An Architecture for Context-Aware Adaptation of Routing in Delay-Tolerant Networks*

Agoston Petz[1], Angela Hennessy[2], Brenton Walker[2],
Chien-Liang Fok[1], and Christine Julien[1]
[1]The University of Texas at Austin, [2]The Laboratory for Telecommunications Sciences
{agoston, liangfok, c.julien}@mail.utexas.edu, {brenton, ahennessy}@ltsnet.net

ABSTRACT

We present a general framework for context awareness in delay tolerant networks. The framework introduces an adaptation portal through which external context agents can affect internal routing behavior. The context agent is decoupled from the router, enabling different forms of context awareness to be supported. In this paper, we use a router based on network-coding and identify an example set of three router configuration parameters that can be tuned by the external context agent. We implement our framework within the DTN2 reference implementation and test it using a prototype Context-Aware Network Coding (CANC) context agent and our own network coded internal router agent. Experimental results performed on both real and channel-emulated testbeds demonstrate our framework's feasibility and the significant efficiency gains (of up to 300%) we achieve in using it.

Categories and Subject Descriptors

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

Keywords

Delay-Tolerant Networks, Context-Aware, Bundle Protocol, Network Coding

1. INTRODUCTION

Delay-tolerant networks (DTNs) are often highly dynamic, demanding agile and adaptive routing protocols. Existing techniques for building such protocols often suffer from a lack of vital context information needed to achieve efficiency. This information is typically unavailable because it is network or application specific, e.g., relevant context often includes a node's role in the application, its future movements, and how data is being transmitted. In this paper, we explore the use of an external *context agent* to monitor such context in a dynamic environment and directly improve routing decisions within a DTN.

Efficient and adaptive routing decisions in DTNs require acquiring and assessing context from a myriad of sources. For example, to efficiently route data from a base camp to a remote village, data should only be transmitted to vehicles traveling towards the village. In this case, context consists of the data being transmitted, the vehicles present, and their planned movements. Context-awareness has long been important in mobile computing environments; in this paper we investigate its potential impact on DTN routing. Previously, to our knowledge, this was only done in tailored solutions that tie a single type of context to a particular DTN routing protocol (see Section 2). In contrast, we provide a general approach that decouples context acquisition and processing from specific routing protocols. This matches the general-purpose nature of many DTN routers like DTN2 [5], one of the most popular, which we enhance with our solution.

The novel contributions of this paper are as follows.

- We design and implement an *adaptation portal* in DTN2 through which *context agents* can improve router performance.
- We implement a *Context-Aware Network-Coded* (CANC) context agent that uses a node's role and movement information through which we can reconfigure router behavior.
- We evaluate our approach through two different network testbeds and demonstrate significant performance improvements when using our CANC context agent to control a previously-developed network coding router for DTN2 [16].
- We identify additional adaptation strategies for context agents that can further benefit DTN routing, with a specific focus on network coded routing.

2. BACKGROUND AND RELATED WORK

In this section, we discuss how context-awareness has been used in DTNs. This is followed by a summary of our prior work on network-coding in DTNs, which we use as the substrate for integrating our general approach to context-aware DTN routing. Specifically, we build upon our network-coding router's implementation within the DTN2 architecture.

2.1 Related Work: Context Awareness

Existing research for DTNs focuses largely on routing messages in challenging environments [2, 7, 12, 13]. Many of

*This work was funded in part by the US Dept. of Defense. The views expressed are those of the authors and may not necessarily reflect the views of the sponsoring agencies.

ExtremeCom '12, March 10-14, 2012, Zürich, Switzerland.

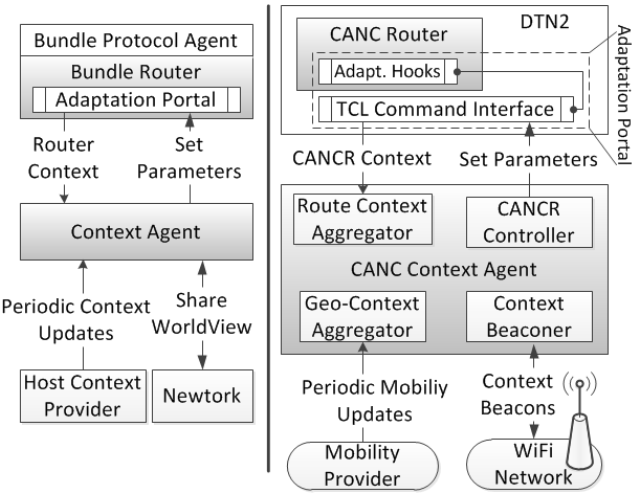
these protocols are tailored to networks exhibiting specific types of mobility [12, 19], negating the need to sense mobility as a contextual cue that influences adaptive behavior. Given the availability of other types of information, routing performance was tuned, for example, using information about the history of contacts [3, 4, 10] or by using cross-layer information [20]. While these approaches use context in DTN routing, they often make strong assumptions about the underlying network and the availability of context information, and, as a result, exhibit very rigid software architectures that are dependent on a single type of context. For example, CAR (Context-aware Adaptive Routing) takes as input mobility and contact pattern information and uses a Kalman filter to determine the best forwarding hops [14]. Generic context was used to infer potential mobility or contact patterns, which can be used by a routing protocol to make forwarding decisions in DTNs [1]. This is similar to the context we use to demonstrate our architecture.

Our work differs from previous approaches in that we focus on the architectural questions of providing a general-purpose framework for integrating context information with *any* DTN router module. We demonstrate the feasibility of our architecture by building a prototype using the DTN2 reference implementation. Our aim for a general-purpose solution is shared by DTN architectures for deep space exploration [15]. Further, we show how context can greatly influence the performance of network coded routing. This is, to our knowledge, the first integration of network coded routing and context-awareness. It is an important step as network coded routing was shown to be beneficial in DTNs [11, 21, 22].

2.2 Prior Work: Network Coded Routing

Previously we designed and implemented a network coding router called SimpleNC as an internal router for the DTN2 reference implementation [16]. SimpleNC splits large bundles into many non-encoded fragments, each labeled with a Globally Unique Identifier (GUID) derived from the original bundle. Rather than transmitting the non-encoded fragments, the router transmits linear combinations of them, called encoded fragments. Intermediate nodes can create new encoded fragments by forming linear combinations of existing ones (re-encoding). When a node receives *enough* encodings, it can reconstruct the original bundle. Optimized encoding and re-encoding schemes can substantially increase the amount of innovative content that nodes share.

In this paper, we extend SimpleNC to be a *Context-Aware Network Coding (CANC) Router* and focus on using context to dynamically adjust protocol behavior. This type of dynamic protocol reconfiguration is applicable to many types of routers. Specifically, in the next section, we describe a general purpose *Context Agent* (which is independent of the CANC Router) and a general purpose *Adaptation Portal* through which the Context Agent can reconfigure the internal router. We also implemented a prototype *CANC Context Agent* that provides context-aware dynamic reconfiguration to the *CANC Router*, and a *CANC Adaptation Portal* that connects the context agent to the router. We compare the performance of the CANC Router to SimpleNC to demonstrate one particular use of context within DTN routing and to showcase the potential benefits of context-awareness.



(a) Architecture (b) Implementation

Figure 1: Architecture and Implementation

3. INTEGRATING CONTEXT WITH DTN2

Figure 1 shows the software components in our context-aware architecture. The novel contributions of this paper are in the following three components:

- a *Context Agent* that is external to any bundle protocol implementation that assimilates and processes context information and influences routing protocol behavior; specifically we provide the *CANC Context Agent* that collects and processes context information for adapting a network coded router;
- an *Adaptation Portal* that exposes configuration hooks into a Bundle Router,
- a highly configurable *Bundle Router* that makes an extensive set of configuration parameters available to the Context Agent. In our prototype, we extend SimpleNC to create the *CANC Router*.

As we describe these components below, we provide both the general-purpose capabilities and the specific details of our CANC framework implementation.

3.1 Architectural Rationale

In our general architecture, we deliberately separate the Context Agent from any DTN-specific architecture. Given a concrete Adaptation Portal, a Context Agent can work with many DTN solutions.

DTN2's External Router API allows a separate process to receive most of the same events provided to an internal router and to make bundle (a unit of data in a DTN) forwarding decisions. In our architecture, the Context Agent is similar to an External Router. One major difference is an external router makes forwarding decisions related to individual bundles that are sent in a single contact. In contrast, the Context Agent does not make per-bundle routing decisions. It monitors the node's context and the router's status, and dynamically reconfigures the router to efficiently move data. Specifically, our architecture assumes that large bundles are transmitted over multiple contacts using encoded fragments that contain information from a larger application bundle. In this setting, the Context Agent's use of a request like **SendBundle**, which is part of the external

router interface, does not make sense. Instead, our Adaptation Portal provides a Context Agent with the configuration hooks necessary to dynamically reconfigure the non-bundle-specific send parameters of an internal bundle router.

3.2 CANC Router

We extended SimpleNC into the CANC Router by implementing algorithms to handle numerous configuration parameters. Both are implemented as internal routers in the DTN2 reference implementation.

There are several reasons to split routing functionality between an internal router and the Context Agent. First, we use an extension block to convey network coding information, such as the coefficient vector, which is easily accessed from an internal router but is not accessible to external routers. Second, with an internal router we can inject and send temporary bundles that are stored entirely in RAM. This allows forwarding nodes to quickly generate and send new linear encodings in outgoing bundles. Using the external router API would result in the payload being written to files at least two separate times per transmission. It is thus more effective to keep the network coding functionality in an internal router but provide extensive configuration hooks that an external process can manipulate. However, the XML-encoding and event-driven model of the external router API are elegant, so adding network coding functionality to the external router module is attractive future work.

3.3 Adaptation Portal

In general, an *Adaptation Portal* specifies the interface between a *Context Agent* (described below), which acquires and assimilates context information, and configuration hooks in the underlying Bundle Router, in the form of assignable parameters. Different implementations of the Bundle protocol will provide different mechanisms for this connection. Within DTN2, the most obvious option is to use the *TCL Command Line Interface*. Ultimately, any Adaptation Portal serves as a bridge over which information transits between the Context Agent and the Bundle Router (sending routing protocol parameters in one direction and DTN context information in the other direction).

CANC Adaptation Portal. In creating the concrete *CANC Adaptation Portal*, we identified three configuration hooks that are immediately useful for controlling the flow of encoding bundles when the underlying routing protocol is the *CANC Router*: weight, rate, and balance. Each parameter can be configured for each globally-unique identifier (GUID) and for each neighbor. The CANC Adaptation Portal tunes these parameters through the DTN2 TCL command line interface. Figure 2 illustrates the effects of these parameters.

- **Rate.** This controls how fast a node sends encodings to a neighbor relative to how fast the neighbor sends to it. For a rate $r \geq 1$, the node can send r encoding bundles for the specified GUID for every 1 it receives in return. For a rate $0 < r < 1$, the node can send an encoding bundle from the specified GUID when it has received at least $1/r$ from its neighbor. A rate of zero ceases sending, and a rate of -1 causes unconstrained sending.[1] In Figure 2, at the top, Node A chooses a higher rate to send to Node B than vice versa.
- **Weight.** If a node is carrying encoding bundles associated with multiple GUIDs, the weight parameter allows the Context Agent to bias the selection of which GUID's bundles to favor. In the SimpleNC router this selection was uniform, which was inefficient if one GUID had a much higher rank than another or if one GUID was much newer than another. Weight parameters are taken into account after GUIDs are checked for eligibility based on rate counters. The weights can be set using the configuration hooks or automatically based on relative GUID ranks. The example depicted in the center of Figure 2 shows a higher weight for GUID2 because its rank is higher.[2]
- **Balance.** If a node has two or more neighbors, the links share the same limited bandwidth. A router may want to bias the bandwidth to a particular neighbor; the balance parameter enables this. The balance parameter may be approximated by coordinating the rate parameters.[3] In the bottom of Figure 2, Node A biases the bandwidth to Node B. This may be because Node B is in an information-starved part of the network.

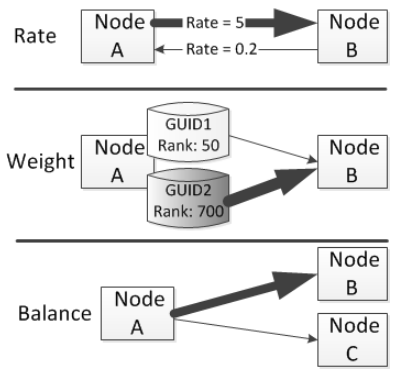


Figure 2: The configuration hooks for the CANC Router

SimpleNC is similar to flood router in trying to disseminate all bundles to all neighbors, which is unsustainable in most practical cases. The static table-based router is probably the most commonly used DTN2 router, and one could view these configuration hooks as a step towards making a network coding-aware static router. *CANC Router*'s adaptation hooks let us give directionality to the flow of data in the network, striking a balance between the approaches.

3.4 The Context Agent

A Context Agent aggregates context relevant to routing—this could include context explored in previous work like social contacts, link performance, contact patterns, mobility, or as in this work, spatial context like direction, speed, and DTN network data distribution. The Context Agent uses this to configure the Bundle Router through user-defined rules. Because context can come from the network stack or locally through device-specific processes, the Context Agent can define any number of context collection submodules. Our implmenetation, shown in Figure 1(b), defines two—one

[1]If nodes A and B have rates of $r_A > 0$ and $r_B > 0$ for a particular GUID, this algorithm is subject to deadlock unless $r_A \cdot r_B \geq 1$. We added a timer so that each node's send counter is reset every second, allowing it to send at least one encoding bundle per second, unless the rate is zero.

[2]We favor the GUID with higher rank because the sender has more information about that GUID and is more likely to be able to complete the receiver's entire application-level bundle. Alternative rationales for assigning weights are possible.

[3]An available MAC-layer broadcast Convergence Layer would make this parameter irrelevant.

for collecting router context, and one for collecting mobility context. A Context Agent also has an interface for controlling a DTN Bundle Router (the *Router Controller*). The *Route Context Aggregator* and the *Router Controller* communicate with the Bundle Router through the *Adaptation Portal* described above.

As the Context Agent collects and processes context, it generates a "world view" of the operating environment. This world view consists of geographically tagged samples of the various context items stored in a dynamically generated "map" of the network. This map (an example is shown in Figure 3) is split into cells, and each cell contains a collection of context tuples (i.e., {*type*:*value*}) observed at that location and annotated with a time stamp. Every node periodically shares its world view by broadcasting it, and when a node receives another's view, the two are merged according to a merge algorithm. This sharing and merging of views allows a Context Agent to approximate the global context. There are other possibilities for efficiently sharing context (aside from naïve periodic beaconing). Work in efficiently summarizing and sharing context is orthogonal to this work; a Context Agent can incorporate space efficient mechanisms for context representation, e.g., [8], that would significantly reduce the overhead of sharing world views.

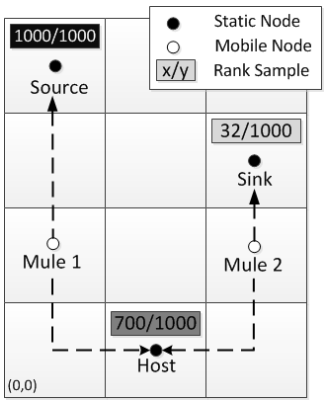


Figure 3: Context World View (with experiment nodes and waypoints)

The CANC Context Agent. In our current concrete implementation, we focus on context that represents the node mobility (i.e., their current location, their destination, and whether they are mobile or static) and the network coding router state (i.e., for each known GUID, the current rank of the decoding matrix and the source and destination of the bundle). With respect to context acquisition and world view generation, our concrete *CANC Context Agent* is general and independent of its use to adapt network coded routing.

Given our use of network coded routing, the routing process itself can be considered an information dissemination problem where the goal is to move data from where there is high information density towards the sink (which starts with zero information). With this in mind, the *CANC Context Agent* uses mobility and router context to set the *rate* at which a node will send encodings to a given neighbor. We implemented two variants of the rules, one that only considers the relative ranks of two nodes and one that also considers node mobility, as shown in Figure 4. Although this *relative rate* scheme is specific to network coding, in general, controlling the sending/receiving balance between a pair of nodes is a baseline control mechanism for many bundle routing protocols.

4. EXPERIMENTAL EVALUATION

To evaluate our architecture, we integrated our Context Agent with the DTN2 reference implementation and evaluated it in two real-world system environments: the Pharos mobile computing testbed [18] and the VMT channel-emulated testbed [6, 9]. [4]

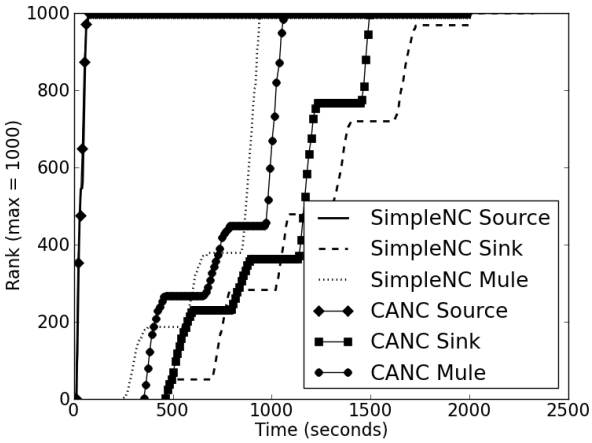


Figure 5: Three Node Robot Experiment

4.1 Pharos Testbed Experiments

The Pharos Testbed consists of autonomous mobile robots built around a four-wheeled chassis each with a Linux-based computer and Atheros IEEE 802.11g wireless radios. Mobility is accomplished through a testbed controller application that runs locally on each robot and navigates by following a line on the ground. Although the robots are capable of autonomous outdoor navigation, we performed our experiments indoors on a floor of the engineering building due to a long period of inclement weather preventing outdoor tests. We configured the testbed controller to provide each node's location and destination to the CANC Context Agent every 2 seconds using a local socket connection. Thus the testbed controller itself functions as the Host Context Provider.

Experimental Setup. We performed two experiments with the autonomous Pharos nodes, one using three nodes (*Source*, *Mule 1*, and *Host*), and one using five nodes (*Source*, *Mule 1*, *Host*, *Mule 2*, and *Sink*). See Figure 3 for a visual overview of their placement and mobility paths—the dimensions of the hallway were 25m by 42m; in all cases, only the mules moved (with speeds of 0.57 and 0.65 meters/second for *Mule 1* and *Mule 2* respectively). In the three-node test, we employed only the *Source*, *Mule 1*, and the *Host*; the *Host* node acted as the Sink. In the five-node test, *Host* was simply a stationary intermediate node. In both experiments the source generated a 100MB bundle at the beginning of the experiment, which it split into 1000 fragments to encode over. The effective wireless connectivity distance between nodes was around 10 to 20 meters, less around corners, and the *Source*, *Host*, and *Sink* were mutually disconnected for the duration of all the experiments despite their physical proximity. They could only send data between each other via the mules. We compared our CANC framework (CANC Router plus CANC Context Agent) with SimpleNC.

Three-Node Experiment Results. Figure 5 shows the results of the three-node experiment. The graphs shows the rank of the decoding matrix at each node vs. time. Although the SimpleNC mule reached full rank before the CANC mule,

[4]For complete details, see: `http://bit.ly/CANC12`

```
if $neighbor_rank == $max_rank:
→$rate = 0; # do not send to any full-rank node
else if $neighbor_eid == $sink_eid:
→$rate = MAXRATE; # unbounded rate to sink
else if $neighbor_rank == 0:
→$rate = MAXRATE; # neighbor has no encodings
else if $my_rank == $max_rank:
→ $rate = MAXRATE # I have full rank
else:
  # default case, use relative rates
→$rate = $my_rank/$neighbor_rank
```

(a) Basic rank-aware rules

```
if $neighbor_rank == $max_rank:
→$rate = 0; # do not send to any full-rank node
else if $neighbor_eid == $sink_eid:
→$rate = MAXRATE; # unbounded rate to sink
else if $neighbor_type == MOBILE AND
        $WorldView.destination.rank == $max_rank:
  # mule's destination has full rank, don't send
→$rate = 0;
else if $neighbor_rank == 0:
→$rate = MAXRATE; # neighbor has no encodings
else if $my_rank == $max_rank:
→ $rate = MAXRATE # I have full rank
else:
  # default case, use relative rates
→$rate = $my_rank/$neighbor_rank
```

(b) Extended mobility-aware rules

Figure 4: *Context Agent* Rules

the CANC sink reached full rank more than 1000 seconds before the SimpleNC sink. This was due to to the CANC Context Agent's control of the rates, which kept the mule from being overwhelmed by encodings from the sink (as happened in SimpleNC). CANC was able to significantly improve the overhead of network coding by intelligently limiting rates. SimpleNC resulted in a combined total of 5951 encoded fragment transmissions compared to CANC's 2149, making CANC almost three times more efficient. CANC achieved close to the absolute minimum number of transmissions needed, which is 2000 (1000 to the mule and 1000 to the sink).

Five-Node Experiment Results. The results from the five node experiment, graphed in Figure 6, also show that CANC outperforms SimpleNC. The ranks of the sources and mules are omitted for clarity. The CANC sink was able to reach full rank faster than the SimpleNC intermediate. The larger performance gain over the three-node experiment is due to the increased congestion at the intermediate, where three nodes (*Mule 1*, *Host*, and *Mule 2*) were often vying for the wireless channel simultaneously. Controlling the send rates in such a situation yielded even greater benefits than when only two nodes were connected at once, and we believe that as the number of neighbors grows, the benefits of adaptive rate control will grow as well. Similarly to the three-node experiment, CANC also provided massive overhead gains; it sent 7149 total encodings across all nodes, compared to SimpleNC's 16765—resulting in a 2.3 times efficiency gain.

4.2 VMT Testbed Results

We also ran several experiments on the VirtualMeshTest (VMT) mobile wireless testbed [6, 9]. VMT allows us to subject Linux-based real wireless nodes with commodity wireless hardware to emulated mobile environments. The wireless testbed is effectively an analog channel emulator based on an array of programmable attenuators. Given a desired physical arrangement of nodes, the system computes the expected path loss between nodes and programs the attenuators to achieve those path loss properties. By updating the attenuations every second, VMT can emulate a mobile wireless environment for real wireless nodes.

VMT Experimental Setup. We used the same three-node and five-node scenarios described in Section 4.1 with some minor changes. Since the effective range of each node was approximately 500m, the emulated distances had to be much greater to achieve disconnections; however the roles and

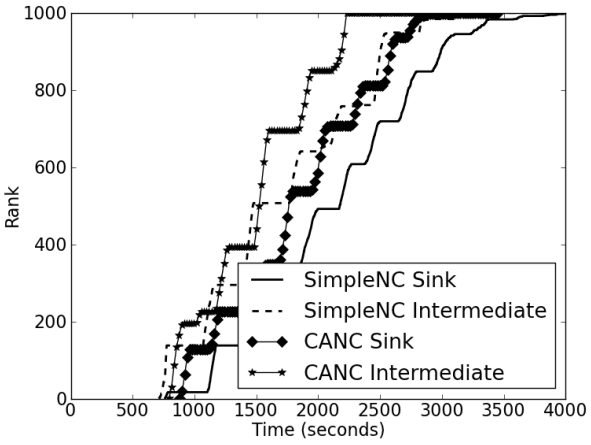


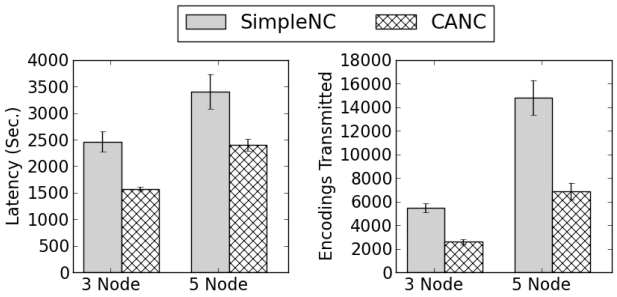
Figure 6: Five Node Robot Experiment

movements of the nodes remain the same. The stationary nodes were spaced 1200m apart (ensuring that the *Source*, *Host*, and *Sink* depicted in Figure 3 were mutually disconnect), and the mobile nodes (*Mule 1*, and *Mule 2*) moved between them at a rate of 10m/s. As with the Pharos experiments, the source created a 100MB bundle at the beginning of the scenario that it split into 1000 fragments to encode over, and as before we compared our CANC framework against SimpleNC.

VMT Results. Several runs of both the three-node and five-node experiments were averaged and Figure 7 shows the average time it took for the sink to receive enough encodings to decode the 100MB bundle (latency) and the average number of bundles transmitted across all nodes (overhead) in the network with the standard deviations. As was the case for the Pharos results, CANC resulted in a lower latency and much fewer total transmitted bundles than SimpleNC. It is interesting to note that although the same number of experiments were run for each router, the standard deviations are much smaller for CANC. This confirms that our CANC Context Agent's rate adaptation results in more stable and predictable behavior.

5. THE POTENTIAL FOR CONTEXT

In this work, we presented a general framework for enabling context-awareness in DTNs and demonstrated its use-



(a) Latency (b) Overhead

Figure 7: VMT Experiment Results

fulness. We limited our experiments to specific types of context (location, destination, rank, etc.) and currently only adapt a network coded routing agent's behavior in response to context. This demonstration provides a concrete instantiation of our general approach and showed that significant performance gains (in terms of bundle delivery latency and overhead) were possible. There are more expressive types of context, more powerful adaptation strategies, and many other DTN routers that could benefit from our architecture.

Path-Aware Geographical Context. Currently we only consider a data mule's destination to determine the appropriate rates. We could easily consider the entire intended path. Considering the whole path would allow the Context Agent to check the known information diversity in all of the locations in the path, and adapt its *rates*, *weights*, and *balance* to maximize the spread of information.

Predictive Context. A node's world view is currently only updated with real context samples taken by itself or by other nodes that with which it exchanges world views. Another useful extension would be to allow for context "prediction". For example, given a data mule (**A**) traveling a specific path (P), a node could "assume" a certain spread of bundles to the cells that P includes, proportional to the amount of time that **A** intends to spend passing through those cells. A node's world view could therefore be updated with predicted data even before real samples of context arrived from those cells.

Decaying Trust. Nodes could assign a "trust" or "certainty" metric to the context samples stored in the world views according to the timestamp of the context tuples. An older context sample would naturally be less trustworthy than a fresh sample, and the decay time could be assigned based on the relative mobility of the nodes in the network, which is easily sensed and estimated using, for example, network dynamics estimation [17].

6. REFERENCES

[1] C. Boldrini, M. Conti, F. Delmastro, and A. Passarella. Context- and social-aware middleware for opportunistic networks. *J. of Network and Computer Applications*, 33(5):525–541, Sept. 2010.

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

[3] B. Burns, O. Brock, and B. N. Levine. MV routing and capacity building in disruption tolerant networks. In *Proc. of Infocom*, pages 398–408, 2005.

[4] P. Costa, C. Mascolo, M. Musolesi, and G. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *IEEE J. on Sel. Areas in Comm.*, 26(5):748–760, June 2008.

[5] DTNRG. DTN bundle protocol ref. implm. `http://www.dtnrg.org/wiki/Code`.

[6] 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.

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

[8] C. Julien. The context of coordinating groups in dynamic mobile environments. In *Proc. of Coordination*, pages 49–64, June 2011.

[9] 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.

[10] J. Leguay, T. Friedman, and V. Conan. Evaluating mobility pattern space routing for DTNs. In *Proc. of Infocom*, pages 1–10, 2006.

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

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

[13] 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.

[14] M. Musolesi and C. Mascolo. CAR: Context-aware adaptive routing for delay-tolerant mobile networks. *IEEE Trans. on Mobile Computing*, 8(2):246–260, Feb. 2009.

[15] C. Peoples, G. Parr, B. Scotney, and A. Moore. Operational performance of the context-aware broker (CAB): A communication and management system for delay-tolerant networks (DTNs). In *Proc. of SPACOMM*, pages 128–133, June 2010.

[16] A. Petz, C.-L. Fok, C. Julien, B. Walker, and C. Ardi. Network coded routing in delay tolerant networks: An experience report. In *Proc. of ExtremeCom*, 2011.

[17] A. Petz, T. Jun, N. Roy, C.-L. Fok, and C. Julien. Passive network-awareness for dynamic resource-constrained networks. In *Proc. of DAIS*, 2011.

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

[19] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Proc. of Percom Workshops*, pages 79–85, 2007.

[20] Y. Wang and H. Wu. DFT-MSN: The delay/fault-tolerant mobile sensor network for pervasive information gathering. In *Proc. of Infocom*, pages 1–12, 2006.

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

[22] 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.