Application-aware DTN Routing

Jörg Ott <jo@netlab.tkk.fi> and Mikko Juhani Pitkänen <mikko.pitkanen@hip.fi>

ABSTRACT

Delay-tolerant networking (DTN) enables nodes to communicate by means of asynchronous messaging even though no end-to-end path may exist at any point in time. Messages may be of arbitrary size and the (email-style) store-carry-and-forward operation keeps copies of messages in possibly many nodes for an extended period of time. Suitably designed application protocols avoid frequent interactions and create self-contained messages instead. We leverage these properties and propose application support for DTN routers. By adding explicit application hints visible to intermediate DTN nodes the latter can selectively aid application performance, by means of application-dependent routing, cooperative caching, distributed storage, or even more complex operations.

1 INTRODUCTION

For many years, caching and (more recently) dynamic content adaptation techniques have been used for web applications in the Internet to serve users more efficiently, distribute load, and tailor results to each requestor. Of particular interest are wireless nodes for which Internet connectivity is often the bottleneck. With increasingly powerful wireless devices (particularly in terms of local storage and communication capabilities), cooperative caching [12] among mobile nodes in mobile ad-hoc networks gains potential in addition to the caching infrastructure offered by service providers.

However, traditional web caching is not very well suited for mobile (ad-hoc) environments: Caches, placed within the infrastructure or located on other mobile nodes, need to be "inserted" between the client and the origin server. This can be achieved by client (auto)configuration or by having the cache intercept IP packets transparently from the passing traffic (which requires that all packets follow the same route and the cache is on-path). Both is difficult to achieve in dynamic mobile environments. As request and response data are transmitted as a sequence of otherwise independent IP packets, caches need to terminate an application connections and reassemble, interpret, and process each application data unit (ADU).

The situation is different with *Delay-tolerant Networking* [1]. As application protocols cannot rely on highly interactive communications, protocol messages (*bundles*) are semantically self-contained (i.e., ADU = bundle). This means that intermediate routing nodes will often only operate on complete ADUs so that implementing further support functions such as caching will not require additional processing (such as reassembling application data). Moreover, assuming that bundles will usually be larger (at least not smaller) than packets, the arrival frequency is lower, rendering per-bundle processing less expensive than per-packet processing. Finally, depending on the routing protocol, DTNs may keep ADUs longer available due to extended storage periods and wider dissemination (when using replication).

These observations motivate using DTN nodes for (mobile) storage and retrieval. In this paper, we leverage bundles queued in DTN routers for the purpose of caching and distributed storage platform but also consider generalizing these concepts to support further application protocol operations. After a review of related work in section 2, we introduce application scenarios in section 3 and outline application support in DTN routers in section 4. We show first simulation results for information caching in section 5, discuss limitations and some open issues in section 6, and conclude with a short assessment and discussion of next steps in section 7.

2 RELATED WORK

DTN has been studied to enable communication in disconnected or challenged environments, including sparse mobile (ad-hoc) networks. Communication in DTNs is asynchronous, modeled after email. Messages may be stored for an extended period of time (rather than discarded) if no end-to-end path exists and may also be carried physically during this period, thus extending communication capacity and reach. In the DTN architecture [1] different (inter)networks are interconnected above the transport by means of a *bundle layer*. Bundles are forwarded hop-by-hop between DTN nodes (*bundle routers* or *agents*). As bundles may be of arbitrary size, fragmentation is supported to deal with storage or transmission constraints. A *custody transfer* imposes persistence requirements on bundle storage in and reliability on forwarding across bundle agents. A bundle lifetime set by the sender limits each bundle's circulation. Administrative bundles may inform a sender about message forwarding progress, end-to-end delivery, or failures.

A DTN routing framework has been presented in [3]. Variants of epidemic routing have been proposed (not just) for mobile environments (e.g. [5]). These routing protocols create multiple copies of a message and choose different paths to increase the delivery probability (using different algorithms for creating, forwarding, and purging copies of messages). Furthermore, erasure and network coding techniques have been suggested [2, 10, 11] to increase the delivery probability while reducing the overall load on the network.

The delay-tolerant nature of DTNs enables more sophisticated membership and delivery models when dealing with multicasting [13]. The *store-carry-and-forward* paradigm is exploited for (application-layer) broadcasting to spread information from a few feeds to many recipients [4], leveraging DTNs for content distribution. In contrast to traditional Content Delivery Networks (CDNs) that use dedicated infrastructure, with DTNs, the user nodes play an active roles in dissemination.

Our recent work on Grid data storages has presented a design for a scalable erasure coding storage showing how to efficiently stripe files into large chunks for redundancy [8] and how the coding can be used with large data units comparable to size of bundles [9].

In this paper, we generalize and extend these ideas and propose tapping into the DTN layer to leverage *and influence* storage and propagation in DTNs in order to support application operations. We start by looking at storing, disseminating, and retrieving self-contained ADUs, but also investigate supporting further operations. In all cases, each DTN bundle carries processing hints to influence its treatment by the DTN nodes.[1]

3 Application Scenarios

We assume that the application protocols are designed for or adapted to be used with DTNs (as proposed for HTTP [6] or email [7]). We start with two specific scenarios concerning retrieval and distributed storage, then look at more general protocol operations, and finally summarize common requirements for DTN support. We focus on realizing the respective protocol functionality and defer the discussion of security issues to section 6.

3.1 Implicit Content Caching

A mobile user wants to *retrieve* web page resources available on the Internet. A web page request $req(U)$ from a requester R contains an HTTP GET message with a URI identifying the resource U sought; header fields may indicate R's relevant capabilities and conditions for resource retrieval. $req(U)$ travels from R intermediate nodes N_i the (fixed) origin server which returns a response $rsp(U)$ containing the complete resource U in question plus a set of descriptive attributes (e.g., its modification date and validity period) or an error notification. Both $req(U)$ and $rsp(U)$ may be replicated several times on their way through the DTN and stored temporarily, at most until their respective lifetimes have expired.

[1]This might be viewed as resembling Active Networking to some extent. In comparison, in DTN-based operation, processing takes place per (large) self-contained bundle rather than per (small) independent packet (and thus less frequently) and DTN nodes will often be user devices rather than infrastructure components.

Depending on the routing algorithms in use, both $req(U)$ and $rsp(U)$ may traverse several paths and multiple copies may be spread across the network. When another client S later also issues a request $req'(U)$ for U, depending on the proximity of R and S, the motion of nodes N_i, and the respective paths chosen by the routing protocol, $req'(U)$ may traverse a node N_j that still holds a copy of $rsp(U)$. In this case, N_j could act as an *implicit cache*, create $rsp'(U)$ from its stored U, and reply to S (not forwarding $req'(U)$). The prerequisite is that N_j is able to identify U, to match $req'(U)$ and $rsp(U)$, and to determine if the freshness requirements from the request are satisfied by $rsp(U)$. If response bundles are stored even longer than necessary for DTN delivery (e.g., space permitting, for the validity period), a cooperative cache can be created among mobile users by leveraging delay-tolerant routing mechanisms. Note that this content caching is different from the content delivery mechanisms presented above since, here, we wait for explicit requests rather than distributing resources proactively.

3.2 Content Storage

The above scenario can be extended to cover *store* operations as well. Assume a (mobile) user T traveling who wants to maintain a blog about her trip and archive her digital pictures, making the blog and a subset of the pictures also available to other travelers who might be in the area as an instant travel guide. T uses some third-party provider in the fixed Internet that accepts DTN-based delivery of contents, generates blog entries, and archives images as specified; all bundles received and successfully processed by the service provider are explicitly acknowledged end-to-end so that the traveler knows when she can delete the information locally.

The storage request $srq(U)$ identifying and containing the resource U will propagate towards the fixed Internet, again, being stored, forwarded, and replicated possibly along multiple paths on nodes N_i. Replication and extended storage on other intermediary nodes can be used to make the U available to another peer P who is interested in this information; its request $req(U)$ can be handled as described above. This very mechanism may also serve as a (short-term) distributed backup storage for user T as multiple copies exist and may prevent information loss in case of hardware loss or failure—or for storage in a DTN for the entire lifetime of U. In both cases, T may choose the lifetime for the $srq(U)$ bundle to match the desired period of (local) availability rather than just the expected time to reach the fixed "origin" server. Explicit *cleanup* bundles may be used to force purging contents once it is no longer needed.

3.3 Further Operations

The above examples show protocol operations acting on complete resources and delivering these in a DTN. Further operations maintaining the integrity of such complete resources are conceivable: *Partial resource retrieval* is straightforward by allowing common subaddressing schemes (such as the HTTP Range: header). *Subscription and notification* services could be realized that allow nodes to monitor known resources for updates. DTN multicasting could be used for efficient distribution with multicast EIDs representing certain resource collections (e.g., content-based or source-based "channels"). Assuming some hierarchical structure of resource identifiers (such as web pages), prefix-based subscriptions would allow efficiently monitoring huge resource collections (e.g., all objects of a web site). *Content transformation* may be performed for a specific requesting node that supplies its own capabilities in the request. Another node will only respond if it is capable of performing the necessary content transformation. As a fallback, the request can be routed to the origin server.

Operations that manipulate or update parts of a resource might also be defined; however, this raises synchronization and authorization issues (see section 6).

3.4 Common Requirements

From the above scenarios, we can extract some commonalities that a DTN bundle router instance could use as hints for tailored processing of bundles. Tailored processing may range from noticing the presence of a particular bundle (e.g., carrying a resource) to modifying store and forward operations to replying to received bundles as we will discuss further in the next section.

First of all, bundles that could benefit from enhanced treatment need to be identified as such. Somewhat similar to the IP *router alert* option, a bundle should be tagged if special handling beyond "fast path" is desired.

Independent of a particular application protocol, the general *class of operation* should be identified to allow for minimal support even if the application protocol details are not understood: it should be visible whether a bundle carries a resource (retrieved or to store) as opposed to just a retrieval request, storage or error response. This allows for different storage and routing policies. Obviously, the *resource* the protocol operation is to act upon needs to be identified independent of the application, typically by using a URI, so that at least content-specific indexing is possible.

If a resource is contained in the message, its *application layer validity period* (lifetime) should be made explicit, so that bundle storage may be adapted accordingly. For more advanced processing, its *content type* and possibly other (MIME) parameters might be provided.

Finally, the *application protocol* should be identified to allow for dedicated handling so that, e.g., specific responses to retrieval requests can be generated. Processing steps requiring further details can then interpret the application protocol and react accordingly.

This common information may be carried in a bundle protocol extension header, *Application-Hints*, providing the following fields: resource URI (including the application protocol), operation indicator, resource lifetime, and possibly further resource-specific parameters. Such a header can be encoded in a few bytes plus the URI and optionally the MIME type and thus will increase the bundle size only marginally. In case of bundle fragmentation, this header needs to be copied into every fragment.

4 Application-aware DTN Routers

Figure 1 shows a simplified diagram of a sample DTN router: a bundle—comprising a DTN header denoted by "H" and a payload—arrives on an incoming interface on the left. After some validity checking (e.g., authentication, TTL) the bundle is passed on to the routing logic where loop detection is performed, a forwarding decision is taken, and the bundle is placed in one or more of the outgoing queues. The routing decision may be revisited whenever a new contact/link becomes available and queues may be updated as new bundles arrive. The bundle is transmitted when a contact/link becomes available, logically preceding bundles have been sent, and the peer accepts the bundle. After transmission, a bundle may be kept further in the same or a different queue, e.g., to await further transmission opportunities to other contacts. A bundle may be purged after successful transmission, reception of a delivery or custody transferred notification, or according to some deletion strategy (e.g., oldest bundle, shortest remaining lifetime, random).

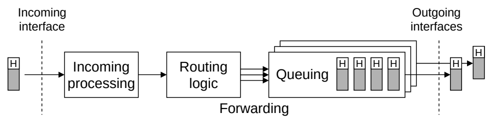

Figure 1: Simplified view of a DTN router

Bundles may stay in router queues up to their TTL. Routing protocols may perform partial or full replication of bundles at different stages: only the sender may create copies, only *n* copies in total may be created, or bundles may forwarded only a fixed number of times (before the destination is met), to name just a few. In all cases, bundle routing/forwarding as well as deletion decisions are taken based upon the bundle headers only. However, these headers indicate transmission parameters but do not necessarily reflect properties of the bundle contents. In the following, we introduce two extension steps to bundle routers based upon the *Application-Hints* header proposed above: a passive and an active mode of operation.

4.1 Passive Router Support

With passive support, the DTN router does not alter its own behavior concerning bundle routing. Merely, as depicted in figure 2a, an additional *application-aware* *processing (AAP)* module gathers information upon every bundle event occurring in the router: about newly arriving bundles (*info*) containing an *Application-Hints* header "A" and when these bundles get *add*ed to and *delete*d from queues. By inspecting the *Application-Hints*, the module maintains a local context from the application perspective. This context allows the AAP module, e.g., to locate a local copy of a resource U in an $srq(U)$ or $rsp(U)$ matching an incoming $req(U)$. If found it may *retrieve* the resource from a queue, use an application-specific component to generate a *reply* bundle, and enqueue it for delivery to the requester

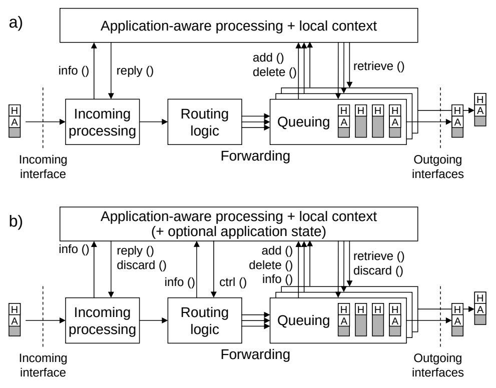

Figure 2: Passive (a) and Active (b) Support in a DTN Router

4.2 Active Router Support

A passive AAP module cannot influence decisions of the local router, which has the benefit of minimizing interference with the operation of DTN routing protocols. In some situations, however, such an influence may be desirable: In a simple example, an AAP module that serves repeated $req(U)$ for a resource from the local queue may want to prevent the bundle containing $rsp(U)$ from being deleted and keep it until its application-level validity period expires. Similarly, an AAP module may speed up deletion of resource bundles that are no longer needed or may treat bundles containing different (classes of) operations differently. Furthermore, an AAP module may supply routing hints or forwarding decisions per application and thus enable different routing schemes to be active at the same time—a feature which could be used to construct overlays as part of the regular routing mechanisms. Finally, this may even allow a mobile node to exercise better control of its local capacity by selectively forwarding bundles for some applications (which the user is interested in) while not accepting them for others.

For this active control, further interfaces are added as shown in figure 2: The AAP module may *discard* incoming bundles. Queuing is extended to keep the AAP module *info*rmed about manipulations (including moving bundles within or between queues) and enable the AAP module to perform similar operations (*ctrl*) as well as to *discard* queued bundles. These operations allow an AAP module speeding up spreading of some bundles while delaying or preventing further dissemination of others. The AAP module now also interacts with the routing logic, receiving information (*info*) about routing tables but also being able to supply such information via the *ctrl* interface.[2] This enables controlling the degree of replication, the choice of outgoing links, or the amount of erasure coding being applied per bundle to match application needs. For example, if a DTN serves as (temporary) storage for certain resources according to scenario 2 above, a higher degree of replication is desirable for storage than for retrieval requests.

In all their decisions and influences, the application modules do not have to strive for absolute correctness or reliability as in a distributed system but may rather operate probabilistically: those requests that cannot get resolved or found locally will have to travel to the origin server. This means that AAP modules can be optimistic in their behavior to achieve their goals and conservative with respect to their impact on the network's operation.

4.3 Generic vs. Specific Operation

In both passive and active mode, the question arises of how much generalization can and should be applied and where application-specific support is absolutely needed (and how much application knowledge should go into a router in the first place). Any router actions that do not generate new bundles (with modified contents) or alter existing ones can operate independently of the application protocol, yet may support selected applications, e.g., by influencing routing decisions or queuing.

It is obvious that a DTN router must understand an application protocol to respond to requests or perform even more sophisticated operations. However, there are some fairly general protocols (such as HTTP) that make up a large fraction of the Internet traffic today so that very limited additional effort may already yield significant gain. As the application protocols can be identified from the hints in a bundle, their processing can be dispatched to specific *plug-ins* that may be added and removed dynamically. Again, since enhanced DTN router modules operate probabilistically, applications will still work even if their bundles are not recognized by some or all of the nodes. This also helps incremental deployment.

5 SIMULATIONS AND OBSERVATIONS

We have carried out a first set of simple simulations to show the effect of caching bundle contents in routers and present some initial findings on the use of application-aware routing combined with caching in bundle based message delivery. As performance metrics, we look at the delivery ratio, response time, and network utilization.

[2]A bundle router API is currently under discussion in the DTNRG.

We have extended a Java-based DTN simulator [3] to support the functionality needed for our application-aware routing: bundle-based transfer, with each bundle identified by a URI; caching bundles on the routing nodes with a certain probability; bundle forwarding without fragmentation with incomplete transfer leading to a retransmission; bundle fragmentation with bundles being send in chunks in a way that each chunk remains to be self-contained and identifiable. For bundle forwarding, we use flooding implemented in the simulator.

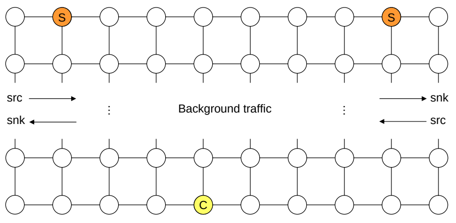

Figure 3: Simple grid topology used in simulations.

Figure 3 depicts the network topology used in our simulations. We have chosen a simple structure of the topology to avoid biases. The client (C) is situated at one edge of a 10x10 grid topology and two servers (S) are located at the opposite side. Nodes connect to their neighbors over a WLAN link at 11 Mbit/s. The link behavior simulates a highly intermittent connectivity pattern where up and down times are exponentially distributed with mean of 16 s for the uptime and a mean of 4 s for the downtime, thus letting our target bundles pass but also leading to transmission failures. Link delays are static at 100 ms and each node contributes 500 MB of queue space. Four different URIs $U_1 \ldots U_4$ are requested every 50 s and both requests and responses time out after 24 s; responses may be cached for 480 s by intermediate nodes. We have run simulation over 3 hours of simulation time with background traffic of 2 MB bundles sent in ten second intervals from two sources ("src") to two sinks ("snk") located in the middle two grid rows on the left and right.

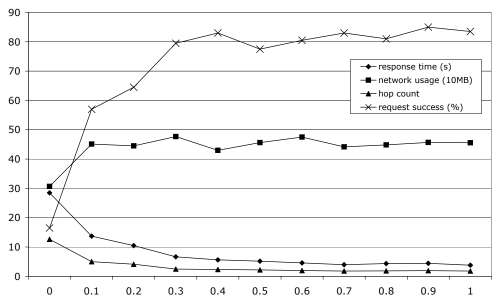

Figure 4: Probabilistic application-aware caching of resources

Figure 4 illustrates the effect of caching messages beyond forwarding with an increasing caching probability (used by each intermediary to determine independently per unique message whether or not to cache). Network usage shows the average message volume in the network queues when responses arrive at the client. With increased caching probability, an the number generated responses grows if a request reaches multiple independent caching nodes.[3] The request success rate shows the success rate of 200 requests being responded to by the server or a caching intermediary. With increasing caching probability, the success rate greatly increases and average time to get a response strongly decreases. The need for requests to travel shorter distances in a lossy network explains the increased performance which is confirmed by the response message hop count.

We note that even small caching probabilities lead to significant performance improvements. Above 30 %, there is little extra gain and above 10 %, there is virtually no extra storage penalty. In this scenario, the delivery ratio does not exceed 85 % and queuing requirements are quite immense, both of which can be addressed by different routing protocols. Particularly the latter also hints at the need for application-aware duplicate detection as two identical responses from different caches are different messages from a bundle layer perspective. A further analysis of the hop count revealed that 75 % of the responses came from less than 3 hops away.

We have compared these figures against two fixed caches placed in the middle rows of the grid: the resulting performance was only marginally better than without any caching (the impact of cache placement has not yet been investigated). We have also simulated (a) selective support per application class where a node only caches a certain resource U_x chosen at startup and (b) additionally 50 % of the nodes not cooperating at all. The results are roughly similar to those obtained for 20-30 % caching as shown above with (a) performing slightly better.

6 Discussion and Open Issues

Adding application-awareness to DTN routing raises numerous issues: should intermediate systems comprising "the network" mess with application information units instead of just forwarding them? If so, where shall we draw the borderline? And which technical issues need to be overcome in the real world?

As discussed in the introduction, we believe that DTN-based communication is much more suitable for inspecting and processing bundles in intermediate nodes than are individual IP packets, provided that we do not make end-to-end communication dependent on anything but regular bundle forwarding in the DTN. Providing explicit

[3]Because of the lossy network, we do not drop requests once they were responded to and also do not perform duplicate detection on bundles from different sources. Significant fine-tuning appears possible here.

hints about application contents also appears more sensible than second-guessing applications (e.g., from port numbers or communication patterns) so that particularly mobile users can choose which contents to process and forward and which to discard—which may also serve as a basis for future incentives to mobile users to carry traffic for others. So far, we also believe that application-specific extensions should be simple and so should be the supported operations. This rules out operations requiring synchronization of concurrent write access to resources as we do not see router support as a way to build distributed applications. We deliberately restrict our general application hints to a common minimum (and also rule out conveying code).[4]

For practical operation, as with any caching but particularly relevant to cooperative environments, applications may need to authenticate the origin and ensure the integrity of the data received. If DTN routers shall be able to generate replies, the protection of the resource contained in a bundle must be independent of the authentication of the entire response bundle. As operations and associated resources are spread and stored across the DTN, resources may need protection against unauthorized access, which can only be achieved by means of encryption. Finally, DTNs are susceptible to all kinds of DoS attacks: senders may lie about bundle contents to obtain better treatment, intermediate nodes may maliciously drop bundles, or nodes may simply flood a DTN with bundles, tagged as arbitrary or one particular application to disturb the regular operation of the network through congestion. However, most of these security issues are not specific to the application-awareness proposed in this paper, but need to be solved for DTNs and particularly DTN-based MANETs in any case.

7 Conclusion and Next Steps

In this paper, we have suggested to take advantage of two DTN features—namely bundles being self-contained ADUs and their extended storage period and distribution—to provide additional support for applications in DTN bundle agents. This allows generic functions such as bundle routing to be performed differently per application or resource, but particularly allows more sophisticated support such as caching, distributed storage, and further functions to be built on top. The proposed extension concept is incrementally deployable and robust as the regular network operation does not depend on any nodes to support a specific application.

From some initial simulations we have carried out for cooperative caching, we observe that a small fraction of DTN nodes cooperating at the application layer suffices to increase caching performance; future investigations will need to look at additional setups (including mobility), different DTN routing protocols, and erasure coding. We will also look at further application protocol operations and particularly investigate the performance and persistence characteristics of a distributed storage. An important aspect is the potential for "feature interaction" of plain and application-aware DTN routers in case the latter actively influence DTN routing (e.g., perform content-based routing). These investigations should help characterizing the space in which adding application-awareness to DTN routing is applicable.

[4]If an application wants to benefit from communicating code, this should be part of the application protocol portion and be validated and executed in the context of application-specific processing.

References

[1] V. Cerf, K. Fall, et al. Delay-Tolerant Network Architecture. Internet Draft draft-irtf-dtnrg-arch-05, Work in progress, 2006.

[2] S. Jain, M. Demmer, R. Patra, and K. Fall. Using redundancy to cope with failures in a Delay Tolerant Network. In *ACM SIGCOMM 2005*.

[3] S. Jain, K. Fall, and R. Patra. Routing in Delay Tolerant Networks. In *ACM SIGCOMM 2004*.

[4] G. Karlsson, V. Lenders, and M. May. Delay-tolerant Broadcasting. In *ACM SIGCOMM CHANTS Workshop 2006*.

[5] A. Lindgren and A. Doria. Probabilistic Routing Protocol for Intermittently Connected Networks. Internet Draft draft-lindgren-dtnrg-prophet-02, Work in Progress, 2006.

[6] J. Ott and D. Kutscher. Bundling the Web: HTTP over DTN. In *Proceedings of WNEPT 2006*.

[7] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking Connectivity in Developing Nations. 37(1):78–83, January 2004.

[8] M. Pitkanen, J. Karppinen, and M. Swany. Erasure codes for increasing the availability of grid data storage. In *HPDC06: Proceedings of Workshop on Next-Generation Distributed Data Management*, 2006.

[9] M. Pitkanen, R. Moussa, M. Swany, and T. Niemi. Erasure codes for increasing the availability of grid data storage. In *Proceedings of ICIW06*.

[10] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure Coding Based Routing for Opportunistic Networks. In *ACM SIGCOMM WDTN 2005*.

[11] J. Widmer and J.-Y. L. Boudec. Network Coding for Efficient Communication in Extreme Networks. In *ACM SIGCOMM WDTN 2005*.

[12] L. Yin and G. Cao. Supporting Cooperative Caching in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, January 2006.

[13] W. Zhao, M. Ammar, and E. Zegura. Multicasting in Delay Tolerant Networks: Semantic Models and Routing Algorithms. In *ACM SIGCOMM WDTN 2005*.