Delay-Tolerant Networks with Network Coding: How Well Can We Simulate Real Devices?

Yuanzhu Chen, Xu Liu
Wireless Networking and Mobile Computing Laboratory
Department of Computer Science
Memorial University of Newfoundland
Canada

Walter Taylor and Jason H. Moore
Computational Genetics Laboratory
Giesel School of Medicine
Dartmouth College
USA

Abstract—**Delay-tolerant networking effectively extends the network connectivity in the time domain, and endows communications devices with enhanced data transfer capabilities. Network coding on the other hand enables us to approach the information capacity of networks by allowing intermediate nodes to process data en route. Both of these were major principal breakthroughs in mobile and wireless communications in the past decade or so. As reported in this article, we are interested in how network coding battles such challenged networks as DTN from an experimental perspective. We conducted tests with both real smart mobile devices and computer simulation and found conditions where their results match. This would give us confidence of using computer simulation to study larger delay-tolerant networks with and without network coding at a much manageable cost.**

I. Introduction

Data communication networks connect computing devices with wired or wireless links to exchange information. For such networks to scale as the number of devices in it increases, we allow messages to traverse multiple communication links from its source node to the destination. Such a "store and forward" technique is the central idea of how the Internet can support numerous of computers. This effectively extends the scope of communication networks spatially. Recently, research on Delay-Tolerant Networking (DTN) [1], [10], [12], [23] has been exploring how to extend communication networks temporally. As mobile devices and networking technologies become more powerful and efficient, such mobile devices can be used to "store, carry, and forward" data when users roam about. That is, even without cellular or Wi-Fi infrastructure and only relying on short-range radios, e.g. Bluetooth and ZigBee, a set of sparsely deployed mobile devices can be used to transfer data automatically especially when the data are not required to be time-sensitive. The DTN technology can be useful in many scenarios, such as mobile sensor networks, disaster recovery, social networking, etc.

The concept of network coding was formulated in the seminal work of Ahlswede, Cai, Li, and Yeung [5] in 2000, and the past decade has seen a tremendous momentum in this area [17]. Its idea breaks away from the principal of traditional multi-hop networking, where intermediate nodes only forward packets but do not modify their contents, much like cars traveling on a highway. Since bits are not cars anyway, network coding allows intermediate nodes to combine packets from different input ports before forwarding them. When treating a packet as a sequence of symbols, even linear network coding defined over small Galois fields can introduce a fairly significant throughput gain. The readers are referred to an easy-to-read and yet informative primer by Fragouli, Le Boudec, and Widmer [9]. Other benefits of network coding include improved robustness of network operations, higher energy efficiency in wireless radios, and better security against eavesdroppers. Network coding proves to be especially powerful and flexible, and can be exercised along with other revolutionary networking paradigms. For example, it was shown that opportunistic data forwarding in multi-hop wireless networks can further increase the capacity of these networks when intermediate nodes strategically combine overheard packets and forward them [6], [21]. As another example, the resilience to lost or delayed information brought about by network coding turns out particularly effective in DTNs, as evidenced by computer-simulated experiments in Widmer and Le Boudec [20] and in Lin, Li, and Liang [13].

In this work, we evaluate how network coding stacks against conventional message passing in DTNs using both real iOS devices and in the ONE simulator in a university building. Our goal is to assess to what extent the ONE as one of the best and most widely used simulators for DTN research can mimic the real world. On one hand, we used real Apple iOS devices to measure how message propagate among roaming users over the built-in Bluetooth radios. On the other hand, we enhanced the ONE with a more realistic link layer by adding a few parameters. We are able to claim that the simulator work fairly closely to iOS devices with these parameters tuned properly. As part of a bigger research project, we may be confident that the simulator can work in place of real devices for efficient studies of larger-scale networks.

The rest of this article is organized as follows. Next section, we provide background in some relevant experimental research on DTN and network coding in this context. In Section III, we describe how message passing is done with and without network coding. We conducted experiments using both real devices and computer simulation. The experimental settings and results are reported in Section IV. Section V concludes this article with discussion and future research issues.

II. Background

While researchers in data communications have been on a quest for the ultimate killer applications of the DTN technology [15], there have been a number of small but interesting experimental applications in infrastructureless computer networking, such as IPN [3], Haggle [2], ZebraNet [11], DieselNet [22], and iSNAC [7], to name a few. Among these, iSNAC is a mobile social networking iPad application that focuses on broadcasting messages to help conference attendees to share information effectively.

Network coding is shown in Widmer and Le Boudec [20] and in Lin, Li, and Liang [13] to be very effective in battling the intermitted connectivity in DTNs while assuming neither do nodes have information about future encounters nor are they able to derive this information from past history. They use a custom computer simulator to compare the packet delivery ratio and delay between non-network-coding DTN forwarding and using random network coding. Their simulation indicates that network coding outperforms DTN forwarding for various network density levels and mobility patterns even when the latter is allowed to use very intelligent beaconing. In addition to experimental results, the latter [13] also provides some nice performance bounds analytically.

In our experiments, we used the Bluetooth radios on Apple iOS devices to form a DTN. Apple provides the Bluetooth communication capabilities through a set of wrapping classes in the GameKit Framework. Although this framework was originally designed for infrastructureless peer-to-peer gaming, it is essentially an API for devices to send generic data among one another over Bluetooth radios. The API provides reliable and unreliable link layer data services between one-to-one and one-to-many devices. We picked the unreliable, one-to-many variant in our experiments. Each invocation of data transmission is allowed to send up to 90KBytes of payload. If the upper layer modules need to send more data, they must first be segmented into smaller chunks. The API was not known to handle rapid connections and disconnections well, so there can be delay in a device detecting another device coming into range, and links may break occasionally when they are busy. Nevertheless, these characteristics of GameKit provides us a perfect vehicle to study the effect of network coding in such an extreme and yet practical scenario.

The ONE (Opportunistic Network Environment) [4] is a discrete-event computer simulator for DTN research. It was written in Java and made open-source for the research community. The ONE implements a number of well known message passing methods in DTN, such as Epidemic Routing [19], Spray and Wait [18], PROPHET [14], etc. It has a large number of hooks to customize behaviors of these protocols and for us to add network coding to it. The simulator supports mobility in a topological structure in addition to traditional free-space 2D mobility. This allows us to specify how users move around in an experiment area easily. The ONE uses a simple link layer model with a binary circular transmission area. Given a set transmission radius, two nodes have reliable communication if they are within range; otherwise, they cannot hear from each other. Apparently, such an idealized link layer model would be unrealistic. We modified the link layer by adding delays in connection establishment and probabilistic link breaks with parameters set similarly to what we saw in the GameKit so that the ONE could faithfully simulate iOS devices.

III. Store, carry, and forward/code

In this work, we are interested in the problem of disseminating messages from a particular source to all other nodes in the network. We focus on a DTN, where a set of sparsely deployed nodes roam around without assuming any predictability. Each node periodically injects a message into the network intended to all other nodes. The strategies that a node employs depend on whether network coding is used. Without network coding, two nodes transfer messages via a handshake protocol. With network coding, a node simply randomly mixes the packets it has received so far and send these random combinations to an encountered peer.

A. *Message prioritization without network coding*

When not using network coding, we take a similar approach to Epidemic Routing [19] in that, when two nodes come into range of each other, one node can transfer a number of messages to the other via a 3-way handshake sequence. However, the difference here is that we must pick thoughtfully which messages to include in a short advertisement packet such that the system has a high overall throughput. Specifically, when node A discovers node B in its transmission range, it sends a *summary vector* SV_A. In the original Epidemic Routing, SV_A contains the unique IDs of all the message that A stores in its knowledge base. In our solution, it needs to be a small subset of these messages because there can be many of them after the network has been up running for some time. After receiving SV_A, node B replies with $SV_A - M_B$, where M_B is the set of all messages stored at node B. As such, node B essentially tells A which messages from A would potentially enrich B's knowledge of messages. Next, node A retrieves messages in $SV_A - M_B$ from its storage and sends them to B in a burst to complete the handshake. If the two nodes are still within range τ seconds after the handshake, they will start another round of handshake to transfer more messages. Similarly, node B would do the same to its neighbors periodically.

Given that nodes typically store more messages than that can fit in a single handshake packet, the strategy taken as of which messages should be in the advertisement and in what order affects the network performance significantly. We call such a strategy *message prioritization*, and some preliminary research on this scheme is reported in our previous article [16]. To reiterate, we are interested in a few simple, and yet very different such methods. In all methods, we assume that node A fills the advertisement packet with l digests ($l = 10$ in our experiments) created out of some of its stored messages.

1) Round robin — Node A maintains a FIFO queue of the messages it has received and generated so far, i.e. by the

time it is injected into the network. It circulates through the queue to compile the message digests using a pointer. When it is about to initiate a handshake, it processes l messages and advance the pointer accordingly. Here, the node maintains separate pointers for different nodes. Note that as the network continues to operate, the time it takes to finish a round becomes longer, and when it does, it starts from the head of the queue again.

2) Tiered — Messages stored at a node are ranked according to three quantities to favor new, short messages, i.e. forward history, age, and length, in a decreasing order of significance. The fewer times it has been forwarded till reaching this node A, the later it was created by its origin, and the shorter its payload is, it is ranked higher in the storage queue. These ranked messages are split into three segments of about the same number of messages, the upper, middle, and lower tiers. Three separate round-robin schedules are executed on the tiers. The upper tier has three opportunities to send an advertisement containing l digests of its own, the middle tier has two, and the lower tier has one. Thus, the system helps newly injected message spread in the network more quickly.
3) Oblivious — Node A maintains a FIFO queue of all messages by the time they are injected into the network as in Round robin. When the node needs to create an advertisement packet, it simply takes the last l messages in the queue. In this method, the node never looks back after it has past a message in the queue, thus, always rigidly favoring the latest messages in the network.

In addition to the three above, we also implemented a *random* prioritization method as a comparison baseline, where node A would randomly pick l messages and advertise their digests. All four methods are essentially different ways to allocate opportunities to messages to be advertised in the network.

B. With network coding

In random network coding, when a set of packets are combined and sent by an intermediate node, both the combined message (i.e. the *information vector*) and how they are combined (i.e. the *encoding vector*) are to be included as being transmitted [9]. The dimensionality of the information vector is simply the number of symbols in the message content, say M, while the dimensionality of the encoding vector, denoted m, is the maximum number of original messages that can be combined. Apparently, when a packet is sent, $m + M$ symbols are transmitted. The greater m is, a higher percentage of communication capacity is consumed by such a coding overhead. When a packet is received by a node, it is inserted in its decoding matrix. Depending on whether the packet is innovative, the rank of the decoding matrix may or may not increase by 1. In any case, the rank of the matrix can be up to m, and is usually in that ballpark in a stable network. Because matrix operations on general-purpose processors can be expensive, a large value of m also implies a large computational overhead. Thus, for practical purposes, we can divide packets into non-overlapping *generations* and only allow packets of the same generations to be combined as in Chou, Wu, and Jain [8]. By tuning the size of a generation, we can control the communication and computation overhead. Widmer and Le Boudec [20] show that the generation size is a crucial parameter for the performance in their simulated studies of DTNs.

In this research, we set the generation size $G = 50$ globally in our tests. Provided we have $n = 10$ devices, every device contributes 5 messages to each generation. Specifically at any given time, for device i ($i = 1, 2, \ldots, n$), its jth message ($j = 1, 2, \ldots, g$ for some latest generation g) belongs to generation $\lceil j/5 \rceil$ of the network. In addition, this message takes dimension $i \times (n-1) + (j \bmod 5)$ in that generation. During the operation of the network, a node would have generated and received packets of various generations. We use P_k to denote the set of packets of generation k, where $k = 1, 2, \ldots, g$. Thus, collectively, we use $\mathcal{P} = \{P_1, P_2, P_3, \ldots, P_g\}$ to denote the generations of packets stored at said node. When a node is within range of any peer, it periodically (every $\tau = 15$ seconds in our experiments) generates a set of random combinations of the packets it has received so far and broadcasts them to its neighbors. These packets are generated as follows. For generation k ($k = 1, 2, 3, \ldots, g$), it creates $\max\left\{w \times \frac{R_{P_k}}{2^{g-k}}, 1\right\}$ random combinations of all packets in this generation, where w is a parameter to control the overall load on the radio, and R_{P_k} is the rank of the decoding matrix corresponding to the kth-generation packets P_k. That is, each generation contributes at least one random packet combination. In addition, when $w = 1$, the latest generation g contributes random combinations of at most its rank, the second latest generation $g - 1$ contributes up to half of its rank, generation $g - 2$ a quarter, and so on. Once created, these packets are broadcast in the neighborhood with the latest generation first and earliest generation last. Apparently, the greater w is, the more packets are broadcast periodically, the larger the communication overhead of the protocol is, and the higher the network throughput may be. Essentially, the purpose of such a weight allocation among generations is to boost the late generations with sufficient initial presence in the network for them to propagate through.

IV. Experiments

We conducted experiments both on a set of 10 Apple iOS devices and in the ONE simulator. These experiments were designed for the same scenario in part of the Engineering Building on campus of Memorial University of Newfoundland. The 10 mobile users follow some prescribed paths in the building in a 30-minute iteration. The users are divided into three groups of 3, 3, and 4 devices, respectively. Each group has a “base” in the building, as numbered in Figure 1. During the test, a user from a group leaves his/her base, walks along the path, for example as depicted in the figure, makes a stop at the other two bases for about a minute each, and returns to the base. Subsequently, the next user in the group would repeat the same routine. Users of different groups follow slightly different paths, especially in opposite directions in certain segments, so that they can meet users of other groups en route. These routines were intended to mimic both grouped and individual mobility patterns in an academic setting, and were used both in real-device and simulated tests. The simulator was programmed to have the same mobile groups and mobility patterns.

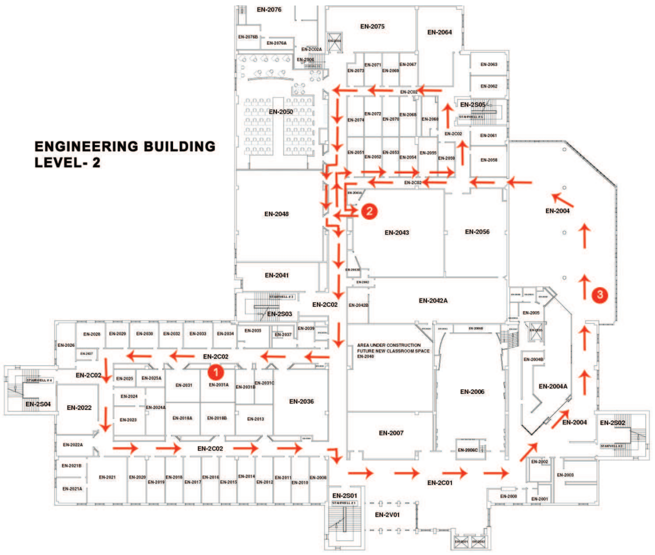

Fig. 1. Path of a mobile user

A. Real-device

To have a heterogeneous network, we used a set of different iOS devices because there is no interoperability between iOS and Android OS over Bluetooth with GameKit. Their model, processor clock, Bluetooth version, and quantity are listed in Table I.

Model	SoC and CPU cores	Bluetooth version	Quantity
iPod touch 4	A4, 1@0.8GHz	2.1	1
iPhone 4	A4, 1@1.0GHz	2.1	1
iPad 2	A5, 2@1.0GHz	2.1	2
iPod touch 5	A5, 2@1.0GHz	4.0	2
iPad mini	A5, 2@1.0GHz	4.0	3
iPad 4	A6X, 2@1.4GHz	4.0	1

TABLE I
iOS DEVICES USED IN EXPERIMENTS

We tested the network-coding-based broadcast against the other four forwarding-based approaches, each in a separate iteration. During an iteration, a devices generates a message every 90 seconds. The messages are randomly coded and sent every 15 seconds (Section III-B) or selectively advertised as digests every 15 seconds (Section III-A). For the case of network coding, we set the generation size to 50 messages, i.e. 5 messages per device per generation. To have about the same link layer data load across these five different methods, we are particularly interested in setting the generation allocation weight w to 0.5. When $w = 0.5$, we were able to keep the data sending rate at about 25kbps and receiving rate at about 50kbps across the board. (Note that we are using a broadcast service from the API, so there is no conservation of data flow.) Relevant parameters are summarized in Table II. We recorded when messages were decoded (for network coding) and received (for non network coding) on each device. At the end of the test, these logs were uploaded to a server for centralized synthesis and post-processing.

Parameter	Value
number of nodes in network n	10
total simulation time T	1,800 seconds
node mobility model	walk along prescribed paths
message generate rate per device t	every 90 seconds
message length s	4,000 bytes
number of digests advertised l	10 messages
size of network coding generation G	50 messages
interval of digest advertisement τ	every 15 seconds
interval of coded packets broadcast τ	every 15 seconds
generation allocation weight w	0.5, 1, 2

TABLE II
PARAMETERS OF DEVICE TESTS

We are interested in how widely messages are disseminated in the network, measured in *extent*, which is somewhat similar to the packet delivery ratio in unicast. After a message is generated, it is first stored at the originator, and as time goes on, it reaches more and more nodes. We observe how many other nodes a particular message has reached by the end of the 30-minute test. For a given message m, we denote the set of nodes in the network that m has reached other than the message originator itself by O_m. Thus, the extent of message m is defined as $|O_m|$, i.e. how many other devices the message has reached in the end. Among our 10 devices, 200 messages are injected in total, collectively denoted by $\mathcal{M}$. As such, we plot a histogram of the extent over $\mathcal{M}$ for network coding with $w = 0.5$, 1, and 2, and for the non-coding approaches in Figure 2. Note that $w = 0.5$ is the case when network coding has a comparable communication overhead as the non-coding approaches.

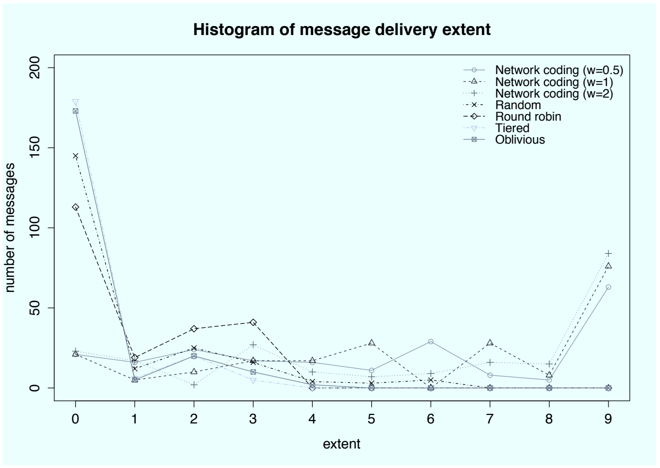

Fig. 2. Broadcast extent for real devices

In the figure, we can see that the non-coding approaches all end up with many messages not being delivered to any other node, i.e. the case of extent 0, because of the very sporadic connections among devices. Among these methods, when there is more equal opportunities of messages being advertised, as with the cases of Random and Round robin, the extent is slightly better. The other two approaches, Oblivious and Tiered, would favor newly injected messages but, unfortunately, they can be relentless moving on with new messages and permanently leave certain old messages behind if they miss the window. In stark contrast, the three network coding variants are able to send nearly half of the messages to all 9 other devices. Although for $w = 2$ the number of messages reaching all devices is slightly higher than when $w = 0.5$ or 1, they are fairly comparable, showing that $w = 0.5$ being very effective and efficient. Table III is a consolidation of the histogram into two cases, messages reaching only the minority in the network (0-4 other devices) and those reaching the majority (5-9 other devices). We can see that network coding was able to utilize the transient links very well while the non-coding forwarding could hardly make any progress. Note that for the messages generated near the end of the experiments, they barely had any time to propagate afar, and all these approaches would have better extent metrics if we gave them more time by allowing a "damping" period at the and of the test.

Method	$1 \sim 4$	$5 \sim 9$
Network coding ($w = 2$)	45.0%	55.0%
Network coding ($w = 1$)	31.5%	68.5%
Network coding ($w = 0.5$)	34.5%	65.5%
Oblivious	96.0%	4.0%
Tiered	100.0%	0.0%
Random	100.0%	0.0%
Round robin	100.0%	0.0%

TABLE III
NUMBER OF MESSAGES REACHING 1-4 AND 5-9 DEVICES

B. Simulated

Although The ONE supports an arbitrary data rate at the Link Layer for nodes within a given range, our preliminary tests show that it could not closely simulate the iOS Bluetooth radios as is. Apparently, we needed the ability to customize other aspects of the link layer. Thus, we modified the simulator by adding two parameters to control its behavior. First, we added a connection delay d for the time that it takes the GameKit to negotiate and connect two devices when they come in range. Second, we also gave a link a failure probability p to accommodate the fact that some transmission may fail when the radio is busy. (Note that these changes are by no means to catch how exactly the Bluetooth radios negotiate between each other, how they create and maintain synchronous or asynchronous links, or how they resolve collisions and provide reliability as it would in ns-2. They are simply to parameterize their synthesized effects at the higher layers.) We then simulated a 10-node network, where all nodes are identical hardware-wise and the data rate was 2Mbps for Bluetooth EDR. After testing various combinations of d and p and measuring the message deliver extent, we found that when we gave d a uniform value between 0 and 10 in the experiment and set $p = 50\%$, the ONE has a very close behavior as using actual iOS devices. Due to limit of space, we only report results for these particular values in Figure 3. Furthermore, we consolidated the data the same way as with real devices and summarize it in Table IV. We can see that the simulator yields very similar relative performance between network-coding and non-coding based approaches in this case.

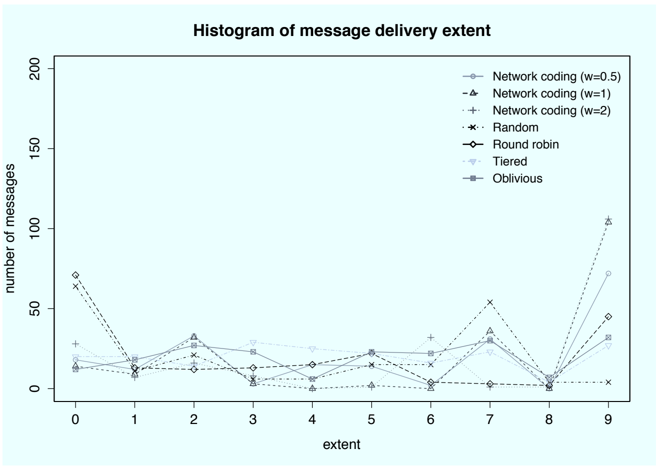

Fig. 3. Broadcast extent in simulation

Method	$1 \sim 4$	$5 \sim 9$
Network coding ($w = 2$)	40.5%	59.5%
Network coding ($w = 1$)	29.0%	71.0%
Network coding ($w = 0.5$)	29.5%	70.5%
Oblivious	54.0%	46.0%
Tiered	62.0%	38.0%
Random	54.0%	46.0%
Round robin	43.0%	57.0%

TABLE IV
NUMBER OF MESSAGES REACHING 1-4 AND 5-9 NODES IN SIMULATION

V. CONCLUDING REMARKS

We started out with a goal of experimental studies of DTNs of medium-to-large sizes to assess their performance with and with out using network coding. Due to the prohibitive cost of using real devices, both in terms of monetary and time senses, the process of directly working with mobile devices can be very laborious and error-prone. We then decided to find out how a simulator widely-used in DTN research, the ONE, would mimic real devices by a side-by-side comparison between the exactly same, real and simulated, scenario of 10 devices. The performance gain of network coding over conventional data forwarding was evident through real-device experiments. More importantly, we found that, after enhancing the link layer with a few necessary parameters to simulate the iOS GameKit, tests using the ONE would produce very similar performance to using the Apple iOS devices. We now have more confidence that the ONE would yield more reliable results in larger networks with the enhancement.

In future research, we would like to use the enhanced ONE simulator to study larger DTNs. For example, we know that the generation size G is an effective parameter to control network coding. If the number of nodes in the network is known a priori, setting this parameter to a fixed value may serve this purpose. However, a more flexible and scalable approach would be to obtain a good value of it as the network operates and to allow it to adjust to the network conditions dynamically. This issue was studied in the simulated tests of Widmer and Le Boudec [20], and we intend to further investigate it using real devices. We also plan to test the ONE against Android OS devices to find the best parameters in order to bridge the gap between these two as well. Knowing the much higher heterogeneity among devices on this platform, we expect more combination tests to achieve such a goal. The benefit would be producing an even more trust-worthy simulator for future experimental research on DTN and network coding.

REFERENCES

[1] Delay tolerant networking research group. http://www.dtnrg.org/wiki/Home.

[2] Haggle project. http://code.google.com/p/haggle/.

[3] The interplanetary network (IPN). http://tmo.jpl.nasa.gov/.

[4] The opportunistic network environment (ONE) simulator. http://www.netlab.tkk.fi/tutkimus/dtn/theone/.

[5] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000.

[6] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. Trading structure for randomness in wireless trading structure for randomness in wireless trading structure for randomness in wireless opportunistic routing. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 169–180, 2007.

[7] Yuanchu Chen, Walter Taylor, Sam Coxon, and Jason H. Moore. isnac: Infrastructureless social networking at academic conferences. In *Demo at the 31st IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2012.

[8] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding. In *Proceedings of Allerton Conference on Communication, Control, and Computing*, 2003.

[9] Christina Fragouli, Jean-Yves Le Boudec, and Jorg Widmer. Network coding: An instant primer. *SIGCOMM Computer Communication Review*, 36(1):63–68, 1 2006.

[10] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 145–158. ACM, 2004.

[11] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, pages 96–107, 2002.

[12] Maurice J. Khabbaz, Chadi M. Assi, and Wissam F. Fawaz. Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges. *IEEE Communications Surveys & Tutorials*, 14(2):607–640, 2012.

[13] Yunfeng Lin, Baochun Li, and Ben Liang. Efficient network coded data transmissions efficient network coded data transmissions in disruption tolerant networks. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, 2008.

[14] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, July 2003.

[15] Anders Lindgren and Pan Hui. The quest for a killer app for opportunistic and delay tolerant networks. In *Proceedings of the 4th ACM Workshop on Challenged Networks (CHANTS)*, pages 59–66. ACM, 2009.

[16] Xu Liu, Yuanzhu Chen, Cheng Li, Walter Taylor, and Jason H. Moore. Message prioritization of epidemic forwarding in delay-tolerant networks. In *Proceedings of International Conference on Computer Networking & Communications (ICNC)*, 2014.

[17] Muriel Medard and Alex Sprintson. *Network Coding: Fundamentals and Applications*. Academic Press, 1st edition, 2011.

[18] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking*, pages 252–259. ACM, 2005.

[19] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-200006, Duke University, 2000.

[20] Jorg Widmer and Jean-Yves Le Boudec. Network coding for efficient communication in extreme networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking*, pages 284–291, 2005.

[21] Jian Zhang, Yuanzhu Chen, and Ivan Marsic. MAC-layer proactive mixing for network coding in multi-hop wireless networks. *Computer Networks*, 54(2):196–207, 2010.

[22] Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing. In *Proceedings of the 13th annual ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 195–206, 2007.

[23] Zhensheng Zhang and Qian Zhang. Delay/disruption tolerant mobile ad hoc networks: latest developments. *Wireless Communications and Mobile Computing*, 7(10):1219–1232, December 2007.