Robust Content Dissemination in Disrupted Environments

Ignacio Solis
Palo Alto Research Center (PARC)
Palo Alto, California, USA
isolis@parc.com

J.J. Garcia-Luna-Aceves*
Computer Engineering Department
University of California, Santa Cruz
jj@soe.ucsc.edu

ABSTRACT

Content dissemination in disrupted networks poses a big challenge, given that the current routing architectures of ad hoc networks require establishing routes from sources to destinations before content can disseminated between them. In ad hoc networks subject to disruption, lack of reliable connectivity between producers and consumers of information makes most routing protocols perform very poorly or not work at all. We present DIRECT (DIsruption REsilient Content Transport), which is a content dissemination approach for ad hoc networks that exploits in-network storage and the hop-by-hop dissemination of named information objects. Simulation experiments illustrate that DIRECT provides a high degree of reliability while maintaining low levels of delivery latencies and signaling and data overhead compared to traditional on-demand routing and epidemic routing.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

Keywords

Disruption Tolerant, DTN, Content Centric, Content Based, Networks

General Terms

Performance

[1]This work was partially sponsored by DARPA under grant W15P7T06CP815 and through Air Force Research Laboratory (AFRL) Contract FA8750-07-C-0169. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

*J.J. is also afiliated with Palo Alto Research Center (PARC)

CHANTS'08, September 15, 2008, San Francisco, California, USA.

1. INTRODUCTION

Two key assumptions in the design of the routing protocols for the IP Internet and wireless multi-hop ad hoc networks (MANETs) have been that: (a) physical connectivity exists on an end-to-end basis between sources and destinations for extended periods, and (b) routes are established to reach the locations of the intended destinations. These assumptions have had profound implications on how communication bandwidth is shared, how routing is accomplished, and how data are disseminated. In particular, routing in packet-switching networks has been based on routing tables derived entirely from topology (or connectivity) information that represents only a snapshot of the state and characteristics of network links at particular instants. Furthermore, while the assumption of routing information to specific addresses renders adequate delivery rates in connected networks with stable topologies, it can lead to very poor resource utilization in disrupted networks when physical paths to the original locations of content are lost for long periods of time, or never exist.

The cost, energy consumption, and form factors of computing devices have enabled embedded computing and networking devices that can be used in environments in which end-to-end connectivity may be intermittent at best. These new application environments range from interplanetary research to wearable computers. Networks in which end-to-end connectivity is not guaranteed have been called delay tolerant, disruption tolerant, intermittently connected, episodically connected, or highly partitioned. In this paper, we use the term *disruption-tolerant networks*, or DTNs. Clearly, routing in a DTN cannot be accomplished in the same way as routing in a network in which end-to-end connectivity is assumed to exist except for extraordinary circumstances. As the seminal work by Fall [10] and others has shown, routing in a DTN must be accomplished differently than in traditional networks. Routing in the Internet and well-connected ad hoc networks has no temporal dimension, because it is based on a distributed or local search of paths obtained from snapshots of the network topology. In contrast, routing in a DTN must be a function of space and time, because physical links exist only temporarily, and paths from sources to destinations can be considered to exist only as functions of connectivity (links) occurring over time.

Section 2 summarizes the prior work on routing for DTNs, which started with the work of the Interplanetary Internet Research Group (IPNRG) [4] within the IRTF (Internet Research Task Force). This survey indicates that prior solutions to the DTN routing problem either have focused on establishing spatio-temporal routes to specific destinations, rather than to content, or have modified epidemic dissemination of content. These solutions have relied on knowing the entire network topology, the ability to control some nodes, or the ability to duplicate data freely in the network.

Section 3 presents DIRECT (DIsruption REsilient Content Transport), which disseminates content by name on-demand to sites with interest in content from the nearest sites hosting the content. DIRECT is similar to Directed Diffusion [14], in that named content is disseminated on the basis of statements of interest. However, DIRECT accommodates disruptions in network connectivity of arbitrary duration. To accomplish this, DIRECT disseminates interest in content persistently across connected components, so that content can be replicated opportunistically as nodes move across such components, but only towards those nodes that have expressed interest.

Section 4 presents the results of simulation experiments used to compare DIRECT against epidemic content dissemination and on-demand routing of content from specific sites. The results indicate that DIRECT provides the best of both approaches. DIRECT attains delivery latencies and percentage of objects delivered that are as good or better than epidemic content dissemination, but incurs only a fraction of the overhead. DIRECT incurs even less overhead than routing of content from specific sites, and the latter delivers only a fraction of what DIRECT can deliver.

2. RELATED WORK

The IRTF's *DTNRG* [8] (Delay Tolerant Networking Research Group) introduced the *bundle* architecture [10] which groups messages into bundles that encompass entire sessions, performs *store-carry-and-forward* of bundles, and employs *custody* [11] transfer for reliability. The DTNRG also designed addressing and naming schemes [4] for DTNs. The routing schemes that have been proposed for DTNs thus far have focused on establishing spatio-temporal routes to nodes or groups of nodes, given that connectivity information (links or contacts) is enforced, scheduled, or random.

Routing schemes based on enforced contacts typically employ specialized nodes such as robots with controlled mobility. Such specialized nodes, called mules and message ferries in many schemes [23, 29, 16, 30, 18], are such that their mobility can be controlled to provide connectivity to other nodes in networks as needed. Much of the work has focused on route scheduling of the specialized nodes and synchronization between their routes. It has also been shown that such data mules or ferries can be used as an energy saving device for other nodes in the network; if there are no ferries nearby, nodes can be turned off to conserve energy.

Several approaches [20] and [15] take advantage of the periodicity inherent to some mobility patterns and assume global knowledge of node schedules. They use variants of *space-time* routing tables, and employ, among other methods, a modified Dijkstra's algorithm to determine shortest paths over time in these structures. Some other approaches address the case of predictable or scheduled mobility (e.g., buses and trains). Some approaches [27, 6] rely on past mobility and topology knowledge to predict future behavior. *MaxProp* [2], showed better performance in a deployed network of buses than an oracle with perfect schedule knowledge. The approach described in [27] tries to predict the future topology of a network by determining *how long* nodes that are connected will remain connected.

Location information has been shown to be very useful for routing in disrupted networks (e.g., MobySpace [17] and MV routing [3]). These methods require an external localization mechanism, such as GPS, and assume that a node who has visited a particular location is likely to revisit it, and therefore is a good candidate to carry messages to that location.

Some approaches have also proposed the deployment of static nodes to increase contact opportunities in DTNs. This is the case of *Throwboxes* [31] which are stationary nodes that typically have greater wireless communication, storage, and power capabilities. *Throwboxes* act as static relays that can receive and forward messages as nodes come in range. If node schedules are known or can be predicted, these static relays can be placed such as to optimize contact opportunities.

When connectivity information cannot be enforced or predicted, routing is done opportunistically. The simplest scheme is *epidemic routing* [28], with which a copy of a message is forwarded to all nodes encountered by the message. The limitation of epidemic routing is that its transmission requirements can be prohibitively expensive. As a result, several schemes [12, 19, 24] have been proposed to improve on epidemic routing by controlling the way in which message flooding occurs. In *spray and wait* [25], only the source can replicate a message; after 'spraying' several copies of a message, the host 'waits' until one is delivered. In *Spray and Focus* [26], after the source "sprays" a message similarly to spray and wait, each copy is forwarded according to a utility-based function aimed at finding better opportunities to relay messages. The CAR [21] algorithm uses adaptive weights on several node attributes as well as Kalman filters to maximize the probability of a node to deliver messages to a destination. Work on *pocket switched* networking [13, 5] models the distribution of contact and inter-contact times in order to better design forwarding strategies, and a number of mechanisms are proposed to extend opportunistic forwarding protocols.

Several schemes have been proposed to extend routing algorithms originally designed for mobile ad hoc networks (MANET), in particular on-demand routing, to operate in scenarios in which end-to-end connectivity does not exist. Disconnected transitive communication [6] is a proposal to enable communication across clusters in MANETs using utility values to decide which node in the cluster is best suited to transmit a message to the destination. The goal of the efficient route discovery mechanism proposed by Dubois-Ferriere et al. [9] is to decrease the amount of route discovery overhead by forwarding route requests in the direction of the destination. The proposal by Ott et al. [22] augments AODV with the ability to identify DTN-capable routers. The source then decides whether to use AODV– (if available) or DTN routes. An underlying DTN routing mechanism is assumed to be in place. The Space-Content-adaptive-Time Routing (SCaTR) protocol [1] also extends AODV to accommodate partitions; however, SCaTR does not assume the existence of an underlying DTN routing fabric nor does it leave the routing decisions to the sources.

In *Island Hopping* [7], nodes run a distributed algorithm to find a set of *concentration points* (CPs), which are the set of connected subgraphs, or "connectivity islands"; and a set of edges representing possible node movements between CPs. Nodes learn the entire graph by gossiping. Given that edges are based on prior node movement, a certain degree of message replication is employed to ensure that at least one is delivered to the final destination.

From the above summary, we observe that prior approaches for routing in disrupted environments do not route named content objects directly. Our approach, which we describe in the next section, is inspired in Directed Diffusion and consists of routing named objects based on interest statements that percolate across connected components of a disrupted network, so that some nodes can choose to copy and carry the objects of interest back to the nodes that originated the interest in the objects.

3. DIRECT

DIRECT is based on the dissemination of interest in and copies of information objects in networks subject to disruption. Content dissemination in DIRECT is based on the names of information

objects rather than the nodes storing them. This contrasts with the current delay-Tolerant infrastructure proposed by the DTNRG[8] and much of the prior work on disruption tolerant networks, where reliable information dissemination is still based on the identification of endpoints reminiscent of TCP connections.

3.1 A Content Centric Interface to Applications

DIRECT defines four high level primitives: Publish, Get, Send and Handle. These primitives give the network more flexibility in managing data and provides more functionality to applications.

- PUBLISH: When a node wants to make some information available to other nodes it *publishes* an object. Publishing an object involves giving the object a name and some attributes. The local network daemon then takes possession of the object. This is equivalent to the node becoming a server for that piece of static data.
- GET: If a node wants some specific piece of data it asks the network to *get* that object. The node needs to specify the name and attributes of the object it wishes to receive. This specification can include expressions to allow for complex queries. The current instantiation of the protocol allows for prefix and postfix matching in the name and publisher of an object. This can effectively be used for hierarchical naming.
- HANDLE: Not all objects need to be created before they are accessible, some objects will be created as replies to specific queries or gets. This is the case of dynamic data, for example. A program can request to handle a subset of the naming space. In this situation, the network will tell the program when it has received a get for a name and the program can generate the object on the fly and issue a publish.
- SEND: When a node knows that a piece of data is intended for a certain recipient it can start the data exchange by issuing a *send*. This is basically a compound function made of a get and publish. This is the function to use for "sender-driven" communication.

In order for nodes to communicate, one node issues a publish when it wants to make some data public, while the other issues a get. If the data object is dynamic, then a node might request a handle to a name.

3.2 Content

We call a named piece of content an object. The content inside the object is a blob of bits that is not interpreted by the network in any way. Objects are characterized by multiple attributes, including a name, a publisher, a timestamp, a type, etc. We call this set of attributes the *absolute name*. Our current design is simple and does not address arbitrary attributes, tags or associations. Such extensions is the subject of in future work.

The name and publisher fields are the basis for matches and queries. If we were to map (in a very simple way) URLs to them, the hostname would be the publisher and the path section would be the name. On a more complete system, the publisher would actually be a security association between the entity doing the publishing and the mapping between the *absolute name* and the data (using signatures and encryption if necessary).

Information about objects is kept in a table at every node. Table 1 shows the structure of the table entries. Information about an object is entered into the table when the object or an object announcement is received . The fields are obtained from the *absolute name* and expanded with local information about the state of the object and who sent us the information. Our initial implementation supports only PUBLISHED, LOCAL CACHE and NEIGHBOR CACHE as object states.

Field	Definition
Name	Object name
Publisher	Object publisher
Type	Type of object
Size	Object size
Time Stamp	Object creation time
State	Local object state
Last Hop	Who sent the object/announcement

Table 1: Object Table Entry

Naming in content-based routing is an ongoing research topic and . there is no consensus on the best way to structure object names. Some proposals argue for a flat naming space, some argue for a hierarchical naming space. Some want to embed information on the name, others want to associate arbitrary attributes to names. In this work, we take an agnostic stance and treat names as a simple string. This can work for organizing names by convention or standards. The same applies to Publishers.

3.3 Queries

When a program wants to retrieve an object, it issues a Get command. This translates directly to a network query. Matching a query to an object is basically a comparison between the query fields and the *absolute names*. A query contains very similar fields to an *absolute name* (name, publisher, timestamp, etc). The name and publisher can be encoded to do prefix, postfix or exact match. This simple addition adds a great deal of flexibility to queries and name structures.

Queries come in 3 basic types: Pub-match, All-match and Stop-match. In Pub-match the query will only be answered by the publisher. That is, even if an object with a matching *absolute name* exists in an intermediate node, the query travels to the publisher. All-match is basically a flooded query. Every object that matches will be sent back and the query will be always forwarded. Finally, a query of type Stop-match will return the first object that matches and stop. There is, however, no guarantee that only one answer is received, because a query can travel in multiple directions.

Field	Definition
Name	Object name being queried
Publisher	Object publisher being queried
Type	Object Type being queried
Time Stamp	Object creation time
QType	Type of query
QFlags	Query flags
QTime Stamp	Query time stamp
Source	Query originator
State	Local query state
Last Hop	Who forwarded the query to us

Table 2: Query Table Entry

Queries are kept in a query table. Table 2 describes the field format. Each entry contains information about the object for which a query is issued. Name and Publisher can contain wildcards for prefix and postfix matching. Type and Time Stamp can be specific values or *any* value. Query Type defines to which of the three types

of query the query applies (there are also subtypes). The Flags carry miscellaneous settings for handling this query. The Query Time Stamp defines when this query was originated and is used for comparisons. The Source is used to know who originated the query. State defines the local state of this query (active, inactive, etc). Finally, Last Hop is used to calculate who the previous hop in the tree is. If we match this query we will try to contact this node with the object.

3.4 Information Dissemination

The DIRECT baseline implementation supports a very simple approach to information dissemination. Nodes in a DIRECT network are configured with identifiers. These identifiers act as hostnames, aliases and group identifiers, and they are used for the publisher field of an object.

Each node maintains three main tables, the neighbor table, the object table and the query table. The neighbor table keeps track of all the neighbors that the node encounters and their status. The object table holds all the information the node has about objects and their (possible) location. The query table contains all the queries the node has received or created.

When a program wants to publish an object it gives the object to the network layer with its *absolute name*. We assume that, in the majority of cases, the publisher of the object is one of the node identifiers. The publisher is tied to a cryptographic signature, which basically assumes that the node has permission to publish under that name (by having the right key). It is possible for a node to "publish" an object created by another entity (for example, a presigned object), but in this case, the node is merely adding the object to the cache, not performing a publish operation.

Once an object is published, it is the responsibility of the network layer and it is added to the object table. A locally published object is never discarded, unless explicitly told to do so by an application.

To retrieve an object from the network, a program creates a query to issue a get. When the network receives the get from the application, it distributes the query. Our current implementation uses flooding to distribute a query. This allows a query to reach all the nodes within a given connected component of the network. Even though flooding incurs a large overhead, it allows us to find objects cached near by that meet certain criteria.

When a new query is received, it is added to query table along with the previous hop information. The query is then compared to the objects in the object table. If a match is made, the object is returned to the previous hop in the query tree. If the query is Pub-Match and the node handling the query is the publisher of the object or it is a Stop-Match query, the query is not forwarded after a match. The query is forwarded otherwise. If a match is made and the query is fulfilled, the query is marked as inactive. If after a certain delay the query has not been answered, the requesting node increase the sequence number assigned to the query and issues the query again. The default value for reissuing a query is set at 20 seconds.

Upon receiving an object, a node adds it to its object table. It then checks whether the object matches any of the active queries. If it does, the object is forwarded to the respective previous hops. If a previous hop is no longer available, the node makes a 1-hop object advertisement. This informs its current neighbors of the existence of a copy of the object. When receiving and object advertisement, a node checks for matches on its query table. If it finds a match it issue a request for that object.

Periodically, a node broadcast hello messages. These messages are used to maintain the neighbor table. When a node hears from a new node, it adds the node as an active neighbor. If a node does not hear from a neighbor for a period of time, the node is marked as remote. If such a node returns to the neighborhood, it is again marked as active. When a node is marked as active, it triggers a new-neighbor event. When a new-neighbor event occurs, a node exchanges active query information with the new neighbor. This effectively spreads queries that have not been answered yet.

Pseudo-code for some of the event handling functions is presented in figure 1. These functions get called when a transmission of a specific type is received.

```
function handle-query(query)
  if(we have this query)
    if(incoming query is not newer)
      return
    endif
  endif
  update local query table
  for every object that matches
    send object to previous hop
    mark query inactive
  endfor
  forward query

function handle-object(object)
  have-to-advertise = false
  for every query that matches object
    if query is local
      deliver local
    else
      if previous hop for query is an active neighbor
        send object to previous hop
      else
        have-to-advertise = true
      endif
    endif
    mark query inactive
  endfor
  if have-to-advertise
    send 1-hop object advertisement
  endif

function handle-advertisement(object-adv)
  if any query matches object-adv
    send object request

function handle-object-request(object, neighbor)
  if we have object
    send object to neighbor

function handle-new-neighbor(neighbor)
  for every active query
    send query to new neighbor
```

Figure 1: Simplified pseudo code

4. PERFORMANCE COMPARISON

4.1 Simulation Setup

We have implemented DIRECT in Qualnet 3.9, which we extended by adding the DIRECT routing framework as well as events such as object publish (with attributes) and object get (attribute based). Our current attributes include name, publisher and type, and is easily extensible.

Our base scenario is an area of 2Km x 2Km. This area is divided into an 8 by 8 grid as shown in Figure 2, and a node is placed in each grid square for a total of 64 nodes. Each node is then given random direction mobility inside the square. A node selects a random direction and a random (valid) point in that direction. It then travels towards that location at a random speed in a range of 1-5m/s. The node then pauses for a short period (random 0-10s) and chooses a

new direction. This scenario was chosen to focus our evaluation on one of the core functions of protocols for DTN environments, namely, how they perform when the network is not connected. As nodes move in a random direction model inside their grid squares, this creates periods of disconnection among any two nodes placed in neighboring squares. By varying the area covered by our grid the nodes become more or less likely to be disconnected. At the low end a size of 200m (that is 200m x 200m) keeps the network connected most of the time. At the opposite end, 400m, the network is almost always disconnected.

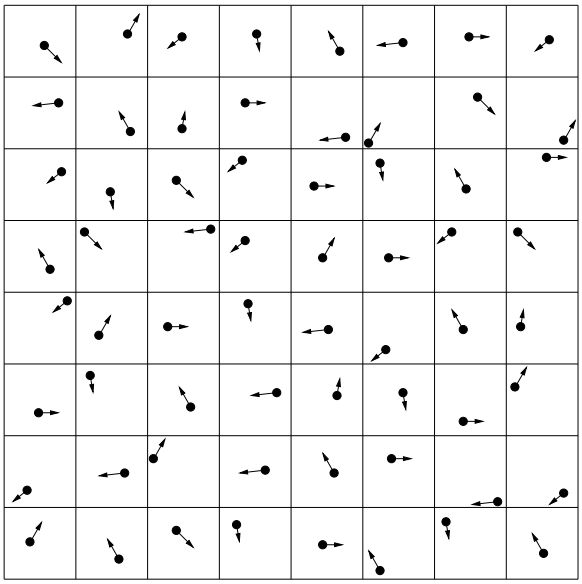


Figure 2: Grid mobility 8x8

100 objects are created and published randomly in the "server nodes". Each object is placed in only one node. We then generate 100 get requests. These requests are issued randomly by one of the "client nodes". Our configuration has all nodes acting as possible clients and hosts for objects. A node is never allowed to issue a get for an object it published.

The default simulation time is 21 minutes. There are no gets in the first or last 30 seconds. Gets are spread randomly in the middle 20 minutes. This allows the network to settle in the beginning and the transmissions to finish at the end. Objects are by default 2000 bytes long and each result is an average of 4 runs.

We evaluated DIRECT by comparing to epidemic routing and a TCP transaction Various parameters were examined, delay, delivery, and packets per object delivered. Our version of epidemic routing works similar to DIRECT except that when an object is received we check with all neighbors to see if they have this object or not. If not, the object is sent. Same check happens when a new neighbor is encountered.

Gets flood the network and establish reverse interests. When an interest and an object match the object is sent pack along the interest path. Gets are queries of type Stop-match, stopping at the first object match. The delay of object delivery is measured from the time a get is generated at the querying node to the time it is received at that node. This means that delay includes both the propagation of the query as well as the delivery of the object. In the case that an object is not received in a certain time interval a new request will be issued. The first request is still used to calculate the start time.

We compare object delivery in DIRECT with simple object delivery using TCP. For this we did not design a complete object retrieval algorithm. We mapped each of the gets into a TCP transmission. So if at time t node A is requesting object X and this object is located at node B, we create a TCP transfer from node B to node A at time t for the size of object X. Using this mechanism we can only measure the delay of the TCP transaction. Comparing this to the DIRECT measurement is a bit unfair since it does not include the "query propagation", only the "object delivery". Naturally this gives a tremendous advantage to TCP in terms of delay. Considering the small size of the default objects this could be as much as half the real delay TCP would incur in.

Delivery is measured as the percentage of gets that were actually completed before the simulation ended. Packets sent per object delivered is a very rough measure of overhead. It includes all the packets sent on the network. In the case of DIRECT and epidemic routing this includes the neighbor to neighbor packets as well as gets and objects. In the case of TCP it includes AODV and object TCP connections.

4.2 Results

Figure 3 shows the results for different grid square sizes. As the grid square size (and disconnection) grows the delay increases. This is the case for all protocols. TCP has a higher delay than DIRECT and epidemic routing for the connected values. At high disconnection the delay is lower. This is because delay is calculated only on the delivered objects and TCP is only able to deliver objects on "easy" connected paths. This can be clearly seen in the objects delivered graph, where TCP can only deliver a third of the objects delivered by DIRECT. At a grid size of 400 it is rare that nodes come close to each other and delivery is abysmal for all protocols; DIRECT still outdelivers TCP by 50%.

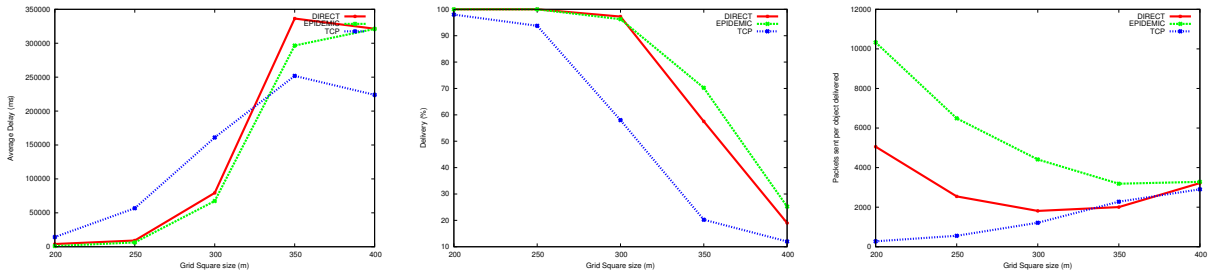
In terms of packets per object epidemic routing is encumbered by many unnecessary transmissions while DIRECT achieves a balance. TCP has a very low overhead on a connected environment since it can find a path and send only the required objects. Because the TCP scenario uses AODV (and doesn't simulate the query part) it has very little overhead. Both DIRECT and epidemic routing require the use of periodic neighbor messages which add up over time.

When the network is disconnected we see that the overhead of DIRECT and epidemic routing converge with TCP. Since TCP cannot deliver many objects, it becomes inefficient.

We wanted to take a closer look at how traffic affected performance. In this scenario we reduced the simulation time to 360 seconds. Once again, there were no gets in the first or last 30 seconds, making all gets happen in 300 seconds. We kept the grid square size stable at 250 meters and varied the number of gets generated from 10 to 1000. The number of objects to choose from was kept constant at a level of 1000. Like in other scenarios, some duplication will occur since the likelihood of an object being requested more than once increases as the number of gets increases. The results are shown in figure 4.

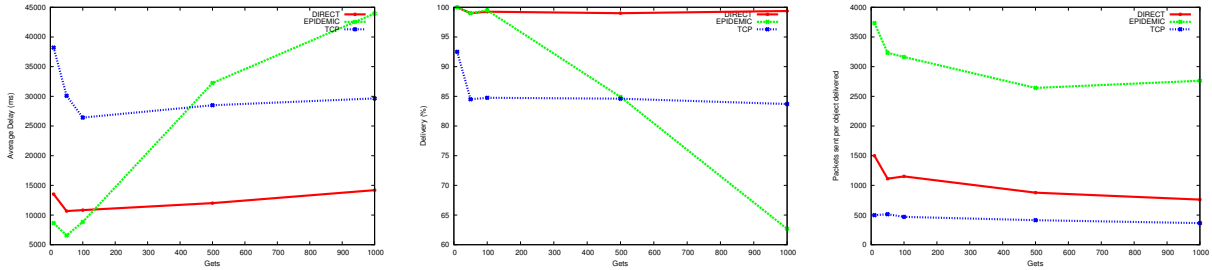
Epidemic routing has a hard time keeping up with deliveries. Its broadcast nature starts to affect performance. Delay increases drastically in both configurations and delivery drops accordingly. TCP and DIRECT show the expected behavior, with DIRECT maintaining an almost perfect delivery record while keeping delay very low. In terms of packet overhead TCP remains the leader with the lowest values but DIRECT doesn't come in far behind. The high traffic allows TCP (AODV) to keep an up to date routing table and hence maintain a low overhead.

Request duplication is something that gives a great advantage to named content. To evaluate how much impact it would have we



a) Average delay b) Average objects delivered c) Packets per object delivered

Figure 3: Results by grid square size



a) Average delay b) Average objects delivered c) Packets per object delivered

Figure 4: Results by gets

varied the number of objects created from 1 to 100 (the default value). When only one object is created all the requests are for that object. The results for these configurations are presented in figures 5.

Because of the rules for object retrieval the first values in the graphs exhibit "strange" behavior. Since a node can only issue a request for an object once you can't have more requests than objects. So the first value is for 63 requests and 1 object.

In terms of delay the cases with high request duplication (1-2 objects) work amazingly well under DIRECT and epidemic routing with an average delay of under 700ms each compared to over 50000ms for TCP. Even in the worst delay case DIRECT is still almost 6 times quicker than TCP. Delivery is maintained at 100% for both configs for DIRECT and epidemic routing while TCP stays in the low 90s.

The amount of data sent shows an interesting phenomenon which is the overhead suffered by epidemic routing at trying to distribute that single object, it is partially an artifact of our implementation of epidemic routing. In DIRECT nodes keep track of what the state of a request is and where the object has been sent, but in epidemic routing nodes have to negotiate all the time to make sure they have all objects.

Our final experiment addresses how the different protocols perform according to the size of the objects. Figure 6 shows the outcome. As the size of the objects grow, it is more likely that retransmissions will be needed. In the case of DIRECT and epidemic routing, where this happens on a hop-by-hop basis, the recovery is straight forward. As we add hops to the transmission, as is the case for TCP, it becomes hard to recover from losses. Our TCP simulations use New Reno and have a hard time with multi-hop long flows. Also, the larger the objects the more traffic the network has to transmit. These configurations use our default values of 100 gets over 20 minutes so the load is not too high, but you can see the impact the object size has on epidemic routing.

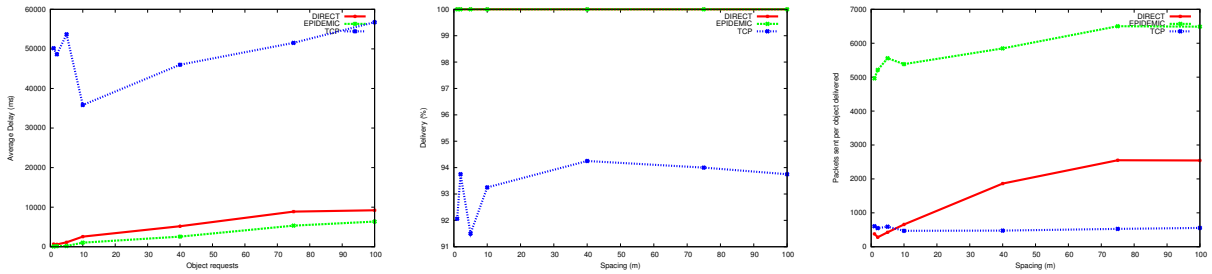
Transmission delay is much lower by using the hop by hop mechanism of DIRECT and epidemic routing with respect to TCP. In a traditional wired network this would have been the opposite since a hop by hop transmission by nature adds delay at each hop. You need to wait until the object is received before you can start sending it. This happens slightly on our scenario too. We plan to investigate what is a good middle point for object size that balances overhead.

In terms of objects delivered, DIRECT is able to maintain near perfect delivery. DIRECT needs less than half the overhead that epidemic routing incurs in packets per object, albeit still more than TCP.

5. CONCLUSIONS AND FUTURE WORK

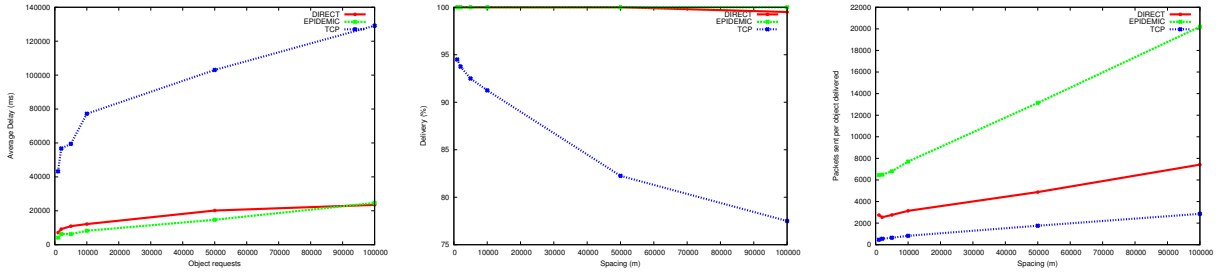
We have implemented the initial version of DIRECT under Qualnet and performed preliminary evaluations. Without any optimizations, DIRECT already shows tremendous promise in terms of delay and delivery compared to epidemic or on-demand routing schemes. We used a "mobile grid" topology in which every node participates in generating traffic to compare the performance of DIRECT with the other schemes. Under all scenarios, DIRECT showed very low delay (even with TCP having a round-trip advantage) and was at least as efficient as epidemic routing. Delivery was kept near 100% within the time limit, beating TCP every single time. Integration with the other components of a DTN architecture will be crucial for efficient performance.

The work we have presented does not address specific naming issues and how naming affects content dissemination; however, the name ontology adopted in such an architecture will play a key role in the development of any future routing infrastructure, including DIRECT. In addition, caching strategies have been widely studied in the past and are currently being studied for DTN environments, and we intend to evaluate how different caching strategies affect the performance of DIRECT. Lastly, our architecture calls for multiple query types, including publisher-based, bounded and continuous, and this requires further analysis for optimization purposes.



a) Average delay b) Average objects delivered c) Packets per object delivered

Figure 5: Results by objects



a) Average delay b) Average objects delivered c) Packets per object delivered

Figure 6: Results by object size

6. REFERENCES

[1] J. Boice, J.J. Garcia-Luna-Aceves, and K. Obraczka. On-demand routing in disruptive environments. In *In proceedings of the IFIP Networking 2007, 2007*, 2007.

[2] J. Burgess, B. Gallagher, D. Jensen, and B.N. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networking. In *Proceedings of IEEE Infocom*, 2006.

[3] B. Burns, O. Brock, and B. N. Levine. Mv routing and capacity building in disruption tolerant networks. In *Proceedings of IEEE INFOCOM*, 2005.

[4] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-tolerant network architecture, 2005.

[5] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on the design of opportunistic forwarding algorithms. In *Proceedings of IEEE Infocom*, 2006.

[6] X. Chen and A.L. Murphy. Enabling disconnected transitive communication in mobile ad hoc networks. In *Workshop on Principles of Mobile Computing, colocated with PODC'01*, pages 21–27, 2001.

[7] N. S. Djukic, M. Piorkowski, and M. Grossglauser. Island hopping: Efficient mobility-assisted forwarding. In *In proceedings of the IEEE SECON 2006, 2006*, 2006.

[8] DTNRG. Delay tolerant networking research group.

[9] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli. Age matters: Efficient route discovery in mobile ad hoc networks using encounter ages. In *Proceedings of Mobihoc*, 2003.

[10] K. Fall. A delay tolerant networking architecture for challenged internets. In *Proceedings of SIGCOMM*, 2003.

[11] K. Fall, W. Hong, and S. Madden. Custody transfer for reliable delivery in delay tolerant networks.

[12] K. Harras, K. Almeroth, and E. Belding-Royer. Delay tolerant mobile networks (dtmns): Controlled flooding schemes in sparse mobile networks. In *IFIP Networking*, 2005.

[13] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and the consequences of human mobility in conference environments. In *Proceedings of ACM Sigcomm (DTN Workshop)*, 2005.

[14] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[15] S. Jain, K. Fall, and S. Patra. Routing in a delay tolerant network. In *Proceedings of ACM SIGCOMM*, 2004.

[16] H. Jun, W. Zhao, M. Ammar, C. Lee, and E. Zegura. Trading latency for energy in wireless ad hoc networks using message ferrying. In *Third IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'05)*, 2005.

[17] J. Leguay, T. Friedman, and V. Conan. Dtn routing in a mobility pattern space. In *Proceedings of SIGCOMM 2005*, 2005.

[18] Q. Li and D. Rus. Communication in disconnected ad hoc networks using message relay. In *Journal of Parallel and Distributed Computing*, volume 63. Academic Press, Inc., 2003.

[19] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, 2004.

[20] S. Merugu, M. Ammar, and E. Zegura. Routing in space and time in networks with predictable mobility.

[21] M. Musolesi, S. Hailes, and C. Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *Proceedings of SIGCOMM 2005*, 2005.

[22] J. Ott, D. Kutsher, and C. Dwertmann. Integrating dtn and

manet routing. In *In proceedings of the ACM SIGCOMM Workshop on Challenged Networks (CHANTS), 2006*, 2006.

[23] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks.

[24] T. Small and Z. J. Haas. Resource and performance tradeoffs in delay-tolerant wireless networks. In *Proceedings of ACM SIGCOMM*, 2005.

[25] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *Proceedings of SIGCOMM 2005*, 2005.

[26] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous correlated mobility. In *In proceedings of IEEE PERCOM, on the International Workshop on Intermittently Connected Mobile Ad hoc Networks (ICMAN), 2007.*, 2007.

[27] W. Su, S. Lee, and M. Gerla. Mobility prediction and routing in ad hoc wireless networks, 2000.

[28] A. Vahdat and D. Becker. Epidemic routing for partially connected ad-hoc networks, 2000.

[29] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of ACM Mobihoc*, 2004.

[30] W. Zhao, M. Ammar, and E. Zegura. Controlling the mobility of multiple data transport ferries in a delay tolerant network. In *Proceedings of IEEE Infocom*, 2005.

[31] W. Zhao, Y. Chen, M. Ammar, M. Corner, B. Levine, and E. Zegura. Capacity enhancement using throwboxes in dtns. In *In proceedings of the IEEE Intl. Conf. on Mobile Ad hoc and Sensor Systems (MASS), 2006*, 2006.