San Francisco Bay area.

(3) Another mobility trace was obtained in a synthetic fashion: the map of Helsinki city center (area dimensions: 4500 m $\times$ 3400 m) is used as input for the ONE simulator [9] and the nodes are configured to move between selected points of interest. Node velocity is uniformly distributed between

0.7 m/s and 1.4 m/s and pause length between 0 and 120 s. This model provides a denser network scenario in comparison to the aforementioned taxi cab scenario.

B. The DTN code

The DTN simulation code is available from [10]. It is implemented in `dtn.cc`, `mypacket.cc` and `mypacket.h` files. The first file includes the code for all DTN node functions as well as the 'simulation script' that creates the network and sets the parameters. The two latter files implement bundle and antipacket headers. Header sizes are 28 bytes for bundles and 26 bytes for antipackets.

Discovery mechanism. In our implementation DTN nodes advertise their buffer content to each other every 100 ms by sending Hello messages. We have limited Hello message size to 2280 bytes. If there is not enough room for all identifiers of buffered bundles and return receipts, the return receipt identifiers are dropped first. In `SendHello` function, we broadcast a Hello message that contains the identifiers of those bundles and return receipts that are stored in this node as well as advertised free buffer capacity of the node. The size of the Hello message depends on the number of items stored in the node. Hello messages have forwarding priority over all other messages; in our code this is implemented with quality of service on MAC level. `ReceiveHello` function receives Hello messages and updates neighbor information, i.e., what bundles the neighbors currently store in their buffers.

Routing and congestion control. A bundle can be generated only if the node has sufficient amount of free buffer space. Moreover, if DTN congestion control (see [12]) is enabled, a bundle can be forwarded only to such intermediate nodes that have enough free buffer space. DTN congestion control is independent of the selected DTN routing algorithm. The routing algorithm is either epidemic routing or binary spray and wait. In the latter, the default number of forwarding tokens is 16, i.e., a bundle can be forwarded to a maximum of 16 nodes. If the sender does not receive a return receipt within retransmission timeout, it will retransmit the bundle. In our code, the number of bundle retransmissions is hard-coded to three. Return receipts also serve as antipackets [6]; their lifetime is the minimum of retransmission timeout (1000 seconds) less bundle forwarding time and bundle lifetime (750 seconds). Antipacket size is 10 bytes without headers.

Buffer management. In `CheckBuffers` function we periodically check what bundles and return receipts can be forwarded. Here we also remove expired bundles and return receipts from the storage. If the MAC queue occupancy is below a given threshold (currently hard coded to two packets), we select the next packet to be transmitted. Return receipts are forwarded first. When the head-of-line receipt has been forwarded to all current neighbors (`SendAP` function is called to send a given antipacket to a given neighbor), one by one, we put that return receipt to the tail of the return receipt queue and dequeue the next return receipt. Then, we forward regular bundles to their destinations (if they are within radio range), in a similar manner as return receipts (`SendBundle` function is called to send a given bundle to a given neighbor). After this, the bundle queue is re-ordered so that the least forwarded bundles are put to the head of the queue. Finally, regular bundles are forwarded to neighboring, non-destination nodes.

Transmitting bundles. Both UDP and TCP can be used for bundle transmission. In the UDP case, acknowledgements (one ack per packet) and retransmissions (retransmission timeout: 1.0 seconds) are applied. With UDP-based transmission, we check before sending each packet if the neighbor is still within transmission range. If that is not the case, we shall continue sending when we are close enough.

Receiving bundles. Bundle reception is handled in `ReceiveBundle` function. Here we first check if all packets of a bundle have been received. If the answer is yes, then we forward, receive or delete the bundle. Tail dropping is applied if bundle storage is full. `ReceiveAP` function receives antipackets (i.e., return receipts) and processes them accordingly: the corresponding bundles are deleted from bundle storage and their identifiers are added to the database so that further bundle copies can be deleted upon their arrival.

C. Wireless Channel Model

Our DTN code can be used with any wireless model found in ns-3. However, in order to validate our code, we chose to use the wireless PHY and MAC parameters listed in Table 1. Some of the parameters are set using the wifi.SetStandard (WIFI_PHY_STANDARD_80211g) command. In the ns-3 802.11g model, there is no support for short slot time. Otherwise, the parameters are almost the same as in [12]. The selected parameters lead to radio range of 130 m for each node.

TABLE I: IEEE 802.11g simulation parameters.

Parameter	Value
RxNoiseFigure	7
TxPowerLevels	1
TxPowerStart/TxPowerEnd	12.5 dBm
m_channelStartingFrequency	2407 MHz
TxGain/RxGain	1.0
EnergyDetectionThreshold	-74.5 dBm
CcaMode1Threshold	-77.5 dBm
RTSThreshold	0 B
CWMin	15
CWMax	1023
ShortRetryLimit	7
LongRetryLimit	7
SlotTime	20 μs
SIFS	10 μs

III. Simulation Results

A. Random Waypoint Mobility

Figure 1 compares routing performance in the random waypoint scenario, showing plain epidemic, spray and wait, and epidemic routing enhanced with congestion control (CC). Node buffer sizes are randomly set to 250 KB or 2 MB so that both occur with equal probability. The size of the individual bundles is uniformly distributed between 1 Byte and 20 KB, and the static congestion threshold (T_C) is set to 0.8. The top row contains ns-3 results while the bottom row contains ns-2 results.

As expected, the results are qualitatively very close to the ones reported in [12] (reprinted here, in the bottom row): a

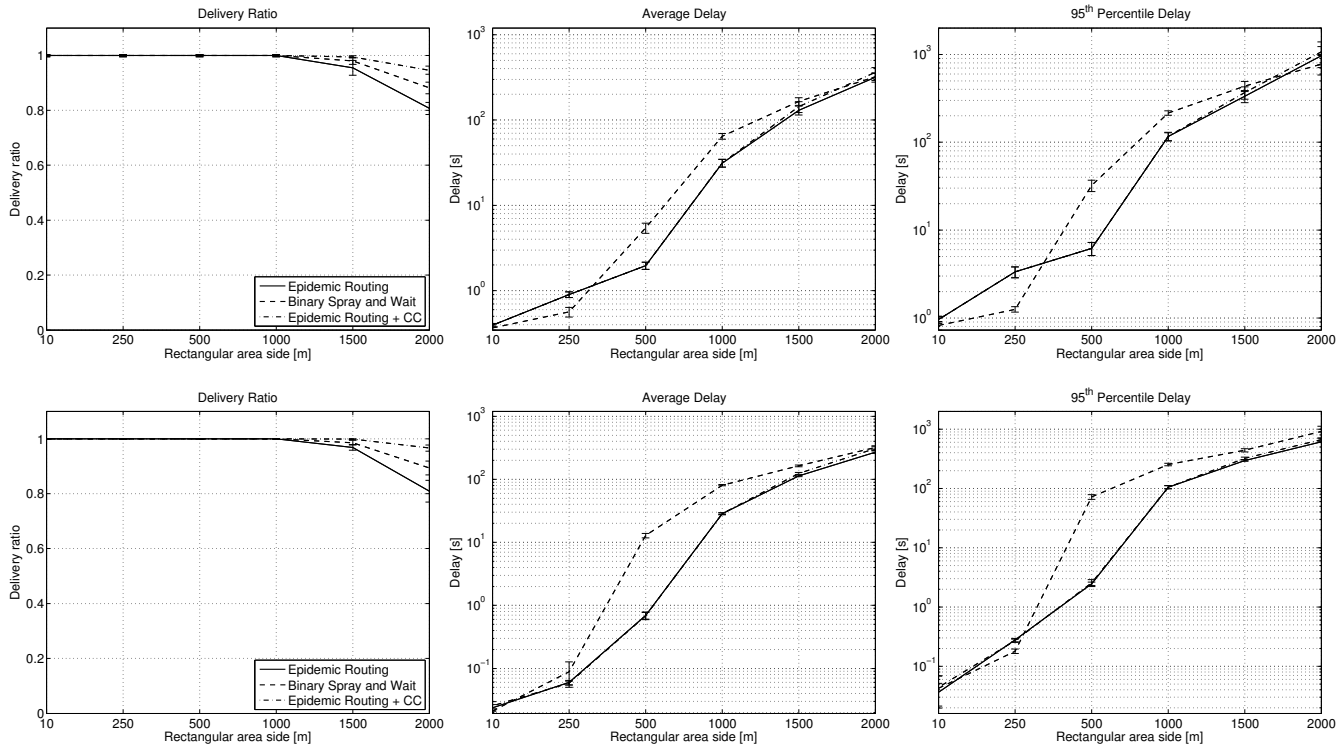

Fig. 1: Heterogeneous nodes, RWP mobility. ns-3 results: top, ns-2 results: bottom.

significant improvement in message delivery ratio is gained by applying congestion control or explicitly limiting the number of copies per message (spray and wait) when the network is sparse (1500 m $\times$ 1500 m and up). However, delays are systematically larger with ns-2. This happens because the simulation model details are not identical. We can also observe that with ns-3, epidemic routing leads to higher delays than spray and wait when the network is dense (250 m $\times$ 250 m). The likely reason this does not happen in ns-2 simulations is the less accurate Hello message implementation compared to ns-3. In ns-2 simulations, Hello messages have a fixed size; in ns-3 they carry bundle identifiers and get bigger when the network is dense, because then the nodes have more neighbors.

B. Trace-based Mobility

Figures 2 and 3 illustrate the performance evaluation with more realistic conditions. Figure 2 shows performance with the San Francisco cab trace scenario and Figure 3 with pedestrian mobility created using the Helsinki City scenario. In both cases, the message size is uniformly distributed and varies between 10 KB and 100 KB with 10 KB granularity. In the San Francisco case, the node buffer size is either 1.375 MB or 11 MB (both occur with 50% probability) whereas in the Helsinki case the node buffer size is always 6 MB. The average node buffer size is almost the same in these two different scenarios.

The results of the two simulation scenarios are quite similar. With both mobility models, using congestion control leads to better delivery ratios. But this gain is marginal in the more dense Helsinki city scenario and it seems that bandwidth is a bottleneck too. We can also see that the use of TCP as transport protocol (ns-3 results only[1]) leads to decreased bundle delivery ratio. The likely reason is that upon antipacket reception the on-going TCP connections related to a given bundle finish their file transfers, rather than being stopped and deleted.

There are some differences between ns-2 and ns-3 results. In the cab trace scenario, spray and wait enhanced with congestion control gives better delivery ratio with ns-2. Moreover, pure epidemic routing gives worse delivery ratio with ns-2. In the Helsinki city scenario, spray and wait gives better delivery ratio with ns-2 and epidemic routing (with congestion control, too) gives worse delivery ratio with ns-2.

The mobility traces and traffic patterns (sending of bundles) are identical in our ns-2 and ns-3 simulations[2]. Even though the wireless models used in different simulators are somewhat different[3], both models result in a radio range of 130 meters. Another difference is the greater realism of Hello messages in our ns-3 DTN code. In ns-2, the Hello messages had a fixed size, while in ns-3 the size of the Hello messages depends on the number of bundle identifiers they carry.

[1] In ns-2 results, which we replicated here from [12], RR stands for retiring replicants [16]. The comparison of our congestion control algorithm and RR is outside the scope of this paper.

[2] The problem in importing ns-2 trace files to ns-3 [8] has been solved.

[3] In our ns-2 simulations, we used a wireless channel model from the *dei80211mr* library (not implemented in ns-3); in our ns-3 simulations, *IdealWifiManager* is used. The trace files (e.g., packet error tables) in these models are not identical.

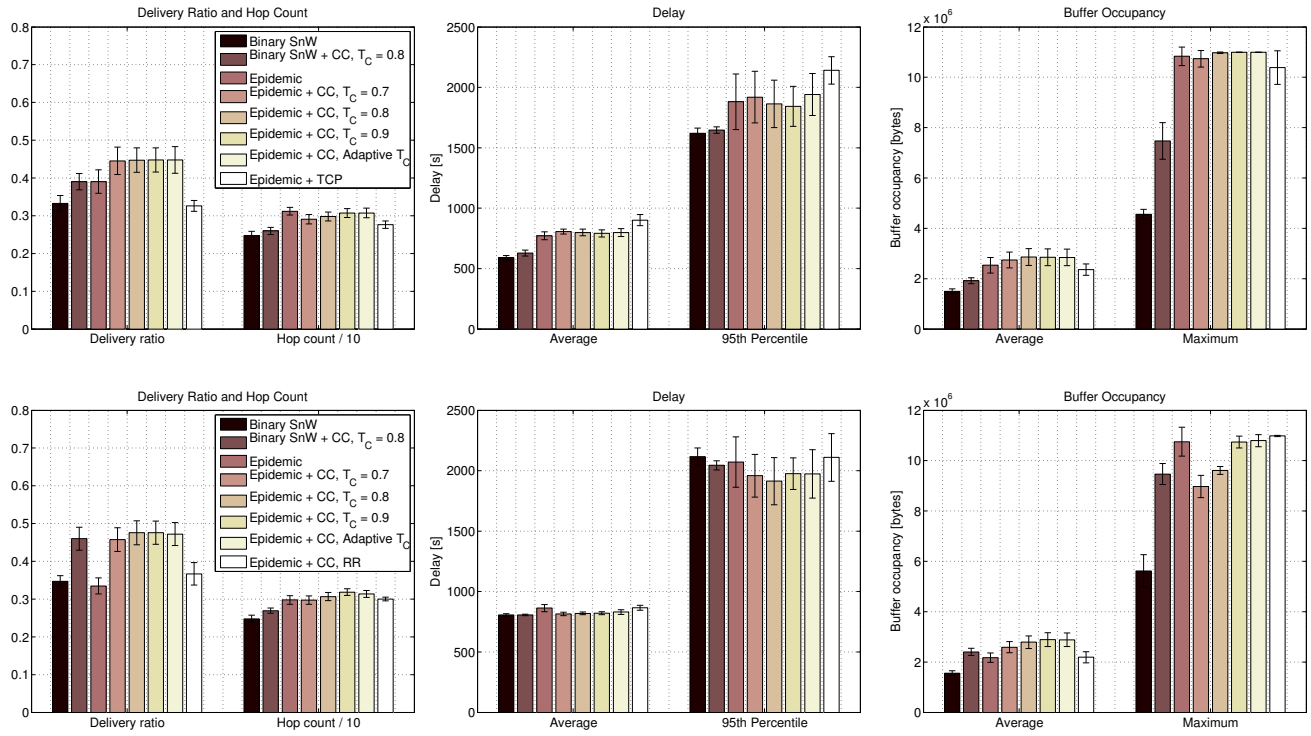

Fig. 2: Real mobility trace. ns-3 results: top, ns-2 results: bottom.

IV. Conclusions and Discussion

We have presented a new DTN module for ns-3 simulator and validated the code by re-running our ns-2 simulation scenarios for simple congestion control mechanism for mobile opportunistic networks. Our new simulation results using the random waypoint model as well as real-life and synthetic mobility traces show that our ns-3 DTN code works as intended. However, due to differences in the two DTN modules (e.g., the more accurate Hello message modeling in ns-3), the results are not identical.

We will end this paper with a few comments based on our experience with the ns-2 and the ns-3 simulators.

• ns-2 is written in C++ and Otcl while ns-3 is written in C++ and Python. In ns-2, the simulation scripts are written in Otcl while the 'building blocks' are implemented in C++. In ns-3, both the 'building blocks' and the simulation scripts can be written in C++. In theory, the simulation scripts could be written in Python, too.

• ns-2 is not as realistic as ns-3. For example, the packet size is given by the user and when new packet header fields are added packet size does not automatically increase. In ns-3, packet header fields affect packet size. (The user-defined tags in ns-3 can convey information inside packets without increasing the packet size.)

• It is often claimed that one advantage of ns-3 over ns-2 is a real-time *emulation* package that allows connecting ns-3 to real networks and that ns-3 code could be better utilized in real network equipment. But ns-2 also supports emulation, although this feature has not been very popular. Moreover, in the latest ns-2 releases, Linux source code for different TCP congestion control algorithms can be used as such – and naturally vice versa.

• *Scalability* has been reported as a problem in ns-2 [7] while its successor, ns-3, is claimed to be more scalable. However, some scalability issues, like the inability to delete TCP connections that have not finished their transmission in time, still exist in ns-3. The aforementioned feature (or bug) increases memory consumption greatly if the simulation script involves dynamic creation of TCP connections in a congested environment. It should be noted, however, that this is not different from ns-2. In general, ns-3 code is faster and consumes less memory than ns-2 code.

• ns-2 is not updated anymore except for maintenance releases. Latest release was ns-2.35 in November 2011. In contrast, ns-3 has periodic releases with new features every three to four months. Latest release was ns-3.16 in December 2012. As ns-2 code is pretty stable, most bugs have already been found and fixed. ns-3 is still work in progress and the bug fixes in new releases usually affect simulation results.

• ns-2 *user community* is still large, but its mailing list (ns-users) is not so active anymore and most questions are left unanswered. This may be a sign that users' interest in ns-2 is diminishing. ns-3 community is still rather small but will likely continue growing. (There were 86,014 downloads of the software in 2011 [3]). Moreover, ns-3 mailing list is active and it might help many users. In ns-2, some features are still not properly documented while ns-3 features are typically documented well.

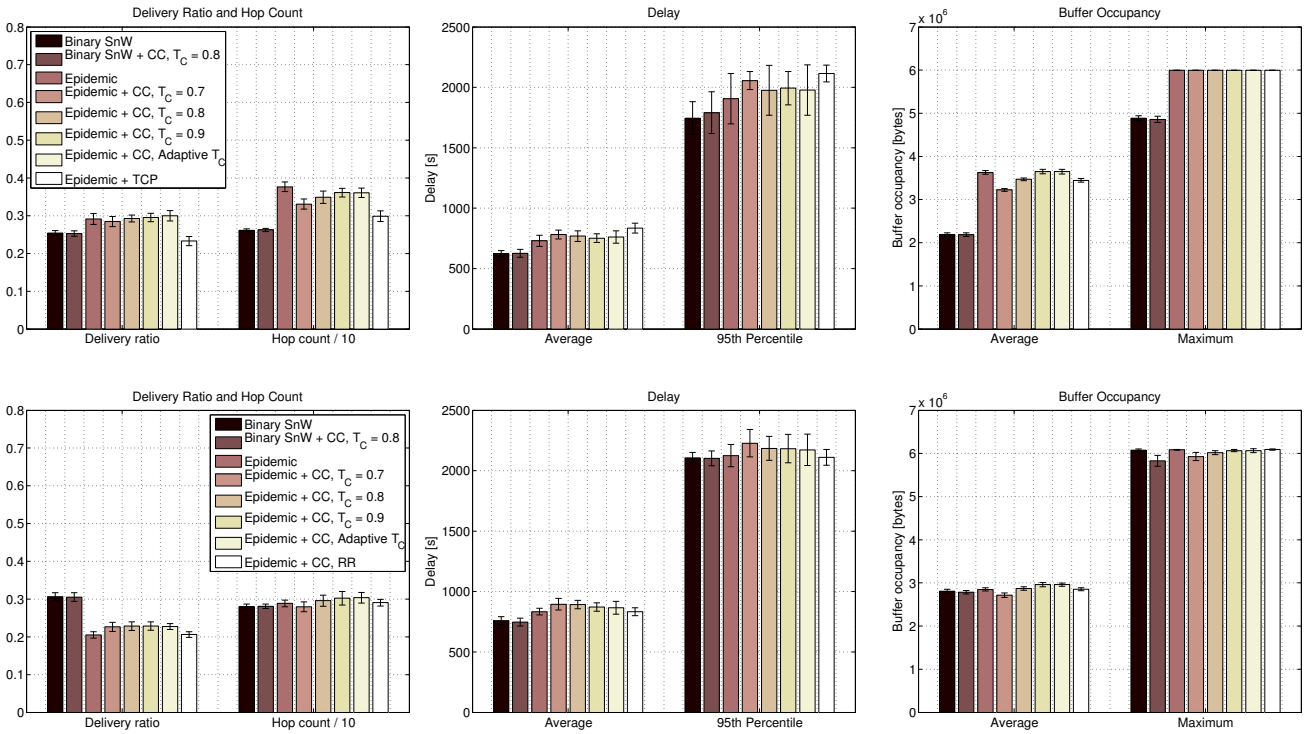

Fig. 3: Synthetic mobility trace. ns-3 results: top, ns-2 results: bottom.

Transition from ns-2 to ns-3 can be difficult, especially if one has no experience with network programming, sockets, etc. Things can be kept more abstract in ns-2. In our opinion, ns-2 experts will not switch to ns-3 easily, while people without previous experience from ns-2 might find the active ns-3 community more appealing. In addition, it remains to be seen whether ns-3 is too real-life like for people with more theoretical (queueing/teletraffic theory) background. These people, if they are not using ns-2, might prefer other tools such as the commercial OPNET Modeler [5] or the free OMNeT++ [4] over ns-3.

ACKNOWLEDGMENTS

This work received funding from the Academy of Finland in the RESMAN project (grant no. 134363) and from the European Community's Seventh Framework Programme under grant agreement no. 258414 (SCAMPI).

REFERENCES

[1] Cabspotting. http://cabspotting.org.

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

[3] Network Simulator - ns (version 3). http://www.nsnam.org.

[4] OMNeT++. http://www.omnetpp.org.

[5] OPNET Modeler. http://www.opnet.com.

[6] Z. Haas and T. Small. A new networking model for biological applications of ad hoc sensor networks. *IEEE/ACM Trans. Netw.*, 14(1):27–40, February 2006.

[7] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM.

[8] N. Kamoltham, K. Nakorn, and K. Rojviboonchai. From ns-2 to ns-3 - implementation and evaluation. In *Computing, Communications and Applications Conference (ComComAp), 2012*, pages 35 –40, jan. 2012.

[9] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09*, Rome, Italy, March 2009.

[10] J. Lakkakorpi. DTN Code for ns-2 and ns-3. https://wiki.aalto.fi/display/~jlakkako@aalto.fi/DTN+Code+for+ns-2+and+ns-3.

[11] J. Lakkakorpi, M. Pitkänen, and J. Ott. Adaptive routing in mobile opportunistic networks. In *ACM MSWIM'10*, pages 101–109, Bodrum, Turkey, October 2010.

[12] J. Lakkakorpi, M. Pitkänen, and J. Ott. Using buffer space advertisements to avoid congestion in mobile opportunistic dtns. In *WWIC*, Barcelona, Spain, June 2011.

[13] S. McCanne and S. Floyd. Network Simulator - ns (version 2). http://www.isi.edu/nsnam/ns.

[14] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC Experimental 5050, IETF, November 2007.

[15] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking*, pages 252–259, Philadelphia, PA, USA, August 2005.

[16] N. Thompson, S. Nelson, M. Bakht, T. Abdelzaher, and R. Kravets. Retiring replicants: congestion control for intermittently-connected networks. In *IEEE INFOCOM*, pages 1118–1126, San Diego, California, USA, March 2010.

[17] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical report, Duke University, April 2000.

[18] B. Walker, J. Glenn, and T. Clancy. Analysis of simple counting protocols for delay-tolerant networks. In *CHANTS'07*, pages 19–26, Montréal, Québec, Canada, September 2007.