Network Coded Routing in Delay Tolerant Networks: An Experience Report

[1]Agoston Petz, [2]Brenton Walker, [1]Chien-Liang Fok, [2]Calvin Ardi, and [1]Christine Julien,

[1]The University of Texas at Austin, [2]Laboratory for Telecommunication Sciences

Email: [1]{agoston, liangfok, c.julien}@mail.utexas.edu, [2]{brenton, calvin}@ltsnet.net

TR-ARiSE-2012-003

ARiSE
Advanced Research in Software Engineering

Network Coded Routing in Delay Tolerant Networks: An Experience Report

Agoston Petz, Chien-Liang Fok, and Christine Julien
University of Texas-Austin
{agoston, liangfok, c.julien}@utexas.edu

Brenton Walker and Calvin Ardi
Laboratory for Telecommunications Sciences
College Park, MD, USA
{brenton, calvin}@ltsnet.net

ABSTRACT

In delay-tolerant networks, end-to-end routes are rarely available, and routing protocols must take advantage of the opportunistic interactions among nodes to deliver packets. Probabilistic routing performs well in such networks and has been the dominant focus of research in this area. However, creating efficient routing protocols is challenging because to reduce latency, one often needs to replicate messages thus increasing routing overhead. Network coding has been explored as a way to increase throughput in DTNs without a significant increase in overhead, and network coded routing approaches have shown promising results. In this paper, we report on our experience integrating both erasure coded and network coded routing into the well-adopted DTN2 Reference Implementation. We implement our routing module and evaluate it via small real-world field tests.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Store and forward networks

Keywords

Delay-Tolerant Networks, Bundle Protocol, Network Coding

1. INTRODUCTION

As delay-tolerant networks (DTNs), dynamic networks in which nodes use opportunistic contacts to forward data, gain traction in both research and real-world deployments, applications increasingly demand efficient and cost-effective routing solutions. DTNs apply in many application instances, especially in developing regions lacking network infrastructure. In an urban setting, commuter transportation systems can carry messages from one region of the city to another [10]; similar mechanisms can connect remote villages that may be well-connected internally but less reliably connected to the wider world [19].

ExtremeCom '11, September 26-30, 2011, Manaus, Brazil.

Significant efforts have focused on routing in DTNs [4, 12, 16, 18], and great strides have been made to support the DTN paradigm. Among routing protocols proposed for challenged networking environments, approaches using *network coding* show significant promise [13, 15, 23, 27, 28]. In network coded routing, intermediate nodes not only forward incoming packets, but also "mix" packets from multiple sources to increase information content in forwarded packets. Such approaches are particularly useful in DTNs, where opportunities to exchange data are intermittent and unpredictable. Network coding can reduce both routing overhead and delivery latencies compared to probabilistic routing without coding [26]. However, the study of network coded routing in DTNs has only been carried out theoretically or in constrained simulations. Given its significant promise, network coded routing must be integrated with practical DTN architectures and tested in deployments.

Our novel contributions are: (*i*) a network coding routing module for DTN2; (*ii*) a documented process for inserting a new routing module in DTN2 ; (*iii*) a testbed evaluation of our network coded routing module; and (*iv*) results from using our network coded routing module to support a real-world DTN deployment in small-scale field tests. Our contributions are almost exclusively practical; such advancements are necessary in ensuring real-world impacts of theoretical DTN research. The lessons learned from practical experiences in building, testing, and deploying real implementations for DTN routing help to inspire future research.

2. BACKGROUND AND RELATED WORK

The benefits of network coded routing in DTNs have been extensively studied and simulated. Erasure coding can improve the worst-case delay in DTNs [2, 15, 25, 26], and network coded routing compares favorably with probabilistic routing in addition to having lower overhead [27]. Combining random linear coding with epidemic routing has achieved better transmit power versus delay performance, especially when buffer sizes are constrained [28]. Network coding can increase throughput even in networks with non-homogeneous mobility [5]. We build on these contributions by providing a concrete implementation of network coded routing for DTNs.

The DTN2 Reference Implementation [7] provides an architecture for developing, evaluating, and deploying DTN protocols. DTN2 is an implementation of the bundle protocol [8], an application-layer protocol for delivering messages (called *bundles*) between endpoints in a DTN. In DTN2, a "router" determines a node's forwarding strategy and gov-

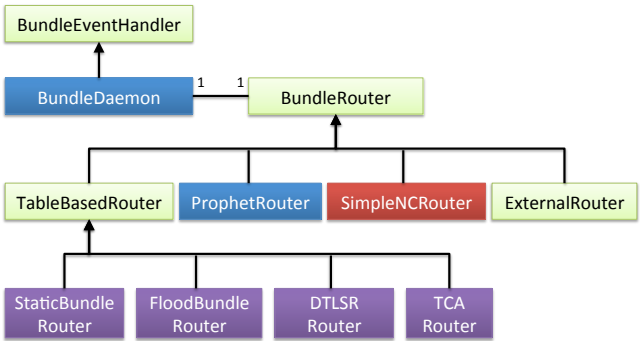

Figure 1: DTN2 modules related to routing

erns how a node determines which bundles (or potentially bundle fragments) should be forwarded to whom and when. Within DTN2, there are two types of routing modules: external and internal. The external router interface allows external programs to register with DTN2 to receive bundle events and make forwarding decisions [6]. It has been used to define and implement new bundle routing mechanisms [3, 11]. Developing an internal router is more complicated because it relies on DTN2's internal architecture. However, internal routing modules can more directly interact with bundle maintenance mechanisms and data structures inherent to DTN2. New internal routing modules have been added to DTN2, most notably PRoPHET [16]. We chose to add network coded routing as an internal router module due to our need to make fine-grained routing decisions. Specifically, a coded routing approach needs to interact with bundle fragments and proactively break a larger bundle into fragments and perform coding across these fragments.

To understand internal routing modules, we look at how routing is structured in DTN2. Figure 1 shows an architectural view of the routing components. All routing modules are instances of `BundleRouter`, to which `BundleDaemon` passes bundles (either from the network or from the API). Except for PRoPHET, all internal routing modules extend `TableBasedRouter`, which maintains data structures that track links to neighbors, routes in the network (and their next hops from this node), and which bundles are waiting to be sent on which links. Routing modules built on `TableBasedRouter` modify its behavior in different ways, for example `FloodBundleRouter` forwards every bundle to every neighbor and ensures that bundles do not get deleted after they are transmitted.

`TableBasedRouter` provides a number of useful abstractions for building DTN routing protocols. However, it also has several drawbacks, most of which arise from its dependence on the data structures that manage links (whether they are currently "up" or "down"), the bundles forwarded and waiting to be forwarded on the links, and the bundles seen and processed by a node. The numerous updates to these data structures (multiplied by the number of links) cause significant routing and re-routing delays. More significantly, routing decisions are forced to be link- and queue-centric; decisions made when processing a bundle and choosing outgoing links are difficult to undo should new information arrive before the bundle leaves the link queue. Finally, there is no randomization or probabilistic behavior; all outgoing bundles are queued on links in the same order. These design decisions in `TableBasedRouter` are motivated largely by its original intended use to support static routing. These limitations proved excessive in more dynamic routing situations, which led us to develop a new distinct router inspired by `TableBasedRouter`.

3. CODING-AWARE ROUTING IN DTN2

Erasure coding and network coding operate on the same basic principles. Both split a data unit, in our case a *bundle*, into fragments and create linear combinations of fragments to send to other nodes. The original bundle is never sent unencoded; different combinations of fragments are disseminated, and the destination needs to only receive some number of linearly independent encoded fragments to reconstruct the original bundle. The two coding techniques differ in which nodes generate encodings. In erasure coding, only the source generates encodings while network coding allows intermediate nodes to generate new random linear combinations of received fragments, resulting in increased "mixing" of information in the network and, theoretically, a more robust randomized routing protocol.

In developing our routing implementation, we present an overview of our terminology:

Bundle: the fundamental data unit of the bundle protocol [22]. Bundles too large to be transferred in a single contact are fragmented; we encode across these fragments. Every bundle has a globally unique identifier (*GUID*).

Fragment x_i: a bundle is split into M (non-encoded) fragments of k bits, such that $x_i \in \mathsf{GF}(2)^k$. Each fragment is associated with its parent bundle's *GUID*.

Coefficient vector $\boldsymbol{c}$: a vector, $\boldsymbol{c} = \langle c_1, c_2, \ldots, c_M \rangle$, where $c_i \in \mathsf{GF}(2)$ and $i \in [1, M]$, controls which fragments to combine (`xor`) to create an encoded fragment.

Encoded fragment / Codeword $w_{\boldsymbol{c}}$: an encoded bundle made up of some linear combination of fragments such that $w_{\boldsymbol{c}} = \sum_{i=1}^{M} c_i x_i$ for some coefficient vector $\boldsymbol{c}$.

Re-encoding vector $\boldsymbol{d}$: on a node with r encoded fragments, a re-encoding vector $\boldsymbol{d} = \langle d_1, d_2, \ldots, d_r \rangle$ can create a new linear combination $w_{\boldsymbol{c'}} = \sum_{i=1}^{r} d_i w_i$. Re-encoding vectors are only used in network coded routing, and re-encoding is only allowed for encoded fragments associated with the same original bundle (i.e., with the same *GUID*).

Our coded routing implementations rely partially on DTN2's ability to break large bundles into fragments. Coded routing protocols can create encoded fragments from bundle fragments and distribute these encoded fragments independently using opportunistic connections that are inherent to DTNs; when a receiver has acquired *enough* pieces of information, it can reconstruct the original data. It is not necessary for the receiver to acquire *all* of the original fragments. Coded routing can decrease overhead and latency in DTNs versus purely probabilistic forwarding [26, 27]. This is intuitive, since an intelligent coding (and re-encoding) scheme engenders *innovative* content in the fragments exchanged, increasing the likelihood that a received fragment increases the receiver's total information. With respect to DTN2, both the original (application) data and the encoded fragments are stored in bundles that move through the modules of the architecture implementation.

In our protocols, when a bundle is received from the network or through the application programming interface (API),

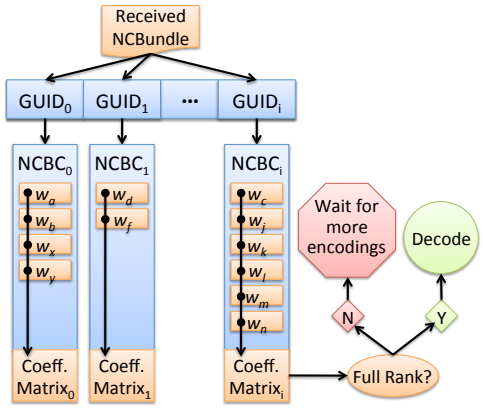

Figure 2: Incoming Bundle Data-Flow Diagram

the node checks to see if the received bundle is an encoded fragment. If not, and the bundle is larger than some threshold, the node splits it into fixed-size fragments. Each fragment is tagged with its corresponding coefficient vector ($\boldsymbol{c}$) and the *GUID* of the original bundle for record keeping. Routers disseminate bundles containing an encoded fragment inside the payload.

3.1 The Essential Data Structures

To enable coding-aware routing in DTN2, we created data structures for managing and manipulating encoded fragments distributed in bundles. The following data structures enable us to intelligently store and forward encoded fragments and to easily reassemble the original bundles:

Network Coding Metadata Extension Block (NCMD Block) This metadata extension block is attached to encoded bundles. The information contained within is based partly on a preliminary Internet-Draft [1] and relies on DTN2's extension block and metadata extension block support [24]. An NCMD Block carries information about the coefficient vector used to generate the payload and the *GUID* corresponding to the original bundle. Putting this data in an extension block, as opposed to the bundle payload, gives the router access to the encoding information without loading and parsing the payload from the data store.

Network Coded Bundle (NC Bundle) The `NCBundle` class is a wrapper for encoded bundles; it is a simple aggregation of a pointer to a bundle and an associated NCMD Block. To keep from processing the extension block repeatedly, we store the fields parsed out from the NCMD Block in a data structure. The bundles encapsulated as NC Bundles contain encoded fragments generated from the original, larger application bundles.

Network Coded Bundle Collection (NC Bundle Collection) Our central data structure is `NCBundleCollection`, a table of collections of NC Bundles, indexed by *GUID*. Figure 2 shows how a node handles incoming bundles, interacting with the appropriate NC Bundle Collection to store encoded fragments and assess the rank of each application-level bundle. A received NC Bundle is first sorted by its *GUID* into the correct collection, and the coefficient vector used for the encoding is copied from the NCMD Block into a row-reduced matrix whose rows span the space of the current collection. The matrix is echelonized and the rank checked. If the rank of the collection increased, the NC Bundle is retained and otherwise discarded.

If the row-reduced coefficient vector matrix in an NC Bundle Collection reaches full rank, the matrix is inverted. The columns of the inverse matrix contain the coefficients needed to sum the NC Bundle payloads together to decode the original bundle fragments. Since the coefficient vectors are are chosen randomly, they may not be linearly independent, and sometimes a node must receive more than N encoded fragments before it can decode the set. The NC Bundle Collection class uses the `m4ri` library [17] to do fast binary linear algebra computations. Our routers can also operate in *non-rank-checking mode* to support experimentation. In *non-rank-checking mode*, matrix manipulations are disabled and an NC Bundle is discarded only if that exact bundle has been seen previously.

3.2 The Protocols

We next describe coding-aware routing protocols. The routing modules sit in DTN2 as shown in Figure 1.

Erasure Coding. `SimpleECRouter` implements an *erasure coded* routing protocol with encoding done at the application layer. The first iteration accomplishes several objectives: (*i*) managing encoded bundles (which are fragments of larger application bundles) using the new data structures and (*ii*) increasing the intelligence with which bundles are chosen to be sent across links, in comparison to existing router modules or *k*-flood protocol.

Given an available link, `SimpleECRouter` selects one of the NC Bundle Collections at random, selects a random encoded fragment from within that collection, and queues it on the link. When the queued bundle is sent, the process repeats itself. This method could be unfair if some NC Bundle Collections contain more bundles than others. Other optimizations are possible; for example, we could give priority to newly created NC Bundle Collections (*i.e.*, new *GUID*s). We did not explore these enhancements in our first iteration.

As long as there are open links, `SimpleECRouter` will send randomly selected encoded fragments without considering which fragments have already been sent. `NonRedundantECRouter`, created as a subclass of `SimpleECRouter`, keeps track of which encoded fragments have been sent on each link to prevent distributing redundant bundles. One of the motivations for coded routing is that the router will, with high probability, send linearly independent bundles, successfully increasing the total data received. This will be especially true if contacts are short and only a few bundles can be sent within a contact interval. However, enhancements that can increase the *innovativeness* of exchanged bundles can increase routing performance, leading us to network coded routing.

Network Coding. In basic network coding, intermediate nodes generate new "mixes" from received encoded fragments, increasing the innovative encodings in the network. Instead of selecting a bundle to send from the NC Bundle Collections (`SimpleECRouter`), `SimpleNCRouter` creates a new encoded fragment to send. It first randomly

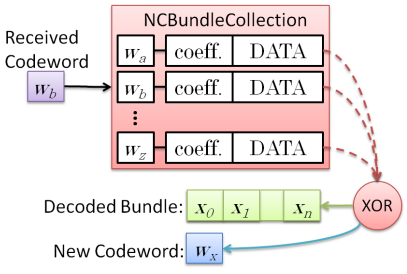

Figure 3: Encoding/Decoding

selects an NC Bundle Collection, then chooses a random re-encoding vector and `xor`s together the payloads of the NC Bundles indicated in the re-encoding vector. It then sends the new bundle on the available link. The configurable weight of the re-encoding vector defaults to the log of NC Bundle Collection rank. Figure 3 shows this process of generating a new NC Bundle and how stored NC Bundles in an NC Bundle Collection are combined to recreate the original bundle. In general, there are many ways to create encoded fragments (*e.g.*, arithmetic over larger finite fields); we use `xor` for simplicity.

`SimpleNCRouter` is fairly general, *e.g.*, `SimpleECRouter` is a special case of `SimpleNCRouter` in which new linear combinations are generated with a weight of one. Building `SimpleNCRouter` required us to inject new bundles from within the router (instead of acquiring the bundles from the API or network), which presented implementation challenges.

Auxiliary Data Structures and Injected Bundles. `SimpleNCRouter` generates new bundles that are linearly dependent on other bundles in the NC Bundle Collection, and are discarded after transmission. We allocate the object associated with a transient NC Bundle as a `TempBundle`, a `Bundle` not stored persistently on disk; these bundles are faster to create and process.

`SimpleNCRouter` deals with three types of bundles; it keeps track of the bundles using three `BundleList` structures.

original_bundles_list: pointers to original un-encoded bundles from the API or network and bundles reconstructed from full-rank collections. `SimpleNCRouter` never transmits these but may deliver them to applications.
temp_list: pointers to the `TempBundle`s described above.
persistent_list: pointers to bundles that have been added to an NC Bundle Collection.

Any non-innovative bundle is not added to any of these lists and scheduled for deletion.

Configuration Options. `SimpleNCRouter` has several configuration options to experiment with performance and functionality (default values are in parentheses):

rank_check (true): if set, the router will discard received NC Bundles that are not innovative.
reencode (true): if set, router will generate new NC Bundles with re-encoding vectors of weight more than 1.
auto_decode (false): if set, router immediately decodes when an NC Bundle Collection reaches full rank.
keep_original_bundles (true): if set, router retains original bundles, even after fragmenting into NC Bundles.
chunk_size (50000B): size of the fragments.
max_weight (0): max weight of re-encoding vectors. If 0, the weight is the log of the rank of the collection.

We discuss a number of potential future refinements to further improve `SimpleNCRouter` in Section 5.

4. EVALUATION

Our goals in evaluating our routing protocol implementations are to demonstrate the functionality of the implementations, show how they support deployments on real devices, and to gain insights into future improvements. We present our results in terms of the cumulative rank achieved at a node in the network. Clearly, a coded routing protocol's success is related to its ability to achieve full-rank as it is only then that a bundle is delivered. In general, a protocol that achieves full rank faster can be considered better assuming equivalent overheads. We do not specifically treat the topic of routing overhead, although it has been shown analytically and through simulation that network coding has comparable overhead to probabilistic protocols [27].

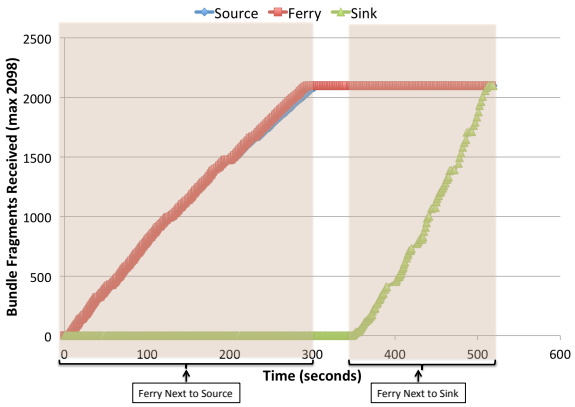

Figure 4: `FloodBundleRouter`, synchronous mobility

We first used the Pharos testbed [20], a mobile ad-hoc network consisting of Proteus [21] mobile nodes. Each node consists of three interchangeable planes: mobility, computation, and sensor/actuator. The mobility plane enables autonomous physical movement, although in this evaluation we use human-based mobility. The computational plane includes an x86 motherboard running Linux v2.6 and a commodity Atheros IEEE 802.11g wireless interface. We did not use the sensor/actuator plane. Our experimental setup consists of three Proteus nodes: a source, a sink, and a data ferry. The source and sink are placed at static locations on two sides of a building such that they cannot directly communicate. The source sends a 100MB file using the DTN2 API, and the ferry moves in loops around the building carrying data from source to sink. We compared our two coding routers with `FloodBundleRouter`.

By default, `FloodBundleRouter` does not fragment the 100MB bundle unless forced to do so due to disconnection. This differs from our coding routers which divide the original 100MB bundle into 2098 50KB chunks. To enable fair comparison, we evaluate `FloodBundleRouter`'s latency of transmitting 2098 50KB bundles. To gain a base-line performance, we parked the ferry within range of the source until it received the full bundle before moving it to the sink. Likewise, we kept the ferry within range of the sink until the full bundle was transferred. We refer to this movement pattern as *synchronous*, since movement is synchronized with communication. Figure 4 shows the results. A non-encoded 100MB bundle can be transferred from source to ferry in 300s, and from ferry to sink in 172s. These latencies assume ideal conditions where connectivity is continuous between participating nodes.

To experiment with erasure coding, we used our `SimpleNCRouter` at the source to generate initial encodings; the ferry node and sink did not do any re-encoding of received fragments (*i.e.*, they behaved like `SimpleECRouter` described in Section 3). In these experiments, the source fragments the 100MB bundle into 50KB encoded fragments and the routers maintain fragment state across connections (*i.e.*, they remember innovative codewords they received even in the face

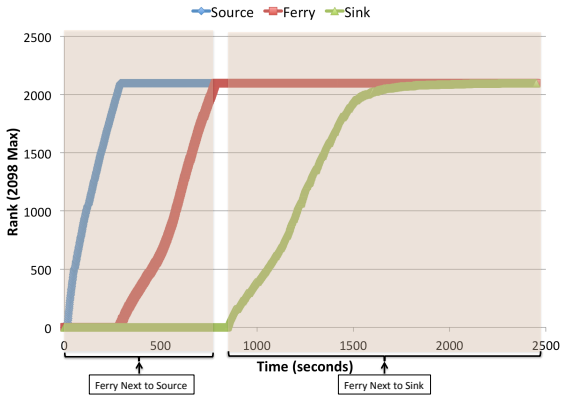

Figure 5: Erasure coded routing, synch. mobility

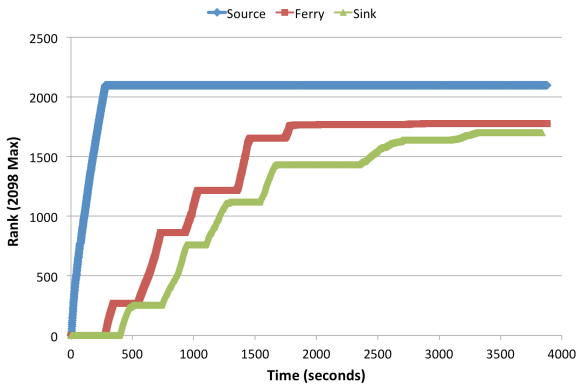

Figure 6: Erasure coded routing, asynch. mobility

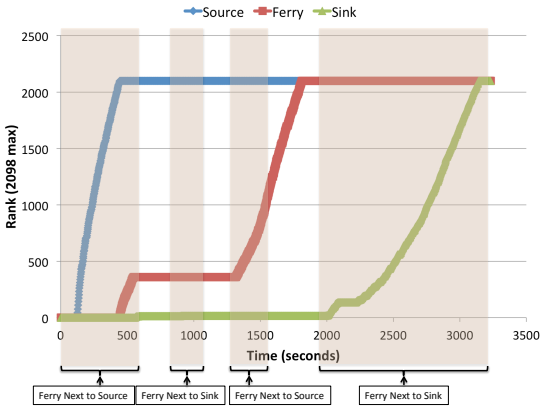

Figure 7: Network coded routing, asynch. mobility

of disconnections). We used the same synchronous movement pattern as above, keeping the ferry next to the source until it had achieved full rank and then moving it to the sink. The results are shown in Figure 5. The transfer time is approximately three times longer than for `FloodBundleRouter`; the difference is likely due to the time spent encoding, rank-checking, and re-transmitting bundles in both directions. Optimizing these functions is an obvious point of future work for coded routing implementations. Also notice that the curve for the sink's rank in Figure 5 reaches a high rank relatively quickly but takes a while to get the last few encoded fragments necessary to reach full rank. This is demonstrative of the *coupon collector's problem*—it takes very little time to collect the first few coupons but a long time to collect the last few. Having a better understanding of the relative innovativeness of generated encoded fragments could help alleviate this problem; we discuss the potential for such an enhancement in the next section.

Synchronizing movement with communication is not possible in real deployments; contact unpredictability is a motivating factor for coded routing in DTNs. We also performed evaluations using *asynchronous* movement, in which node movement is decoupled from data transmission. Figure 6 shows the results for erasure coded routing under asynchronous movement. The ferry is initially able to consistently deliver innovative encoded fragments from the source to the sink, but the ranks of both the ferry and the sink eventually level off, short of full rank. To understand why, we observed the ferry rapidly sending encoded fragments to the source, which the source promptly deleted because they were not innovative. These send actions from the ferry appears to have prevented the source from sending anything to the ferry, effectively starving itself of new innovative encoded fragments. A similar phenomenon (albeit at a smaller scale) was observed between the sink and the ferry.

Network coding creates new linear combinations in the network. We performed the same asynchronous movement experiment using `SimpleNCRouter` on all three nodes; the results are in Figure 7. The benefits of network coding are clear; the sink is able to reach full rank. However, the first time the ferry and sink met, only 16 innovative bundles were delivered to the sink. This is because the sink also sends bundles back to the ferry and in this case it sent bundles so rapidly that the ferry did not have a chance to send bundles to the sink. Further optimizations could address this problem, including storing and adapting to the innovativeness of outgoing bundles on a link relative to incoming bundles on that same link. In addition, reaching full rank under asynchronous movement in `SimpleNCRouter` takes approximately six times longer than under `FloodBundleRouter` (albeit with synchronous movement). This is due largely to the time spent encoding bundles, re-transmitting bundles, and rank checking; this overhead is significant and future protocol revisions must consider ways to reduce its impact on data delivery latency, for example by pre-generating encodings. In addition, using multiple ferries can increase the performance of network coded protocols by introducing additional points of opportunism and information mixing.

This evaluation is limited in that it demonstrates the successful operation of our erasure and network coding implementations for very small (three node) networks. This was motivated by our desire to show the operation on real devices in real networks while still being able to carefully control the experiments. While we have been able to demonstrate the benefits of coding-aware routing in these small networks, we expect that its behavior will shine even more in larger multi-node networks. These expectations are bolstered by further experiments we have performed in the VirtualMeshTest (VMT) mobile wireless testbed [9, 14], which allows us to subject real wireless nodes running real DTN stacks to emulated mobile environments. Using the testbed, we have done extensive experiments in larger networks. As one example, we have shown in a 50 node network with a highly mobile source and a highly mobile sink, that the `SimpleNCRouter` can move 1GB of data from the source to the sink while the `FloodBundleRouter` failed to deliver even 100MB. The complete description of these results is omitted from

this paper for brevity. We have also used `SimpleNCRouter` as the basis for a more complete network-coded router used in small scale field tests in Boston and New York City, in which we showed that the router successfully delivered large bundles from a single source to multiple nodes acting as ferries and receivers. These various experiments have given us a variety of valuable feedback on bugs in our routing implementations, practical challenges of real environments, and hints about features we should include in future versions of our routers.

5. CONCLUSION AND FUTURE WORK

We presented an implementation of coded routing for DTNs and provided our methodology for developing and inserting a new router into the DTN2 Reference Implementation of the bundle protocol. Until now, the study of network coding for DTNs has been constrained to analytical or simulated domains, and our evaluation on commodity hardware is the first to demonstrate the practicality of network coding in a real-world DTN. Although our contribution is mostly of a practical nature, we provide a platform on which future improvements in network coding can be easily tested. The lessons learned from building and deploying a concrete network coding protocol provide necessary insights if such protocols are to be applied to real DTN deployments.

We are planning a number of refinements to our implementation. `SimpleNCRouter` does not check that encodings are innovative with respect to previously created encodings. Doing so could increase the processing overhead on the sender, but would reduce the communication overhead, and we plan to investigate the trade-off. Additionally, we intend to improve the scheduling of receive and send processing, since our current implementation tends to fall behind processing received bundles. Also the initial bundle fragmentation and decoding tasks could be moved outside the event thread, which currently slows sending and processing bundles. Finally, when a link becomes available, we use a fairly naïve algorithm to select the collection from which to generate an encoded fragment. Our policies are potentially unfair, and future protocol refinements will explore different policies for these selection mechanisms, for example by favoring newly discovered *GUID*s, or prioritizing the collections according to some metric.

6. REFERENCES

[1] A. Caro and J. Zinky. DTN erasure coding protocol (preliminary notes). To appear as Internet Draft.

[2] E. Altman, F. De Pellegrini, and L. Sassatelli. Dynamic control of coding in delay tolerant networks. In *Proc. of INFOCOM*, 2010.

[3] A. Balasubramanian, B. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. In *Proc. of SIGCOMM*, 2007.

[4] C. Boldrini, M. Conti, I. Iacopini, and A. Passarella. HiBOp: A history based routing protocol for opportunisitic networks. In *Proc. of WoWMoM*, 2007.

[5] M. Chuah, P. Yang, and Y. Xi. How mobility models affect the design of network coding schemes for disruption tolerant networks. In *Proc. of NetCod*, 2009.

[6] DTN external router interface. `http://www.dtnrg.org/docs/code/DTN2/doc/external-router-interface.html`.

[7] DTNRG. DTN bundle protocol reference implementation. `http://www.dtnrg.org/wiki/Code`.

[8] K. R. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. of SIGCOMM*, 2003.

[9] D. Hahn, G. Lee, B. Walker, M. Beecher, and P. Mundur. Using virtualization and live migration in a scalable mobile wireless testbed. *SIGMETRICS Perform. Eval. Rev.*, 38:21–25, January 2011.

[10] K. Harras, K. Almeroth, and E. Belding-Royer. Delay tolerant mobile networks (DTMNs): Controlled flooding in sparse mobile networks. In *Proc. of Networking*, pages 1180–1192, May 2005.

[11] HBSD: An external router for DTN2. `http://planete.inria.fr/HBSD_DTN2`.

[12] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *ACM SIGCOMM Computer Comm. Rev.*, 34(4):145–158, October 2004.

[13] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the air: Practical wireless network coding. In *Proc. of SIGCOMM*, 2006.

[14] Y. Kim, K. Taylor, C. Dunbar, B. Walker, and P. Mundur. Reality vs emulation: Running real mobility traces on a mobile wireless testbed. In *HotPlanet 2011 (to appear)*, 2011.

[15] Y. Lin, B. Li, and B. Liang. Efficient network coded data transmissions in disruption tolerant networks. In *Proc. of INFOCOM*, pages 1508–1516, 2008.

[16] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *Proc. of SAPIR*, pages 239–254, 2004.

[17] M4RI(e). `http://m4ri.sagemath.org/`.

[18] T. Matsuda and T. Takine. (p,q)-Epidemic routing for sparsely populated mobile ad hoc networks. *IEEE J. on Sel. Areas in Comm.*, 26(5):783–793, June 2008.

[19] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking connectivity in developing nations. *IEEE Comp.*, 37(1):78–83, January 2004.

[20] `http://mpc.ece.utexas.edu/pharos`.

[21] `http://proteus.ece.utexas.edu`.

[22] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), Nov. 2007.

[23] S. Sengupta, S. Rayanchu, and S. Banerjee. An analysis of wireless network coding for unicast sessions: The case for coding-aware routing. In *Proc. of INFOCOM*, pages 1028–1036, 2007.

[24] S. Symington. Delay-Tolerant Networking Metadata Extension Block. RFC 6258 (Experimental), 2011.

[25] B. N. Vellambi, R. Subramanian, F. Fekri, and M. Ammar. Reliable and efficient message delivery in delay tolerant networks using rateless codes. In *Proc. of MobiOpp*, 2007.

[26] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure-coding based routing for opportunistic networks. In *Proc. of WDTN*, pages 229–236, 2005.

[27] J. Widmer and J.-Y. L. Boudec. Network coding for efficient communication in extreme networks. In *Proc. of WDTN*, 2005.

[28] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. On the benefits of random linear coding for unicast applications in distruption tolerant networks. In *Proc. of WiOpt*, pages 1–7, 2006.