DTLSR: Delay Tolerant Routing for Developing Regions

Michael Demmer
Computer Science Division
University of California, Berkeley
Berkeley CA 94720, USA
demmer@cs.berkeley.edu

Kevin Fall
Intel Research Berkeley
2150 Shattuck Ave, Penthouse Suite
Berkeley CA 94704, USA
kfall@intel.com

ABSTRACT

We consider the problem of routing in delay tolerant networks deployed in developing regions. Although these environments experience intermittent connectivity (hence the desire to use DTN), in many cases the topology has an underlying stability that we can exploit when designing routing protocols. By making small, yet critical, modifications to classical link state routing, we derive a more effective algorithm capable of leveraging predictions of future link uptimes. We describe a complete and fully-implemented protocol, capable of being deployed in the DTN reference implementation without modification. Using a simulation incorporating real-world network characteristics, we demonstrate that our system operates effectively when conventional routing and forwarding may fail.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing Protocols

General Terms

Algorithms, Performance, Design

Keywords

Routing, Delay Tolerant Networks, Developing Regions

1. INTRODUCTION

Although the Delay Tolerant Networking Architecture [4, 9] seeks to address the needs of challenged networks such as those in developing regions, the lack of a deployable routing protocol implementation inhibits the ability of the DTN reference implementation [6] to be useful for real world deployments. Perhaps ironically, one of the challenges in developing a robust DTN routing algorithm is the large scope for which the architecture is applicable [10]. In particular, DTN has been proposed for use in mobile ad-hoc networks, deep-space environments, underwater and nautical buoy deployments, sensor networks, as well as many developing region contexts. These environments display a wide range of connectivity and node characteristics. While the DTN architecture provides a common service framework for data forwarding, it does not mandate a particular routing protocol, instead offering a framework where a variety of routing protocols can be used. Such protocols may be applicable to the entire scope of network types for which DTN is designed, or may instead be applicable for use in specific areas within a network that exhibit particular properties.

In this work we focus on the types of networks typical of rural areas in developing regions, based on our experience deploying wireless networks in countries such as India, Cambodia, Uganda and Ghana. These environments experience intermittent connectivity due to a variety of reasons, yet the topology often has an underlying stability. In particular, the set of neighbors for any node is usually small and does not change frequently; network dynamics result from link failures rather than unpredictable node mobility.

In a common scenario where connectivity is provided by dialup modem or satellite links, nodes are typically fixed computers in an information center or kiosk, and outages occur due to unmanageable congestion or signal loss [8]. These outages create a network partition between a subset of the network (i.e. computers in the center) and the rest of the Internet or extended LAN. In the case of long-distance point-to-point wireless links [17], interference and/or unreliable power can cause short-term outages on the links, yet the placement of towers and antennas means that neighbor relationships between routers are fixed over the long-term.

Even in cases where connectivity is provided by ferrying data (i.e. data mules or buses) network dynamics have a significant regularity, as the mobile routes are known in advance. For example, in the environments typified by DakNet [18], store-and-forward connectivity is provided by physical transfer of data using motorcycles. Because the vehicles travel on regular routes and visit the same set of end points, they have a contact regularity that does not really fit into the mobile *ad-hoc* networking (MANET) model of a mobile node, generally characterized as having more random mobility.

Thus, in contrast to highly mobile or unstructured environments, many networks in developing regions have a comparatively stable and predictable underlying topological structure. Despite periodic link outages, the relationships between nodes are stable, and the network topology is more akin to classical wired networks than to MANETs with random node mobility patterns. In this work we leverage this insight to develop an effective routing protocol for intermittent networks in developing regions.

2. ROUTING PROTOCOL DESIGN SPACE

To design a routing protocol that is well suited for DTN operations in the types of networks we are interested in, we first examine standard distributed routing approaches, more recent MANET routing protocols, and contemporary DTN routing protocols. We wish to develop a protocol that simultaneously provides good performance for the above-mentioned environments, while avoiding

NSDR'07, August 27, 2007, Kyoto, Japan.

unneeded complexity. Therefore, we leverage existing protocols whenever possible, and only augment them where we believe the benefit to be substantial.

2.1 Standard Approaches

Standard well-known distributed algorithms for routing include distance vector (DV), path vector (PV), and link state (LS). For the purposes of this discussion, DV and PV have fundamentally the same characteristics, so we discuss DV without loss of generality. Unless otherwise specified, we describe the ordinary operation of DV and LS that select paths based on a message's destination and make independent forwarding decisions at each hop.

A node's correct DV operation depends on the correctness of routing tables present at its neighbors. Nodes provide only their chosen next hop for each destination, so the path selected is based on the use of a common routing metric implemented by all participants in the routing graph. Failure to use the same metric leads to loops or dead-ends. DV has been popular due to its simplicity; early versions computed a single shortest path based on hop count, though more sophisticated variants (e.g. EIGRP [1]) include alternate metrics and multi-path routing. Concerns with DV routing generally include scalability and convergence time problems. In particular, changes are propagated by (re)announcing an entire routing table, that are processed by each node (taking some time) and the resulting table is again advertised.

LS routing involves informing all nodes of the network topology and allowing each node to independently compute its preferred next hop(s). The topology is generally flooded using link state announcements (LSAs), which may also contain other information relating to a node's state (e.g. buffer occupancy, location, etc). Because a node receiving LSAs becomes informed of the entire network topology, it can make full *path* decisions for traffic, rather than next-hop decisions, although this is atypical in practice. To implement path selection, forwarding nodes would need to support some form of source routing. Drawbacks of LS routing are mostly issues of scalability, as each node must store the full network graph and recompute paths after modification. A common technique to improve scalability is to divide the network into sub-regions (such as in OSPF *areas* [16]), using different protocol instances in each region and an inter-region protocol to span the divisions.

Selection of entire paths (instead of only next hops) can also be implemented using tag switching as with multi-protocol label switching (MPLS [19]). Tags can be assigned using a distributed or centralized routing algorithm based on a function of the message or other network properties Message-to-tag bindings are then distributed to label switches using some update protocol or can be piggybacked on a routing protocol. Tags may be also be re-assigned or pushed and popped from a label stack at each hop.

2.2 MANET Routing

Mobile ad-hoc routing is focused on situations where mobile nodes act as routers. MANET routing schemes fall broadly into the two major categories of proactive and reactive. Proactive protocols compute routes ahead of time; reactive protocols compute a route to a destination when traffic for the destination is ready to be sent. Reactive protocols save network resources by not advertising routes that are never used, but initial traffic for a new destination may be delayed as the route discovery process takes place on-demand. Typically, MANET protocol evaluation is performed using network simulation with a mobility model dictating where nodes move. While nodes move, some network path (a cascaded list of nodes) is generally assumed to be available between any sender and receiver.

DV, LS, and source routing have all been used in MANET contexts. In each case, the goal of routing convergence is to determine an immediately available path from source to destination. In areas where node coverage is dense, this approach may be appropriate. For rural areas in developing regions, however, this is rarely the case. Moreover, while MANET routing tends to focus on selecting paths from many options, we are instead interested in being as efficient as possible with the few paths we have available.

2.3 DTN Routing

When an end-to-end path may fail to exist between a source and destination, both standard and MANET routing protocols no longer suffice. As a result, a number of proposals for disruption or delay tolerant (DTN) routing have recently surfaced. These schemes do not assume that an end-to-end network path necessarily exists, but rather than such paths(s) exist over time. In addition, routing information is not always assumed to be 100% accurate – a node may only have a probabilistic chance of successfully delivering a message. Thus, message *replication* is typically used to enhance delivery probability. Due to the richness and apparent novelty of the DTN routing problem, it has been a very active area of research. Although numerous routing designs have been proposed, few have been used in practice; we now briefly discuss a few that have.

The simplest DTN routing protocol is flooding or epidemic flooding. In this scheme, messages are simply copied to any node that is reachable and does not already have a copy of the message. As new nodes become reachable due to mobility or other reasons, additional copies are made. Because of the overhead of these schemes, they are generally deemed to be too expensive for practical use, although they have been used for small networks (e.g. in Zebranet [14]).

The Prophet routing protocol [15] bases the likelihood of a next hop being able to successfully deliver a packet on its previous behavior, and only replicates a message if this probability exceeds a threshold. This protocol has been used in a real application – to provide basic Internet access (web, email) for Reindeer herders in northern Sweden [7].

The Maxprop and RAPID protocols have been used in the context of DieselNet, a network of buses in Amherst, MA [2, 3]. These protocols rely on distributed algorithms to optimize delivery using constrained replication in the face of limited storage and bandwidth resources. Both Prophet and the DieselNet approaches assume a fairly random connectivity graph, unlike the networks we typically see in developing regions, and the protocols reflect this assumption in their structure.

Finally, in work that is most closely related to ours, Seth, et al. [21] propose an architecture for connecting rural kiosks in developing regions using a combination of DTN routing over regular bus routes and proxies connected to the Internet that use a distributed hash table for node location. The authors suggest that one limitation in their current system is the lack of a deployable DTN routing framework, for which this work would be appropriate.

3. WHY LINK STATE

Although there is a wide range of starting points from which we may base a design for DTN routing in our networks of interest, we believe a modified standard link state approach offers the greatest flexibility and suitability to our needs.

3.1 Features of Link State

Link state LSAs, by definition, convey the connectivity status of nodes in the network, and when aggregated provide a complete picture of the topology. Therefore, collecting sets of LSAs over time gives the time evolution of the network topology. This is pre-

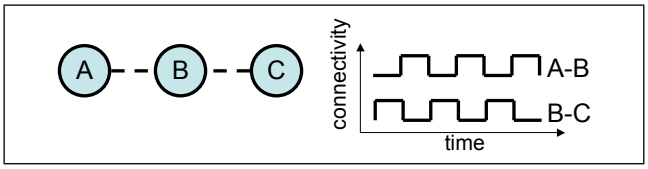

Figure 1: Simple three node network with no end to end path. Conventional routing protocols will never discover connectivity between A and C.

cisely the type of information needed to compute paths over time and multi-path routes. In addition, LSAs can easily carry additional information that may be of interest. For DTN, normal LSAs can be expanded to include buffer occupancy, and shortest path computations can be weighted based on both link availability and buffer occupancy [12].

Although MANET style link state approaches may be useful for the connected components of the topology graph in our scenario, protocols such as OLSR [5] are designed both for large scale, which is not typical of the types of access networks we typically find in developing areas, and for networks with end-to-end connectivity. Therefore, it is both more complicated than we require in terms of scaling and yet insufficient for helping to determine routes over time we desire for DTN routing.

Turning to DTN routing approaches, much of the existing work is targeted at supporting opportunistic connectivity and involves message replication. Given that our scenarios do not typically involve much mobility, and the mobility that is present is relatively periodic (e.g. buses), the benefit of replication seems limited. Thus, even though algorithms like RAPID or MaxProp may be adequate for our networks, a simpler and more efficient approach is to modify a standard link-state algorithm.

In choosing link state as the basis for a DTN routing algorithm, we inherit some other appealing properties of link state routing. For example, in connected portions of the network, LSAs are propagated relatively quickly, leading to fast convergence. Furthermore, the size of an LSA is limited, as the size of the networks we have found to date are small (significantly less than 100 nodes). If we augment nodes with forwarding based on tags or source routes, we can also support multiple forwarding metrics and paths simultaneously (e.g. based on message size, priority, message content).

Link state also has two practical features which are of interest in field deployments. First, because nodes build an entire topology picture, remote management and network configuration is simpler relative to other schemes – the entire network topology is available by interrogating any single node. Finally, the "control" network carrying LSAs need not necessarily be the same network that carries message data. This separation can be useful in scenarios where low-bandwidth routing information can be cost-effectively transferred by other means (e.g. by SMS) but data traffic cannot.

3.2 Modifying Standard Link State

Consider the example network shown in Figure 1, where at no point in time is there an active end to end path between nodes A and C. DTN's store and forward operation can clearly forward a message from A to B, then queue it, waiting for the B-C link to become active. This is not possible in conventional routing and data packet forwarding; when links are down, they are removed from consideration for routing (i.e. partitions equal failure). Thus, the route computation would never discover the fact that with store/forward, a message could be conveyed from A-B-C just fine.

To modify standard link state, we wish to consider that even though a link may not be available currently, it may become so in the future. For our networks of interest, the future probability a link may become available is likely to be related to its past. Thus we first require LSAs that can persist across partitions, and we also need a modified shortest path computation that can leverage a link's dynamic history.

4. THE DTLSR PROTOCOL

Delay Tolerant Link State Routing (DTLSR) is modelled closely on classic link state algorithms. As the network state changes, link state announcements are flooded throughout the network. Each node maintains a graph representing its current view of the state of the network, and uses a shortest path computation (e.g. Dijkstra) to find routes for messages.

Borrowing a similar idea from OSPF [16], each node in the system is assigned to an administrative *area*, and a link state protocol instance operates only within a single area. This helps to constrain the size of the network graph and limits the scope of announcement messages, if required. Nodes that have neighbors in other areas learn the set of endpoint identifiers reachable via the other area and announce themselves as a gateway to those endpoint identifiers.

4.1 Messages and Flooding

Link State Announcement (LSA) messages convey the network connectivity for a node in the system. Each LSA contains the source node's endpoint identifier, a sequence number, an area identifier, and a vector of link state information. The per-link information includes the next hop destination, measured (or configured) bandwidth and latency, configurable cost, and queue occupancy. In our current implementation, we include the full set of link information in each LSA update, and do not separate the vector of neighbors from the details about each neighbor.

Unlike some classical approaches, we do not rely on announcements to determine the next-hop neighborhood relationships. Because DTN is an overlay network, the different transports (called convergence layers) may have protocol-specific mechanisms for discovering nearby nodes. The convergence layers therefore issue upcalls to the routing layer when connectivity is detected (or lost) between nodes, and the routing layer distributes this connectivity information throughout the network.

To distribute these messages throughout an area efficiently, we implemented a constrained flooding algorithm within the DTN bundle forwarding layer. LSA messages are sent as bundles with a wildcard destination; in our current implementation, we use `dtn://*/dtlsr?area=xyz`. Thus the bundle is forwarded to all adjacent nodes (due to the $\star$ wildcard), delivered to the router application (due to the `dtlsr` service), but forwarding is constrained to nodes in area `xyz`.

4.2 Update Frequency / Expiration

In traditional link state routing, nodes are responsible for both determining reachability to their immediate neighbors, and for implementing a flooding protocol to distribute LSAs throughout the network. When a new link is established or a partition is healed, nodes on either side of the link learn of each others' reachability when they receive acknowledgments for their queries (e.g. the OSPF HELLO protocol). If a node A fails to hear enough HELLO responses from its neighbor node B after some time, A infers that the link between A and B is down.

In contrast, DTLSR operates in a DTN, where nodes are assumed to have long term storage they can use for store-and-forward operations on messages even when some links are down. That is, a path may be a valid DTN route even if it does not provide current con-

nectivity to a particular destination, and the lack of recent contact with a neighbor does not imply that the node is necessarily down.

Thus, the first major distinction of the DTLSR algorithm as compared with conventional LS is that LSAs are sent with very long lifetimes (on the order of hours or even days), and all nodes maintain a persistent cached copy of the most recently LSA from other nodes in the area. Nodes therefore queue LSA updates in case of a network partition, and when a link is established to a neighbor, the flooding process checks whether or not each cached message needs to be sent to that neighbor.

This feature means that a node does not need to periodically rebroadcast LSAs to ensure propagation to all nodes. Regardless of what the network connectivity state is when an LSA is generated, all eventually reachable nodes will receive all LSAs. Therefore a DTLSR node only needs to generate an LSA if it determines there is some new information to convey that affects the route weight computations (described below). In our current implementation, we only send LSAs in response to link state changes, and all LSAs have a lifetime of one year.

4.3 Calculating Best Paths

Calculation of shortest (or "best") paths using conventional LS routing is straightforward. If a path is currently available between two nodes, then traditional metrics such as hop count work adequately for many situations. The challenge for DTLSR is determining how to utilize paths that may not be available at the time when a node needs to make a routing decision, but which may be available in time before a message expires.

Building on conclusions from prior work on DTN routing [12], we chose to focus on minimizing the expected delay for all messages as a proxy for maximizing the overall delivery rate. However, because a node may have an incomplete or inaccurate view of the network, we follow an approach of minimizing the estimated expected delay (MEED), introduced by Jones, et al [13].

When computing paths, a node can use local knowledge about link connectivity, queueing, and traffic, the current snapshot of the state of the rest of the network, and historical data conveyed in LSAs or calculated locally. From this information, the node calculates an estimate for the delay that it would take to forward a message using the given link. In our current prototype, we use a simple heuristic that uses only the most recent network snapshot.

We distinguish between links thought to be available from those thought to be down when calculating paths. For available links, the delay to send a message on a link includes the time to first drain the link queue. Thus using estimates of the number of messages and total size of the link queue ($qnum$ & $qlen$), the per-message latency of the link ($latency$) and the bandwidth of the link (bw), we calculate the estimated delay as $qnum \times latency + qlen \times bw$.

For unavailable links, we use the duration of the outage so far (capped at 24 hours) as an estimate of the delay – this simple heuristic captures the belief that links which have been down for a long time are unlikely to come up soon, and allows us to easily include a time dimension when computing the best path. This approach also effectively ignores the transmission and propagation delay of links considered to be down; this is reasonable, as outage durations tend to be far in excess of propagation and transmission delays on the types of links we are considering.

The key distinction between DTLSR and conventional LS routing with respect to best path computation is that in DTLSR even links to currently-unreachable nodes are eligible components of best paths. In contrast, classical link state routing removes unavailable links from path consideration, and thus can only compute paths that are available at the time of route computation.

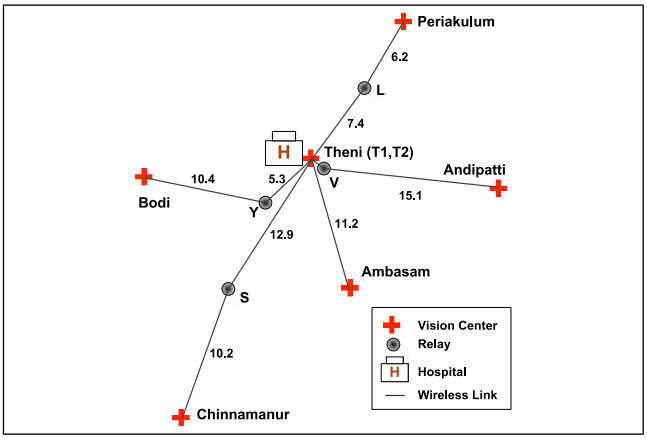

Figure 2: Map of the Aravind wireless network used as a basis for the simulation experiments.

5. EVALUATION

We implemented the DTLSR protocol in C++ as a routing algorithm option for the DTN reference implementation (DTNRI) [6]. All communication, for both application traffic and LSAs, is performed using DTN bundles [20]. Using bundles to carry LSAs allows the algorithms and protocols studied to be deployed in a variety of operating environments, thanks to the DTNRI's support of various underlying transport protocols.

This decision also allows us to compare the behavior of DTLSR with conventional LS routing by using different values for the DTN bundle lifetime field. More specifically, using short lifetimes relative to link down times emulates Internet-style forwarding. Sending messages with long lifetimes uses DTN's long-term store and forward capability to queue messages and wait for a partition to heal.

Also, we can select one of two link weight functions used for route computation. The first (called LSR) assigns a constant weight to links that are up and an infinite weight to links that are down. The second (DTLSR) is the weight function described above in Section 4.3 that estimates the delay for each link.

5.1 Protocols Compared

In the following discussion, we denote a particular algorithm as FN$[E^{lsa}_{app}]$. FN is one of the two weight functions mentioned above. The E_{app} subscript expresses the lifetime of application messages, and the E^{lsa} superscript expresses the lifetime of LSA messages.

The LSR$[E^{30sec}_{5sec}]$ algorithm is most akin to classical link state routing and Internet style forwarding. LSA messages have a short lifetime of five seconds and are broadcast both after each link state change and also periodically every five minutes. As mentioned above, the LSR weight function only chooses paths that are known to be up, so this algorithm will only be able to route to the currently-connected component of the network. Furthermore, the lifetime on each application message is set to 30 seconds, so a message is only delivered successfully if the system finds an end to end path at the time when the message was sent or soon thereafter. We expect this algorithm to do poorly, as it cannot take advantage of the DTN long-term store and forward capability.

In the LSR$[E^{12hr}_{5sec}]$ algorithm, we use the same LSR weight function and LSA parameters as the previous algorithm, but increase the application message lifetime to twelve hours. This means that a router may queue application messages at the source for some time, waiting for a route to arrive. However, application messages are only forwarded if the router finds an available end to end path.

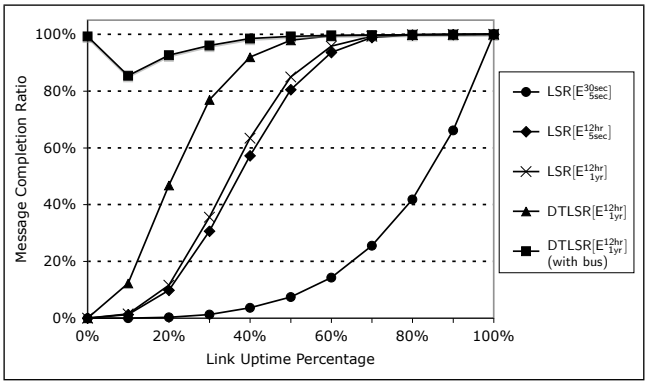

Figure 3: Message delivery for various routing algorithms on the simulated Aravind network. LSR[E^{30sec}_{5sec}] emulates traditional link state and Internet-style forwarding, exhibiting poor performance as link quality degrades. In LSR[E^{12hr}_{5sec}] and LSR[E^{12hr}_{1yr}], long message lifetimes enable queuing until an end to end route is present. DTLSR[E^{12hr}_{1yr}] further improves delivery by using partially down routes and leveraging predicted future uptimes, and is the only algorithm to gain additional benefit from the bus links.

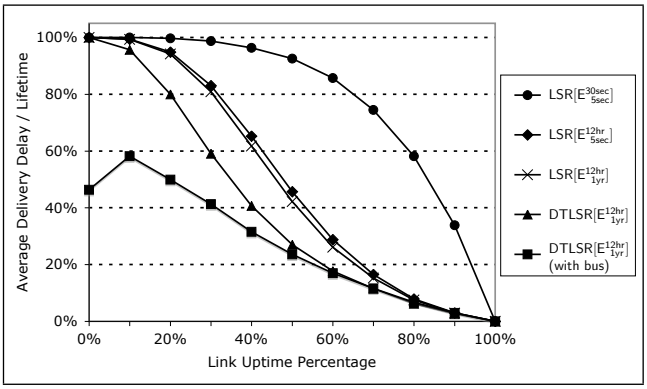

Figure 4: The percentage of a message's lifetime that it spent in the network for various routing algorithms on the experiments described in Figure 3. Results show that DTLSR accurately chooses between the wireless links and the bus link to minimize the delay in most cases.

In the LSR[E^{12hr}_{1yr}] algorithm, we use the same weight function as before, but adopt the changes described in Section 4.2 by increasing the LSA lifetime to one year (essentially infinite). Nodes store the most recent LSA received from all other nodes, so when a partition is repaired, LSA updates stored on one side of the partition are transferred to the other right away. We therefore only need to send new LSAs when the link state changes, not every five minutes.

Finally, the DTLSR[E^{12hr}_{1yr}] algorithm also sends LSAs with a one year lifetime, but uses a weight function that computes the expected delay of the link, as described in Section 4.3. Therefore this algorithm may select paths that include links which are down at the time of route selection but are believed to come up again in the future, before the application message expires.

5.2 Simulation Scenario

The DTNRI can be deployed in the field as a routing daemon, and also compiled into a single process discrete event simulator. For this evaluation, we use its simulation capability to run experiments modeled on a long-distance wireless network our research group has deployed to connect remote rural vision centers with the Aravind Eye Hospital in Tamil Nadu, India [22]. Figure 2 depicts a map of this network, including the distances between the five centers, each of which is connected to the central hospital in Theni using wireless relay nodes.

The goal of this set of experiments is to compare the performance of DTLSR to a traditional link state routing algorithm as we vary the network connectivity. We fixed the bandwidth for each wireless link at 1Mbps and 10ms latency, and arrange to send a 64KB application message once per hour from each center to all other centers. We then bring the wireless links up and down randomly to achieve a desired average uptime percentage, and vary this percentage for the different experiments.

In addition to the wireless links, we also ran experiments in which we simulated buses that travelled between the hospital at Theni to each of the vision centers. We assume the drive takes two hours, the bus lingers at each center for five minutes, and when connected, the bandwidth between a bus and a center is 100Mbps. We assume buses always reach their destinations, and have no storage limit. Thus, if a router always forwards using the buses, all messages would ultimately be delivered, but only after a relatively long delay. Also, only the DTLSR algorithm can take advantage of these links, as discussed in Section 3.2. We therefore only plot results for the buses using the DTLSR algorithm, as the other schemes have identical results with or without the buses.

All message activity, including both application messages and LSA updates, was logged at each hop. We then calculated the end to end delay for each message that was delivered successfully. We simulated one month of operation for each experiment.

5.3 Delivery Results

Figure 3 shows the message completion ratio (i.e. the percentage of the transmitted application messages that arrived before they expired or the simulation time ended) as we varied the average uptime of the wireless links.

As expected, the LSR[E^{30sec}_{5sec}] algorithm achieves no progress when the link uptime percentage is low (below 20%), but improves exponentially with increased link uptime percentage, since the probability of delivering a message depends on all links along the path being up at the same time. In this experiment, routing cannot use the DTN store and forward capability effectively because application messages expire too soon.

In the LSR[E^{12hr}_{5sec}] experiment, application messages can be queued for relatively long periods of time at the source. This approach delivers a large percentage of messages successfully, provided the average link uptime is at least 60%, because the probability is quite high that an end to end path will be found before any message expires. However, the delivery ratio declines sharply as the link uptime percentage declines below 60%, resulting in a very low message completion rate with a link uptime percentage of 20% or less. The LSR[E^{12hr}_{1yr}] algorithm shows virtually identical delivery results, though it requires much less LSA traffic to update its routing state since LSAs are not sent periodically with high frequency.

Because the DTLSR[E^{12hr}_{1yr}] algorithm can estimate that a down link may come up in the future, it need not wait for all links along the path to be up in order to find a path and can instead predictively forward messages along the expected best path. Thus, DTLSR exhibits much better performance, delivering close to 80% of all messages even with only 30% link uptime.

Finally, when we add the buses to the environment, DTLSR recognizes the wireless network is performing poorly and shifts traffic

to the buses instead, resulting in high message completion ratios regardless of the wireless link uptime percentage. The initial decline in performance of DTLSR with buses between 0% and 10% link uptime percentage is the result of an imperfect route prediction algorithm; when the wireless network is very bad, DTLSR sometimes makes the wrong decision to use a wireless link as opposed to waiting for the bus. However, overall the heuristic seems to perform fairly well.

5.4 Delay Results

Figure 4 shows the average time each application message remained in the system as a function of its lifetime, i.e. messages delivered immediately spent a small fraction of their lifetime in the network, while expired messages spent 100% of their lifetime in the network. These results mirror the message completion ratio graph, as reducing the delay of messages in the system results in a better delivery ratio.

This result also demonstrates that in the scenario when the buses are included, DTLSR makes the right decision about whether to forward messages to the bus or to the wireless network, since the average delay is strictly lower when the buses are used. Were the system to have only used the buses, all messages would be delivered, but the average delay would have been much higher.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we described the DTLSR routing protocol, adapting link state routing techniques for intermittent networks in developing regions. The key insight of our approach is that because the underlying topology is often stable, fairly minor changes to a classic algorithm result in an effective routing system for intermittent networks.

We are encouraged by the success with which a simple weighting heuristic can capture the path selection criteria for these networks. To continue this investigation, we plan to explore more sophisticated weight functions along with richer state conveyed in LSA messages that more accurately account for buffer occupancy and message queueing. Additionally, we plan to explore tradeoffs in determining when to send new LSAs.

We also plan to add support for advertisement of external routes to allow areas of DTLSR deployment to interoperate by exchanging summaries of the set of reachable endpoint identifiers across an area boundary. This feature is needed for inter-area routing as well as integration with other routing protocols.

Our approach also lends itself to integration with prior work on sending erasure coded bundle fragments over multiple paths to achieve high probability that a bundle arrives at its destination without error [11]. DTLSR can calculate estimates of link delivery probabilities based on uptime history, thus a path selection algorithm is straightforward: a node iteratively selects paths using a weight function that biases for both link delivery probability and independence with other paths, continuing the process until enough paths are chosen to achieve a desired delivery probability. We also plan to develop a simple source routing mechanism to implement these decisions.

Finally, we plan to deploy DTN-enabled applications in several developing regions, including log collection and remote management for our networks in India and Ghana, a pilot telemedecine system also in Ghana, and distribution of syndicated radio content in Guinea Bissau. All of these deployments fit the general characteristics for which we designed DTLSR, and therefore we expect to gain additional insights into the characteristics of how links behave in the field and to the efficacy of our approach.

7. REFERENCES

[1] ALBRIGHTSON, B., GARCIA-LUNA-ACEVES, J., AND BOYLE, J. EIGRP - a fast routing protocol based on distance vectors.

[2] BALASUBRAMANIAN, A., LEVINE, B., AND VENKATARAMANI, A. DTN Routing as a Resource Allocation Problem. In *SIGCOMM* (Aug. 2007).

[3] BURGESS, J., GALLAGHER, B., JENSEN, D., AND LEVINE, B. MaxProp: Routing for vehicle-based disruption-tolerant networks. In *Infocom* (2006).

[4] CERF, V., ET AL. RFC 4838: Delay-tolerant networking architecture, Apr. 2007.

[5] CLAUSEN, T., AND JACQUET, P. RFC 3626: Optimized link state routing protocol (OLSR), Oct. 2003.

[6] DEMMER, M., ET AL. Implementing Delay Tolerant Networking. Tech. Rep. IRB-TR-04-020, Intel Research Berkeley, Dec. 2004.

[7] DORIA, A., UDEN, M., AND PANDEY, D. P. Providing Connectivity to the Saami Nomadic Community. In *Development by Design Conference* (2002).

[8] DU, B., DEMMER, M., AND BREWER, E. Analysis of WWW traffic in Cambodia and Ghana. In *WWW* (2006).

[9] FALL, K. A delay-tolerant network architecture for challenged internets. In *SIGCOMM* (2003).

[10] FARRELL, S., AND CAHILL, V. *Delay- and Disruption-Tolerant Networking*. Artech House, 2006.

[11] JAIN, S., DEMMER, M., PATRA, R., AND FALL, K. Using redundancy to cope with failures in a delay tolerant network. In *SIGCOMM* (2005).

[12] JAIN, S., FALL, K., AND PATRA, R. Routing in a Delay Tolerant Network. In *SIGCOMM* (Sept. 2004).

[13] JONES, E., LI, L., AND WARD, P. Practical routing in delay-tolerant networks. In *WDTN* (2005).

[14] JUANG, P., OKI, H., WANG, Y., ET AL. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS-X* (Oct. 2002).

[15] LINDGREN, A., DORIA, A., AND SCHELÉN, O. Probabilistic routing in intermittently connected networks. In *SAPIR* (August 2004).

[16] MOY, J. RFC 2328: OSPF version 2, Apr. 1998.

[17] PATRA, R., NEDEVSCHI, S., SURANA, S., SHETH, A., SUBRAMANIAN, L., AND BREWER, E. WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks. In *NSDI* (Apr. 2007).

[18] PENTLAND, A. S., FLETCHER, R., AND HASSON, A. Daknet: Rethinking connectivity in developing nations. *IEEE Computer 37*, 1 (Jan. 2004).

[19] ROSEN, E., A.VISHWANATHAN, AND R.CALLON. RFC 3031: Multiprotocol label switching architecture, Jan. 2001.

[20] SCOTT, K., AND BURLEIGH, S. Bundle protocol specification. Work In Progress. Internet Draft, Apr. 2007. `draft-irtf-dtnrg-bundle-spec-09.txt`.

[21] SETH, A., KROEKER, D., ZAHARIA, M., GUO, S., AND KESHAV, S. Low-cost communication for rural internet kiosks using mechanical backhaul. In *MobiCom* (2006).

[22] THE ARAVIND WIRELESS NETWORK. `http://tier.cs.berkeley.edu/wiki/Aravind`.