2011 31st International Conference on Distributed Computing Systems

Peer-to-peer Data Replication Meets Delay Tolerant Networking

Peter Gilbert
Duke University
Durham, NC
gilbert@cs.duke.edu

Venugopalan Ramasubramanian
Microsoft Research
Mountain View, CA
rama@microsoft.com

Patrick Stuedi
Microsoft Research
Mountain View, CA
pstuedi@microsoft.com

Doug Terry
Microsoft Research
Mountain View, CA
terry@microsoft.com

***Abstract*—Work on delay tolerant networks (DTN) and peer-to-peer replication systems has made rapid advances with the similar goal of permitting reliable message delivery in challenged communication environments. Techniques developed for both types of systems also bear some similarity. Both exploit opportunistic connectivity to route messages and updates to their desired destinations while making minimal assumptions about end-to-end connectivity. However, they also have some unique characteristics. DTNs have recently incorporated forwarding algorithms that make use of historical information on past encounters between participants and predictions of future connectivity. Modern replication systems utilize protocols with low overheads that guarantee eventual consistency and at-most-once delivery while supporting content-based filters.**

In this paper, we show how a DTN-like messaging system can be readily built as a simple application on top of a peer-to-peer replication platform. To reduce delivery delays while retaining the desirable replication guarantees, we then extend the replication substrate to permit pluggable DTN routing protocols. We describe the implementation of four representative DTN schemes as replication policies and evaluate these extensions with emulations driven by traces of e-mail messaging and vehicular mobility. We conclude that DTNs and replication systems can benefit substantially from a cross-fertilization of ideas.

I. Introduction

Challenged networks, communication infrastructures with links that experience extraordinarily long delays or highly intermittent connectivity, have been a focus for both networking and systems researchers over the past decade. Such networks arise in military settings, mobile sensor systems, and rural settings that lack fixed and reliable network providers. Communicating parties must rely on ad-hoc wireless connections between mobile devices, buses or even animals to transport data.

To provide an effective messaging service over challenged networks, researchers have developed delay tolerant network (DTN) architectures [7]. Work on DTNs has focused on policies and techniques for routing messages in situations where end-to-end connectivity may not exist between senders and recipients, and hence, messages must be routed through intermediaries in a delay-tolerant manner [9]. At the same time, peer-to-peer replication (PPR) techniques have been devised to manage shared data in network-challenged environments [4] [13]. This work has focused on replicating shared data items between intermittently connected nodes and reliably propagating updates to ensure eventual consistency. We observe that many similarities exist between the problems faced and the solutions employed in these independently developed technologies.

DTN-based message systems strive to deliver a message from a sender to a specific recipient or possibly a set of recipients. Similarly, PPR systems deliver updated data items to a select set of nodes, specifically those nodes that store replicas of the updated items. In both cases, messages or updates are propagated in a hop-to-hop manner via an overlay network. DTN protocols can operate over a variety of overlay topologies. Similarly, PPR systems are designed to be topology-independent. DTNs do not assume end-to-end connectivity between senders and recipients or bounds on message delays. Similarly, PPR systems permit disconnected operation and exploit opportunistic connectivity to transfer updates. While DTNs do not necessarily guarantee that a message will be reliably delivered, they do include mechanisms to increase delivery probabilities, such as per-hop acknowledgements and multiple-copy routing. Similarly, state-of-the-art PPR systems include techniques to guarantee that updates fully propagate to all interested nodes, thereby ensuring that replicas eventually reach a mutually consistent state. Many replication protocols, for example, maintain per-replica "knowledge" that serves as a type of vector-based acknowledgement scheme.

The main contributions of this paper are three-fold:

1) We show that a simple DTN-like messaging system can be readily built on top of a PPR system. In particular, we implemented reliable messaging as an application of the Cimbiosys replication platform. In short, messages are the data items that are replicated between nodes, and Cimbiosys's node-specific filters are used to deliver messages to the appropriate destinations. The nice properties guaranteed by Cimbiosys, notably eventual consistency, exactly-once update delivery, and compact metadata overhead, are then automatically realized by our messaging application. Essentially, this demonstrates how DTNs can benefit from techniques utilized in PPR systems.
2) Although our DTN application on Cimbiosys is fully functional, it suffers from long delivery delays for the

1063-6927/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCS.2011.64

connectivity and workload traces that we studied. We show that routing policies taken from previous work on DTNs can be incorporated as simple extensions to Cimbiosys. We have implemented and experimented with four common DTN routing schemes: Epidemic routing, Spray and Wait, PROPHET, and MaxProp. Essentially, this demonstrates how PPRs can benefit from DTN techniques.

3) We present an evaluation of our DTN messaging application running on Cimbiosys. This evaluation, driven by real workload traces, quantifies the delay benefits of the various DTN routing schemes. Experiments also clarify the trade-offs between delivery delay, message traffic, and storage overheads.

In the next section, we provide more background on previous work on DTN and PPR systems. Section III then expands on the similarities between such systems. Section IV discusses details of how we built a DTN application on Cimbiosys. Section V goes on to describe the extensions that were made to Cimbiosys to incorporate DTN routing policies. Section VI presents the results of our experiments to evaluate these routing schemes in this context. We then summarize our overall conclusions in Section VII.

II. Background

A. Delay or disruption tolerant networks

Delay tolerant networking (DTN), also sometimes called disruption tolerant networking, involves communicating in highly-partitioned networks where contemporaneous end-to-end paths may never exist. There is interest in deploying a wide variety of applications in DTN environments: using mobile nodes as data mules to provide Internet connectivity in areas with no existing infrastructure [2], military ad-hoc networks in hostile environments, animal tracking sensor networks [12], and certain delay-tolerant Internet applications like email, news services, and web search [3]. The opportunistic nature of communication in these scenarios presents further complications, as future contact patterns among nodes in such networks may be hard to predict. At the foundation of DTN research is the idea that traditional network architectures and routing algorithms, which require connected end-to-end paths, are not well-suited for these scenarios.

Initial work on DTN focused on constructing a network architecture suitable for networks in which traditional assumptions such as the existence of end-to-end paths, relatively small round trip times, and low drop rates do not hold. Fall proposed a DTN architecture consisting of an overlay above existing transport layers described as an "internetwork of challenged networks" [7]. DTNs are seen as a number of internally-connected islands or regions connected to each other intermittently through gateways. Given this kind of network architecture, traditional routing algorithms are ineffective. Instead of choosing the next hop for a given message among available neighbors, DTN nodes must choose which messages to transmit during potentially brief periods of connectivity.

An important parameter of the DTN routing problem is the amount of information about the network that is available to the routing algorithm. It has been shown that a Linear Programming model can be formulated to compute an optimal store-and-forward solution for DTN routing given that schedules of future contacts and future network load are known [9]. However, these assumptions generally do not hold for practical DTN scenarios, many of which involve only unscheduled meetings among nodes. Because an optimal store-and-forward solution does not exist for most DTN scenarios, practical approaches must rely instead on multiple-copy routing techniques to improve delivery. Routing protocols commonly choose to flood each message within the network and control the extent of flooding using heuristics, such as an estimate of the likelihood that a host will eventually come into contact with the intended recipient. While this allows multiple paths to be exploited in delivering the message, reliable delivery is not guaranteed.

Because flooding results in duplicate transmissions, flooding-based DTN protocols must employ mechanisms to avoid excessive overhead. Typically, DTN protocols use a variation on one of the following techniques: store identifiers of recently seen messages and compare this information with a communication partner before exchanging messages, or append the ID of each host which forwards a copy of a message to a hoplist included in the message. DTN routing protocols frequently use ad-hoc implementations of common mechanisms such as flooding and message histories to try to ensure properties such as reliable delivery and duplicate suppression.

B. Peer-to-peer filtered replication

To address the problem of managing shared data across intermittently connected networks, the systems community has proposed the model of *weak consistency* or *optimistic* replication [14]. Systems providing weak consistency allow lazy propagation of updates to replicated data items. A consequence is that reads to replicated data items may return inconsistent values, potentially violating assumptions made by applications expecting stronger consistency guarantees. However, many applications can handle the possibility of inconsistent reads, and weakly consistent replication systems have been shown to provide high availability while achieving good scalability and good read/write performance.

Peer-to-peer replication (PPR) systems are weakly consistent systems in which updates are propagated among hosts through pairwise exchanges, allowing the system to tolerate intermittent and ad-hoc connections between nodes. A *replica* hosted on a device stores some or all of the items in a collection of data. Applications (and their users) may

update, insert, or delete items from a local replica at any time, regardless of the availability of network connectivity and without coordinating with other nodes. When a link is available between two hosts, they exchange updated, inserted, or deleted items through a peer-to-peer *synchronization* operation, occasionally referred to simply as a *sync*. State-of-the-art PPR systems are *topology-independent* [4], meaning that each host need not synchronize directly with all other hosts, though paths of pairwise synchronizations must occur between all pairs of replicas in order to ensure eventual consistency.

While the majority of PPR systems replicate entire file directories or entire databases, some do support *partial replication*. To accommodate users' diverse interests, as well as devices that differ widely in their communication and storage capabilities, such systems allow replicas to receive and store only a subset of the data items in a large collection. In this paper, we are particularly interested in a class of PPR systems that permit the set of items stored in each replica to be defined by a host-specific, content-based *filter*, a query-like predicate over the contents of data items. Cimbiosys [13] is the prototypical example of such a *peer-to-peer filtered replication* (PFR) system, and is the replication platform used for the experiments reported in this paper. Cimbiosys ensures *eventual filter consistency*, the guarantee that all updated items will eventually be received by all hosts whose filters select the corresponding item.

A key feature of PFR systems is that they ensure eventual filter consistency while minimizing the overhead of synchronization operations. During synchronization, a host receives only items that match its filter and for which it does not already store a current or more recent version. To ensure this property of *at-most-once delivery*, each host in Cimbiosys maintains a data structure representing the set of known versions called *knowledge*. Knowledge is represented in a compact form, as a version vector, with size proportional to the number of replicas rather than the number of items in the system. While performing a synchronization, hosts exchange their current knowledge and use it to determine which locally stored versions of data items are unknown to, and hence should be transmitted to, the remote sync partner.

III. DTN AND REPLICATION

The work described in this paper was motivated by the observation that fundamental connections exist between problems considered by the research areas of data replication and delay tolerant networks. In previous work we suggested the duality between message routing protocols and peer-to-peer replication protocols at a high level [8]. In this section, we expand on that discussion. In particular, we observe that a number of key properties sought after by DTN approaches correspond to properties provided by PFR systems. These properties make PFR systems attractive as a potential substrate for building DTNs. We describe the applicable properties of PFR systems relative to four key properties desired in DTNs: *disruption tolerance*, *reliable delivery*, *selective delivery* and *duplicate suppression*.

- **Disruption tolerance and disconnected operation.** The ability to operate in the presence of intermittent network links and frequent disconnections is a fundamental requirement of DTNs. Like DTNs, weakly consistent replication systems are designed to tolerate network instability and failures. PFR systems support disconnected operation, allowing hosts to continue to access replicated data items when disconnected from other replicas. A host may read or write existing items as well as insert new items into the local replica at any time, regardless of connectivity to other hosts.
- **Reliable delivery and eventual consistency.** DTNs would like to ensure, but do not always succeed in guaranteeing, that messages are reliably delivered to their destinations. This corresponds to the property provided by PFR systems of eventual filter consistency—the guarantee that each data item will be eventually replicated at all interested hosts.
- **Selective delivery and partial replication.** A key property of PFR systems relevant to message routing systems like DTNs is support for partial replication, or the ability of a host to replicate only a subset of the items in the system. Hosts indicate which updated items they wish to receive by defining content-based filters. This property corresponds to the common goal of message routing systems of delivering messages only to their intended destinations while avoiding the overheads of unnecessary forwarding. This is especially important for DTNs, in which the bandwidth to transfer messages during encounters may be scarce, and messages may occupy limited buffer space for extended periods of time.
- **Duplicate suppression and at-most-once delivery.** Due to the possible scarcity of bandwidth and the limited duration of encounters between hosts, DTNs try to avoid the overhead of transmitting multiple copies of a message to the same host. Duplicate suppression is typically achieved by exchanging IDs of previously seen messages or by appending to each message a list of IDs of hosts who have already forwarded the message. PFR systems guarantee that each data item is delivered only once to each host by maintaining and exchanging metadata called knowledge.

IV. BUILDING DTNS ON A PFR SYSTEM

Due to the key properties guaranteed by PFR systems outlined in the previous section, PFR systems provide the basic functionality to support a DTN as a strikingly simple messaging application. In this section, we describe the construction of such an application, observe the benefits and

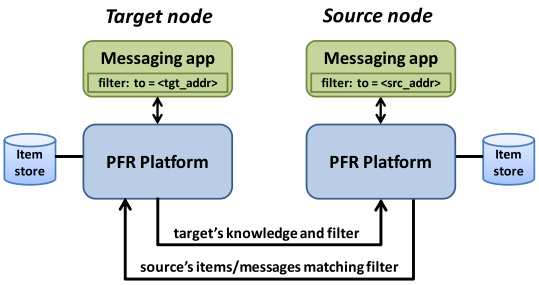

Figure 1. Architecture of PFR-based DTN

limitations of building on a replication substrate, and propose some extensions that are further explored in subsequent sections.

A. A simple messaging application

A DTN can be built on a PFR system by treating each message as an item that is replicated between the sender and the intended recipient(s). This can be achieved simply by including a "destination address" metadata attribute in each data item representing a message, and expressing each host's address through its filter. To send a message, a host creates an item representing the message and submits it to the replication layer. Each host's filter, which indicates the items that should be replicated locally, is set to select the messages addressed to it. Hosts synchronize when connections become available, and eventual consistency guarantees that each message is delivered to all recipients whose filters match the destination address attribute. At-most-once delivery ensures that hosts do not receive multiple copies of the same message. The basic operation of a prototype DTN built on Cimbiosys is depicted in Figure 1.

The simple DTN application demonstrates the usefulness of PFR systems as a substrate for building DTNs. Instead of reimplementing mechanisms such as controlled flooding to increase message delivery and message histories to avoid duplication transmissions, the application benefits from the properties of disconnected operation, eventual consistency, partial replication, and at-most-once delivery offered by the PFR system. Moreover, the efficient knowledge exchange used in modern replication protocols [4], [13] ensures that hosts can quickly determine which messages need to be delivered during brief encounters. After a message is received and processed, the destination node can simply delete the item, causing it to be discarded by forwarding nodes; no special acknowledgements are needed. The result is one of the simplest applications one could imagine building on such a replication platform.

However, the messaging application has a significant limitation which inhibits its performance in practice: without additional measures, hosts will not forward messages on behalf of arbitrary other nodes—an important feature of effective DTN approaches. In PFR systems, including the Cimbiosys platform on which we built our DTN messaging application, a replica receives and stores items that match its filter and nothing more, a property known as *eventual filter consistency* [13]. Thus, if each filter only selects messages for a particular destination address, and if each message is sent to a single address, then messages are delivered to their recipients only during direct encounters with the senders. Because connectivity opportunities typically are rare, and source and destination hosts may never connect directly, it is often necessary to enlist intermediate hosts as forwarders in order to ensure reliable and timely delivery.

Previous work on DTNs has shown, and our experiments presented in Section 6 confirm, that taking advantage of multi-hop routes is crucial for improving message delivery and delays. In the following subsections, we consider two options for enabling multi-hop forwarding in a PFR-based DTN: one based on expanding each host's filter and one that permits more sophisticated forwarding policies.

B. Including addresses of other hosts in filter

A simple way to enable multi-hop forwarding in a PFR-based messaging system is for a host's filter to include destination addresses other than its own. This allows a host to receive messages addressed to hosts other than itself during the synchronization process. For example, a user who owns multiple devices could configure the filter on each device to request messages sent by or addressed to any of his devices. One device could then forward messages en route between other devices. Similarly, message delivery rates could be improved by including in each host's filter the addresses of frequently encountered partners. For example, a person's filter could include a list of buddies from a social networking site that offers a delay-tolerant messaging service. In the extreme, if all filters select all messages, then the system provides epidemic flooding.

This approach potentially offers very good delivery rates in scenarios where hosts synchronize regularly with a relatively small set of partners and routes of only a few hops are expected to exist between sources and destinations, as envisioned in Fall's original DTN architecture [7]. It also applies well in scenarios where hosts would like to implement identity-based forwarding policies based on a message's source or destination. Users have complete control over the set of messages for which their devices will act as forwarding nodes. Filters can be set and adapted at any time, with the only restriction being that a host's filter needs to contain at least its own address. Moreover, this scheme requires no changes whatsoever to the underlying replication platform.

However, filters in a PFR system have limited expressiveness. The approach cannot take full advantage of unexpected communication opportunities between pairs of hosts, such as those arising from device mobility that is not completely predictable. A number of DTN application scenarios rely

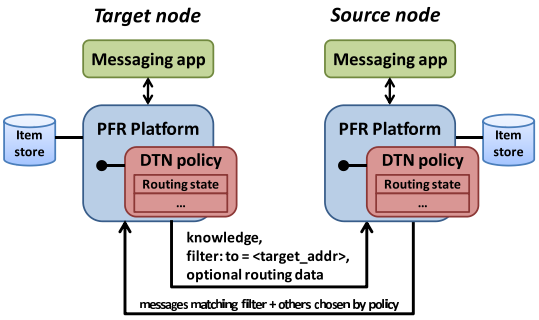

Figure 2. PFR-based DTN with routing policy extension

on opportunistic communication during encounters between arbitrary hosts and utilize more than the identities of the source and destination when choosing which messages to forward.

C. Transferring additional items during sync

State-of-the-art DTN protocols maintain data to aid routing such as information about previous host encounters in order to predict if a neighbor is a good choice to forward a message to its destination [11], [5]. To ensure that all possible routes can be exploited, we suggest an approach that allows such forwarding decisions. As described previously, filters select only those messages destined for the host, but additionally the PFR system can transfer extra items during synchronization that do not match the target's filter. The choice of which extra messages to forward can be based on routing state maintained by the source and additional data sent by target in a synchronization request.

The benefit of this approach is that it avoids fundamentally changing the meaning of filters. The source responds with a batch of messages from its store including those that match the target's filter (i.e., those addressed to the target) as well as messages chosen by a forwarding policy invoked by the source. Because filters do not change, the guarantee of eventual filter consistency ensures that all messages will eventually be delivered to their destinations given a connected synchronization topology.

Having a PFR system transfer items that do not match the target's filter is not a totally foreign concept. Systems like Cimbiosys already do this in limited situations in order to ensure the propagation of *out-of-filter* updates; Cimbiosys includes a *push-out store* in which each replica locally stores items that do not match its filter but need to be forwarded to the replica's parent [13]. Extending this existing mechanism to allow additional items to be sent and accepted during synchronization is not unreasonable. In the next section, we describe in detail an extension to a PFR system designed to allow existing DTN routing protocols to be implemented by expressing their forwarding logic as policies for transferring extra items during synchronization.

V. Adding support for DTN routing policies

In this section, we outline the implementation of an extension to Cimbiosys that augments the functionality of the replication platform with various DTN routing policies. The extensible platform retains its nice properties, such as reliable at-most-once delivery, while supporting pluggable modules for different DTN forwarding algorithms. Figure 2 depicts at a high level a synchronization between two hosts augmented with DTN routing policies. This section presents the interface for expressing forwarding policies and then describes the implementation of four popular DTN routing protocols using this policy interface.

A. Requirements

The motivation for extending the replication platform is to support transferring arbitrary items not matching the target's filter during synchronization—the choice of which items to send is determined by a policy input. DTN routing protocols use two broad types of information to decide whether to forward messages: 1) information associated with messages such as source and destination addresses and TTL values, and 2) additional routing state maintained by hosts such as historical encounter data.

Recall that, in our PFR-based DTN, messages are represented as replicated data items, which include both the message contents and metadata attributes such as a *source address* and *destination addresses*. At a minimum, a destination address is included with each message; other optional fields include message type or a timestamp. In addition to metadata fields whose values do not change for the duration of a message's lifetime in the network, some routing protocols maintain additional transient metadata associated with a specific copy of a message stored on a host, for example, a *time-to-live* (TTL) field. These host-specific metadata fields must be treated differently by the PFR system: updates to these fields should not be replicated through the system.

Our goal is to provide an interface that allows the logic of DTN routing protocols to be expressed concisely as forwarding policies. The requirements, in addition to extensible per-item metadata which is already supported by some PFR systems, are that: 1) DTN routing policies can define persistent data structures which are serialized to disk and retrieved whenever a synchronization operation is invoked, 2) synchronization requests can be augmented with additional data fields for routing data, defined and interpreted by DTN policies, and 3) items not matching the target's filter can be added to the batch of items prepared by the source. The interface used to express these policies is described in detail in the next section.

B. DTN policy interface

In our extension to Cimbiosys, each forwarding policy is expressed as a class in C#, implementing the interface `IDTNPolicy`, shown in pseudocode in Figure 3. Only three

Target node:

```
routingState = DTN.generateReq()
send knowledge, filter, and routingState to source

for each item received
     add item to local store
     update knowledge
```

Source node:

```
receive knowledge, filter, and routingState
DTN.processReq(routingState)
for each item in local store
     if item unknown to target
         if item matches filter or DTN.toSend(item)
             add item to batch
sort batch by priority
send batch to target
```

Figure 4. Sync protocol with DTN policy extension

```
public interface IDTNPolicy {
     DTNRequest generateReq();
     void processReq(DTNRequest);
     Priority toSend(Item, Filter);
}
```

Figure 3. DTN policy interface

methods must be provided by the implementer of a new forwarding policy.

The `generateReq()` method is called once each time that a host initiates synchronization with another machine with which it has come into contact. It returns policy-specific information that is then added to the synchronization request. Such information might include statistics about past encounters involving the requesting host, for example. This information is used by the receiving host to help determine which additional messages to return during the synchronization process.

The `processReq()` method is called upon receiving a synchronization request to process the policy-specific information contained in the request. Typically it stores this information in the receiving host's persistent routing state, perhaps using the new information to update its previously stored state.

The `toSend()` method expresses the forwarding logic of the routing protocol—using metadata attached to the item as well as routing state maintained by the policy. The method returns a code indicating whether or not the item should be included in the batch of items to be exchanged as well as a priority, to provide support for DTN routing protocols which forward messages in a specific order. To accommodate the DTN protocols which we examined, we defined a priority to consist of a "class" value, ranging from "lowest" to "highest," and a real-valued "cost" to break ties inside a class.

These methods can call on the existing Cimbiosys interfaces to access persistent data maintained in the replica's local item store and metadata store. They are invoked during the synchronization protocol as illustrated in the pseudocode presented in Figure 4. The next section provides additional details on the implementation of specific DTN policies.

C. Routing protocol case studies

To demonstrate the policy interface, we constructed policy classes for four DTN routing protocols that are representative of the entire set of approaches proposed in the literature. DTN routing protocols differ in how they replicate messages throughout the network and their usage of historical topology data to try to predict future network behavior. As case studies, we chose four well-cited DTN routing protocols with varying forwarding schemes and usage of past network state: Epidemic routing [16], Spray and Wait [15], PROPHET [11], and MaxProp [5]. Table I summarizes the implementation of policies for the four DTN routing protocols. The table describes the additional routing state that must be maintained by each host, the routing data that the target host adds to synchronization requests, and the policy applied by the source host to decide which non-matching items to transfer to the target.

In the remaining part of this section we give short descriptions of each of the routing protocols and explain how the corresponding policy is implemented.

1) Epidemic routing: Epidemic routing, one of the earliest proposed approaches to DTN routing, uses a simple flooding scheme but restricts the number of hops that a message might traverse. Messages are replicated during each encounter until a specified hop count, also called a time-to-live (TTL), is exceeded. Hosts store messages in a hash table, keyed by a unique ID associated with each message. Each host stores a bit vector, called a *summary vector*, indicating which entries in their local hash tables are set. Before hosts exchange messages, they exchange summary vectors to avoid transmitting messages that the other host has already received.

A routing protocol that floods messages throughout the network is trivial to implement. The `toSend()` method simply selects any item for which it is called; `generateReq()` and `processReq()` methods are not needed at all. To avoid transmitting the same message to a host multiple times, Epidemic routing calls for nodes to exchange summary vectors before exchanging messages [16]. We do not need to implement this mechanism, as the underlying replication platform prevents duplicate transmissions. In other words, Cimbiosys will only deliver each message once to each host, regardless of the path by which that

Protocol	Routing state	Added to sync request	Source forwarding policy
Epidemic	TTL per message		When TTL > 0
Spray&Wait	# copies per message		When # copies $>= 2$
PROPHET	Vector of delivery predictabilities: P[d] for each dest d	Target's P vector	Messages addressed to dest when target's P[dest] $>$ source's
MaxProp	Estimated meeting probabilities for all pairs	Target's meeting probabilities	All messages, ordered by priority (modified Dijkstra calculation)

Table I
SUMMARY OF POLICIES FOR DTN ROUTING PROTOCOLS

message is received.

To avoid excessive bandwidth usage, some Epidemic routing protocols limit the degree of flooding using TTLs or hop counts. Our Epidemic routing policy adds a TTL metadata field to each item representing a message. During a synchronization, the `toSend()` method selects every item with a nonzero TTL. When it encounters a message that does not have a TTL field, which will occur for new messages that have been directly added to the local replica by the messaging application, the `toSend()` method updates the stored message to add a TTL field with the default initial hop count limit. This method also decrements by one the TTL value of each selected item that is being forwarded to the target. This TTL update only affects the in-memory copy of items being sent, and hence does not affect the TTL values for messages stored in the source's item store.

2) Spray and Wait: Spray and Wait, an alternative to flooding-based approaches, injects a fixed number of copies of each message into the network. Spray and Wait challenges the idea embraced by many advanced DTN protocols that the best performance can be achieved by using history to predict future network characteristics. Instead, when a node injects a message into the network, it allocates a fixed number of copies, and, when encountering another node, transfers half of the copies it holds of any message. Thus, a "spraying" distributes copies of the message in a binary tree pattern rooted at the message source.

While the original Spray and Wait paper [15] does not discuss a mechanism for avoiding duplicate transmissions, a more recent implementation of the protocol [10] uses a similar approach to Epidemic routing: before exchanging messages, each host sends a request with a list of message IDs it would like to forward, and the other host responds with a list of message IDs that it has not already seen.

Implementing Spray and Wait as a forwarding policy is also straightforward. Each item representing a message contains a host-specific metadata attribute storing its "number of copies." This per-message attribute is added by the `toSend()` method when missing and initialized to a number specified by the policy. The `toSend()` method selects each item for which the local replica stores two or more "copies". It also updates the value of the "number of copies" field for both the locally stored item and the item in the synchronization batch with the original value divided by two. This results in the source sending half of its "copies" of each message to the target. Modifying the locally stored item is done through an internal Cimbiosys interface that avoids generating a new version number for the item; this allows a local adjustment to the "copies" field without causing the item to appear as an updated version that needs to be resent during subsequent synchronizations.

As in the case of Epidemic routing, the `generateReq()` and `processReq()` methods do nothing, and the guarantee of at-most-once delivery provided by the underlying replication platform permits us to avoid implementing Spray and Wait's duplicate suppression mechanism.

3) PROPHET: PROPHET employs flooding limited by an estimate of the likelihood that a node will be able to deliver a message to its destination in the future, based on past encounters. A probabilistic metric called the *delivery predictability*, $P(a, b) \in [0, 1]$, is maintained for each source a and destination b, indicating the likelihood that a will be able to deliver a message to b. Messages are replicated during each contact, utilizing the delivery predictability metric to limit the degree of replication: a copy of each message is forwarded only to nodes with a greater delivery predictability. Like Epidemic routing, PROPHET exchanges summary vectors to avoid duplicate transmissions.

The main idea behind delivery predictability is that two nodes that met recently will be likely to meet again in the future. Delivery predictability is maintained as follows: when nodes a and b meet, $P(a, b)$ is increased; $P(a, b)$ is aged down while a and b remain disconnected. And delivery predictability is transitive, meaning $P(a, c)$ is increased whenever $P(a, b)$ or $P(b, c)$ increases.

In order to incorporate the PROPHET forwarding logic in our DTN application we implement the policy interface as follows. The forwarding policy for PROPHET provides a `generateReq()` method that returns the target's delivery predictability vector, causing this information to be included in each synchronization request. The source's `processReq()` method saves this information in its local metadata store, where it can be accessed by the `toSend()` method.

The `toSend()` method simply compares its predictabil-

ity value for the message destination with the target's to decide whether to forward this message. If the target's delivery predictability for the message destination is greater than that of the source, this method returns a non-zero priority, causing the message to be added to the synchronization batch.

Before completing the synchronization process, the source updates its delivery predictability vector using the target's values that it received in the sync request according to the procedure outlined in the PROPHET protocol [11]. Because hosts synchronize twice during each encounter, with each host taking the role of source for one synchronization session, each host's delivery predictability vector is updated only once for each pair of synchronizations.

4) MaxProp: MaxProp is another DTN routing algorithm that maintains historical information to estimate the likelihood that nodes will meet again in the future. This information is used to determine the order in which messages are transmitted during an encounter. Each node maintains a probability distribution indicating the identity of the next node it will encounter; when another node is encountered the associated probability is increased and the distribution is normalized. Nodes exchange their probability distributions during encounters. For each message it might forward, a node considers all possible paths in the network along which the message could be delivered and assigns a score to the message based on the lowest cost path, defined as the sum of the probabilities that each connection along the path does not occur.

When two nodes meet, messages are transmitted in the following order: messages addressed directly to the neighbor; any "new" messages with a hop count below a threshold, sorted by hop count; and finally remaining messages ordered according to the aforementioned scores. Acknowledgements are flooded through the network to clear buffers of messages that have been delivered to their destinations. MaxProp's approach to avoiding duplicate transmissions is different from the previous three protocols. A host appends its own ID, as well as the IDs of all nodes to which the message is being forwarded, to a hoplist stored in each message that it forwards. This suppresses, but does not completely eliminate, duplicate transmissions.

The MaxProp policy is implemented as follows. The policy module maintains a probability distribution over all hosts that each other host will be the next to be encountered and updates this information at each encounter according to the protocol specification [5]. Similar to the PROPHET policy, this information is exchanged and processed by the `generateReq()` and `processReq()` methods. We take advantage of Cimbiosys's support for partial synchronizations between hosts and ordering inside synchronization batches. The `toSend()` method selects every item, as in the Epidemic flooding scheme, but assigns priorities that are computed by considering the number of hops the message has traversed and the likelihood of delivering the message along all possible paths. Messages that match the target's filter are transmitted first; remaining non-matching messages are transmitted in order of decreasing priority.

VI. Evaluation

The main objective of our evaluation was to explore the various tradeoffs between message delivery, delay, and storage consumption for different approaches to building a DTN messaging system using Cimbiosys. After describing our experimental setup, we present results for experiments designed to evaluate the performance of a DTN application built on unmodified Cimbiosys, the same application running on Cimbiosys extended with DTN routing policies, and finally the DTN application and routing policies under bandwidth and storage constraints.

A. Experimental setup

To evaluate our system, we constructed an environment that runs many instances of our DTN application on the same physical machine. Each DTN application instance represents a different device and is paired with a Cimbiosys replica. Whenever a host sends a message, the DTN application simply inserts the message into the sending host's replica. During an encounter between two hosts, we performed two syncs between the corresponding replicas, alternating the source and target roles between syncs.

Our experiments were driven by real traces of vehicular mobility and e-mail workloads. A schedule of encounters was taken from traces of meetings between buses in the DieselNet mobile testbed [6] in Amherst, MA. The message workload was provided by a version of the Enron Email dataset published by the UC Berkeley Enron Email Analysis Project [1]. The Enron Email dataset is unique as a large, openly-available e-mail dataset. We used this dataset to determine which node sends messages to which other nodes.

The DieselNet trace is split into separate days, with a set of buses scheduled for each day. Bus schedules in the trace are not constant; a bus might have a different schedule on different days or might not be scheduled at all. For each day in our experimental run, the experiment uniformly distributes e-mail users to the buses scheduled on that day. This models a scenario where messages sent between users are routed through a network of vehicular nodes. An average of 23 buses were active each day. Sync operations were triggered whenever buses encountered each other in the trace. We selected those days from the trace that had encounters from early in morning (8:00am) until late in the evening (11:00pm). The total number number of days used in our experiments was 17, and the total number of encounters between buses was approximately 16,000.

Messages were injected during a two-hour period in the morning (8:00am–10:00am) of each day, at two-minute intervals. Message injection is stopped after the eighth day

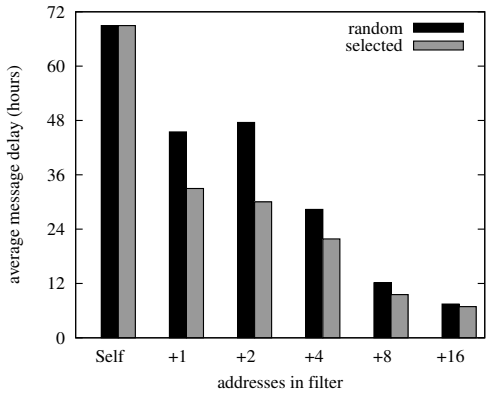

Figure 5. Average message delay in a simple DTN application that uses multi-address filters to route messages.

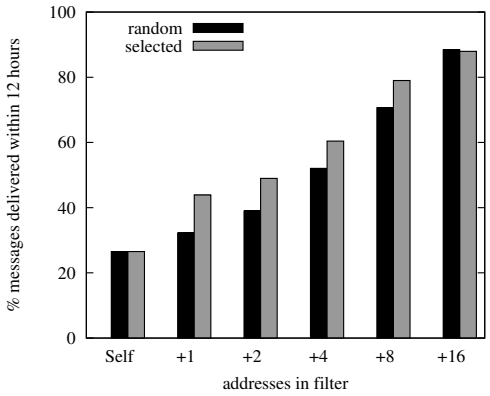

Figure 6. Message delivery of a simple DTN application that uses multi-address filters to route messages.

to allow for eventual convergence. A total of 490 messages were injected during each experiment. Messages received by the destination are reported as successfully delivered and messages are never deleted. Unless explicitly noted the experiments place no bounds on network bandwidth or per-host storage.

B. Benefits of multi-address filters

We first examine the performance of an unmodified PFR system as a DTN. Section IV proposed a simple way to enable multi-hop forwarding in a DTN built on Cimbiosys by having hosts include addresses other than their own in their filters. We consider two strategies for populating filters: 1) *random*, which populates a host's filter with addresses of k randomly chosen other hosts, and 2) *selected*, which picks the k other hosts that a given host will encounter most in the trace.

Figure 5 shows the mean delivery delay of all messages for the aforementioned filter strategies. The number of additional hosts included in a node's filter is varied from $k = 0$ to $k = 16$. The value $k = 0$ represents basic Cimbiosys that exchanges messages only when the sender directly encounters the destination. Naturally, it has a high delay compared to other accelerated, multi-hop variants. The average delay in our experiment for basic Cimbiosys was about 70 hours, counting the delivery time of all messages.

For values $k > 0$, we observe that message delivery accelerates with the number of addresses in a host's filter. Even with a single host added to the filter, average delay drops to under 35 hours (about 50%) for the *selected* strategy. A larger number of addresses allows the host to forward messages on behalf of a greater number of destinations, resulting in more opportunities to deliver messages, and therefore lesser delay. Moreover, choosing for a host's relay set other hosts it is most likely to encounter reduces message delay compared to choosing randomly. The benefits of the *selected* over the *random* strategy diminish, however, as the number of entries in the filter approaches the total number of hosts in the network. In the limit, both strategies are identical and equivalent to simple flooding.

Figure 6 shows the percentage of messages delivered within a maximum delay of 12 hours. This figure shows what message delivery rate would look like for messages with bounded lifetimes. In this case, we pick 12 hours because around 12 hours after message injection each day, buses return to their shed. Figure 6 shows that basic Cimbiosys delivers about 30% of messages within a day. The reason some messages remain undelivered within a day is because not all buses encounter all other buses on the same day. Other filter-based strategies ($k > 0$) show similar improvements to the 12-hour message delivery as they did for the average delay. That is, delivery improves as more hosts are added to the filter.

To accelerate message delivery, one would want to add as many addresses as possible to a host's filter. However, there is an important tradeoff between message delivery and storage consumption. Adding more addresses to filters increases the degree of replication of messages, with the the total number of copies of each message in the system converging towards $2 + k$. In practice, a DTN application running on Cimbiosys would need to manage this tradeoff based on the storage capacity of the hosts and the delivery delay the application is willing to tolerate.

C. Effects of DTN routing policies

Next, we examine the performance of the DTN application on top of Cimbiosys extended with the four routing policies described in Section V. The DTN policies have the same configurable parameters as the original protocols. For our experiments, values were chosen based on the properties of our scenario and recommendations provided in the papers whenever possible; the values are listed in Table II. Descriptions of these parameters can be found in the corresponding papers.

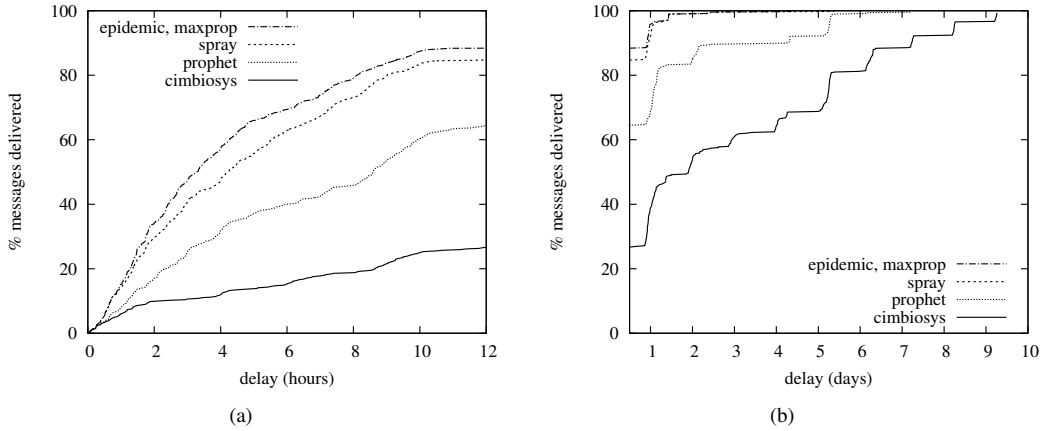

Figure 7. Cumulative distribution of message delays for different DTN policies a) for the first 12 hours, b) for the time greater than 12 hours

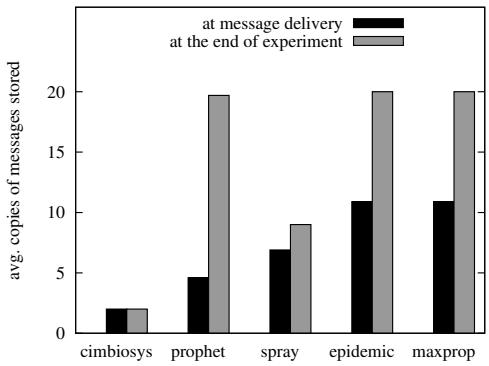

Figure 8. Message copies stored in the network at time of delivery and at the end of the experiment for different DTN policies

Epidemic	$TTL = 10$
Spray&Wait	$copies\ per\ message = 8$
PROPHET	$P_{init} = 0.75$, $\beta = 0.25$, $\gamma = 0.98$
MaxProp	$hopcount\ priority\ threshold = 3$

Table II
DTN PROTOCOL PARAMETERS

We plot the percentage of messages that are delivered within various delays (that is, the cumulative distribution function of message delays) for the different policies, split into two figures. Figure 7(a) plots the percentage of messages delivered for delays of 0 to 12 hours (on the x-axis). Figure 7(b) continues this graph for delays longer than 12 hours. As a reference, a line is included that reports the results for unmodified Cimbiosys, with no additional addresses in the hosts' filters—this is the same configuration shown in Figure 6, with $k = 0$.

An important observation from Figure 7(b) is that allowing Cimbiosys to continue delivering messages beyond a day eventually ensures 100% message delivery. However, it takes more than 9 days to deliver all the messages. Extending Cimbiosys with DTN routing policies substantially reduces this worst case delay: PROPHET reduces it to 7 days, and Spray and Wait, MaxProp and Epidemic flooding reduce it to approximately 4 days. Note that Epidemic and MaxProp have identical delay distributions for this experiment because they differ in the messages forwarded only when the network bandwidth is constrained.

The benefit derived from the extensions varies for different routing policies. Epidemic and MaxProp, which rapidly flood the entire network with messages, provide the most benefits. PROPHET, which forwards messages through selected paths, is less effective[1], while SprayAndWait, which limits the maximum copies of messages generated, is in between.

Minimizing message delay in DTNs is trivial if there are no limits on the number of messages that can be exchanged during each encounter and stored by hosts. In this case, replicating all messages at each encounter (like Epidemic) delivers messages at the earliest possible time. However, a fair assessment of the performance of a DTN policy also needs to take into consideration the storage consumption and network traffic for storing and delivering messages. In Figure 8, we examine the storage consumption of each DTN policy. We report the number of copies of a given message stored in the network at the time the message was delivered and at the end of the experiment. The number of copies stored at the time of delivery models an ideal situation in which messages are deleted from the entire system immediately after they have been received. The number of copies stored at the end of the experiment shows the worst case, in which messages are never deleted, and continue to be forwarded even after they reach their destinations.

[1]It appears that encounters in the DieselNet trace do not follow predictable patterns that PROPHET can exploit.

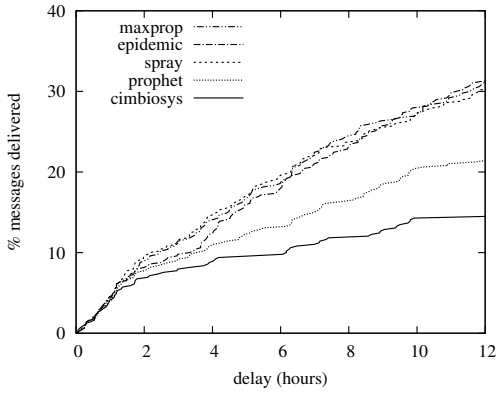

Figure 9. Cumulative distribution of message delay for DTN policies under bandwidth constraint (only one message exchanged per encounter)

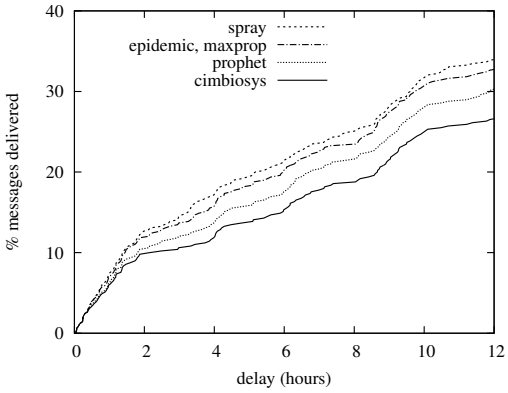

Figure 10. Cumulative distribution of message delay for different DTN policies under storage constraints (only a maximum of 2 messages must be stored on each node)

As can be observed from Figure 8, unmodified Cimbiosys generates the fewest number of copies of messages. Cimbiosys stores only two copies in the network per delivered message: one at the sender and the one at the receiver. PROPHET and SprayAndWait store a few more copies in the network while delivering a message, but given their much lower message delivery delay (Figure 7(b)), the additional copies seem to be well invested. The rest of the DTN policies have similar storage consumption when measured at the time of message delivery. When looking at the number of copies stored in the network at the end of the experiment, it stands out that SprayAndWait performs particularly well. This is reasonable since SprayAndWait limits the maximum number of times messages can be forwarded.

D. Performance under limited resources

So far all our experiments have assumed unlimited storage capabilities and unlimited network bandwidth. In reality, the wireless bandwidth, the actual encounter duration and the storage capacity of devices are limited. We repeated the previous experiments after placing constraints on storage and bandwidth. To emulate the effects of limited bandwidth, we allow only one message to be exchanged during an encounter. For limited storage, we set a limit of 2 messages stored per node at any point in time, excluding messages for which the node itself is the sender or the destination. These rather stringent constraints allow us to examine the performance of the system under worst-case conditions.

Figure 9 shows the cumulative distribution of message delay for the different DTN routing strategies up to 12 hours. Message delivery, however, can continue for days as depicted in Figure 7(b). As expected, message delays increase due to the network being a bottleneck; this is shown in Figure 9. For basic Cimbiosys, the percentage of messages with more than a day's delay increases to about 85% from 70%. However, extensions based on DTN routing policies continue to reduce delay and deliver substantially more packets within 12 hours. As in the unconstrained-network experiment, Epidemic, MaxProp, and SprayAndWait are far more effective than PROPHET.

Figure 10 illustrates the performance of each of the DTN routing policies with storage constraints. A FIFO strategy was used to expunge messages when the storage consumption on a node exceeded the storage limit. Cimbiosys is not affected by the storage limitation as it does not exploit relay opportunities. By limiting a node's storage capacity, we restrain the node's capability to forward messages on behalf of others. Consequently, although the DTN policies are able to reduce message delay and accelerate message delivery compared to basic Cimbiosys, they are less effective than they were with unbounded storage.

VII. Conclusions

A delay tolerant messaging application can readily be constructed on top of a peer-to-peer filtered replication platform with messages as replicated items and filters that select messages for particular destinations. Doing so imbues the application with the desirable properties of the underlying replication platform, including guaranteed delivery, topology independence, duplicate suppression, and concise metadata. Guaranteed delivery derives from the replication system's property of eventual consistency, assuming that there is sufficient connectivity in the network to eventually deliver messages in a hop-by-hop manner through pairwise synchronizations. Topology independence allows the system to forward messages during opportunistic encounters between arbitrary hosts. Duplicate suppression equates to at-most-once message delivery and is achieved through the exchange of knowledge in the replication protocol. Representing such knowledge in a data structure whose size is proportional to the number of replicas, rather than number of messages, enables efficient synchronizations. However, as observed in our evaluation of a DTN-like application that was built

on the Cimbiosys platform, if a host's filter selects only messages that are addressed to that host, then only single-hop paths are used to deliver messages. This can result in exceedingly long delivery delays and even low delivery rates when messages have limited lifetimes.

DTNs benefit from multi-hop routing through intermediary forwarding nodes that replicate messages within the network. DTN forwarding algorithms range from simple flooding to more sophisticated techniques based on known network characteristics and histories of past encounters. This paper showed how such algorithms can be incorporated into the Cimbiosys replication platform using pluggable modules with a simple interface. Evaluations of Cimbiosys extended with four representative DTN forwarding schemes confirm that significant reductions in delivery delay can be attained while preserving the replication guarantees and the simplicity of the driving messaging application. Of course, reduced delivery delays come at the cost of increased bandwidth and storage usage by forwarding nodes. Our experiments quantify this tradeoff using mobility traces from city buses and communication patterns from an e-mail dataset. Even with very restrictive storage and bandwidth limits, the evaluated forwarding policies demonstrate significant improvements to the basic replication substrate. We conclude that innovative ideas from both the DTN and replication communities can be effectively integrated to help build better systems.

REFERENCES

[1] UC Berkeley Enron Email Analysis. http://bailando.sims.berkeley.edu/enron_email.html.

[2] Wizzy Project. http://www.wizzy.org.za/.

[3] A. Balasubramanian, B. N. Levine, and A. Venkataramani. Enhancing interactive web applications in hybrid networks. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 70–80, 2008.

[4] N. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 5–5, 2006.

[5] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *In Proc. IEEE INFOCOM*, 2006.

[6] J. Burgess, B. N. Levine, R. Mahajan, J. Zahorjan, A. Balasubramanian, A. Venkataramani, Y. Zhou, B. Croft, N. Banerjee, M. Corner, and D. Towsley. CRAWDAD data set umass/diesel (v. 2008-09-14). Downloaded from http://crawdad.cs.dartmouth.edu/umass/diesel, Sept. 2008.

[7] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[8] P. Gilbert, V. Ramasubramanian, P. Stuedi, and D. B. Terry. The duality between message routing and epidemic data replication. In *Proceedings of the Eighth ACM Workshop on Hot Topics in Networks (HotNets 2009)*, Oct 2009.

[9] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, 2004.

[10] E. Kuiper. Details on LAROD and Spray and Wait Implementations. Technical report, Linköping Sweden, December 2009.

[11] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *SIGMOBILE Mobile Computing and Communication Review*, volume 7, 2004.

[12] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 256–269, 2004.

[13] V. Ramasubramanian, T. L. Rodeheffer, D. B. Terry, M. Walraed-Sullivan, T. Wobber, C. C. Marshall, and A. Vahdat. Cimbiosys: a platform for content-based partial replication. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 261–276, 2009.

[14] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[15] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259, 2005.

[16] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University, July 2000.