# Alternative Custodians for Congestion Control in Delay Tolerant Networks

Matthew Seligman
Laboratory for Telecommunication Sciences
mattseligman@gmail.com

Kevin Fall
Intel Research
kfall@intel.com

Padma Mundur
Dept of CSEE, UMBC
pmundur@csee.umbc.edu

## ABSTRACT

We approach the problem of handling storage congestion at store-and-forward (DTN) nodes by migrating stored data to neighbors. The proposed solution includes a set of algorithms to determine which messages should be migrated to which neighbors and when. It also includes an extension to the DTN custody transfer mechanism enabling a "pull" form of custody transfer where a custodian may request custody of a message from another custodian. This approach allows us to decouple the problem of storage allocation among a relatively proximal group of storage nodes from the overall problem of path selection across a larger network. Doing so admits the possibility of localized routing loops for some messages which has been shown to be desirable for avoiding some head-of-line blocking problems. We select eligible storage neighbors using a function of available storage and incident link characteristics. Using simulation, we evaluate this approach and show how migrating custodian storage in this fashion can improve message completion rate by as much as 48% for some storage-constrained DTN networks.

**Categories and Subject Descriptors:**
C.2.2: Routing Protocols
**General Terms:** Algorithms, Performance, Theory
**Keywords:** Routing, Delay Tolerant Network

## 1. INTRODUCTION

The Delay Tolerant Networking architecture (DTN) [5] supports a *custody transfer* concept implemented by an acknowledged transfer of data to persistent, reliable storage. A node "taking custody" of a message makes a commitment to deliver the message to its destination or another custodian node, effectively migrating one or both of the ends described in the end-to-end argument [12] to new locations. The goal of custody transfer is to use hop-by-hop (custodian-to-custodian) reliability to improve end-to-end reliability and to free retransmission buffers at a sender as soon as possible. To implement this facility, the node taking custody ("custodian") must generally reserve storage for messages it takes custody of, resulting in a reduced amount of storage remaining for either taking custody of subsequent messages or for merely doing its ordinary task of switching messages.

When faced with persistent demand, a custodian unable to release or otherwise transfer custody of its messages will ultimately exhaust its storage resources– a form of DTN congestion. This type of congestion can easily result in head-of-line blocking, preventing further traffic from flowing even when some outgoing connections are available [6]. Easing congestion at a custodian is a nontrivial task. The options include discarding messages, moving them toward their ultimate destination (typically the most desirable case), or moving them to some other place. The potential of long delays and interruptions of custody transfer operations between custodians makes the management of message migration to combat congestion especially difficult.

If complete knowledge of the network evolution is known in advance (traffic mix, buffer state, topology dynamics), a linear program (LP) can be used to compute an optimal schedule of transmissions that avoids the congestion by never oversubscribing available buffers or links [1]. Interestingly, this approach sometimes provides optimal routing/transmission schedules including loops. These loops can avoid head-of-line blocking problems by moving data away from congested nodes and possibly back again once the congestion is less severe. Unfortunately, this approach is too computationally heavyweight to be practical. Instead, DTN route selection based on more conventional routing algorithms that avoid loops [9] has been suggested.

While some of these loop-free approaches take buffer occupancy into account when making path selection decisions, they do nothing to beneficially move existing data buffered in the network to alternative storage locations that may be *further* from the destination without the aid of an oracle. Of course, in practical networks, oracles are not normally available, and the computational complexity of executing an LP for scheduling on even moderate sized networks is not realistically viable. Nevertheless, performance can suffer severely if an algorithm is selected that is unable to moderate congestion. As a consequence, we have a limited set of options: use a non-optimal global heuristic to attack the LP formulation for the whole topology, separate the buffer management mechanism from the route selection problem, or decompose the problem into separate routing domains where loops may be permitted among only a subset of the nodes. In this paper, we approach the problem using the last two options, leaving the first option for future work.


*SIGCOMM'06 Workshops* September 11-15, 2006, Pisa, Italy.

## 2. MESSAGE MIGRATION

When a DTN custodian node becomes congested, it must generally migrate its stored messages to alternative storage locations to avoid loss. If the alternative storage locations for messages are closer to the messages' ultimate destinations, it is usually preferable to migrate messages further along their conventional routing path using existing DTN custody transfer procedures. If, however, moving messages along this path will not ease congestion (or simply is not possible at some instant of time), migrations to more distant locations may still be important to avoid message loss, requiring the development of some form of message migration.

### 2.1 Layering Selection

A message migration facility for DTN custodian congestion could be implemented either using DTN-layer mechanisms, or by hiding the storage subsystem from the DTN layer and instead employing a separate distributed storage system. In cases where the potential storage nodes are relatively well interconnected, the distributed system approach may be simpler. A DTN custodian simply accepts custody for messages, stores them in the storage system while maintaining custody, and retrieves the messages at appropriate times when they must be forwarded or otherwise processed. If the storage system is intelligent with respect to load balancing and caching, the custodian need only keep track of where stored messages have been placed and retrieve them later.

The major problem with this approach is the potential for inconsistency should a custodian be unable to access its distributed storage system. In such a circumstance, the custodian may be able to begin a custody transfer with another custodian but be unable to complete. Other operations are similarly affected. This approach is, in effect, a violation of Clark's *fate sharing* argument from [3]. This argument would suggest the custodian of a message should be colocated with the message itself— that is, their fate should be shared.

Given the potential problems with the migration approach discussed so far, we turn to the idea that the migration facility be visible to the DTN layer. In other words, custody of a messages and the message itself travel together, and DTN protocol mechanisms such as custody transfer requests and acknowledgments (CACKs) are employed to effect the migration. DTN routing is used in most circumstances unless a loop is desired. In these cases a new "pull custody" operation is constructed to retrieve messages from other custodians, generally creating small-distance loops. We call the approach *Push-Pull Custody Transfer*.

### 2.2 Push-Pull Custody Transfer

Migrating of messages among and between custodians can be initiated by either a sender or by a requesting receiver. The current DTN custody transfer model [11] is sender based, such that all decisions about initiating custody transfer are *push* operations. While this approach is likely to be adequate for DTNs employing only loop-free routes, it is not sufficiently flexible for supporting short-term loops in routing paths that do not disturb an otherwise-loop-free routing protocol. In particular, although an operational loop-free DTN routing solution *could* be locally adjusted in the vicinity of a congested custodian to effect message migration, this approach would require extreme attention to detail in how routing would be locally adjusted in order to prevent global routing algorithm instability.

To avoid many of the pitfalls of locally modifying a global routing algorithm, we instead extend the custody transfer mechanism with a novel *pull* operation. In this form of custody transfer, a node with available resources can request to receive custody of messages from other nodes by means of a new *custody request* protocol operation without disrupting the global routing system.

In providing both push and pull operations for initiating custody transfer (together called *push-pull custody transfer*), we are able to locally circumvent global DTN path selection to improve congestion control. Push-pull custody transfer operates at the DTN layer, so DTN custody transfer mechanisms are available to implement the transfers. In addition, we can limit the topological scope of nodes that are concerned with buffer storage management to those nodes proximal to congested custodians. To take advantage of push-pull custody transfers, a congested custodian must make decisions about when to invoke push or pull operations to satisfy its data routing and congestion goals. This involves a set of algorithms we call *Storage Routing* (SR).

### 2.3 Joint Custody and Migration

Migration may occur after a message has been forwarded to the next hop custodian but before an acknowledgment is received from the next hop custodian. Since the fate sharing argument requires a message and its custodian to be colocated, this type of migration creates multiple custodians for the same message, which is referred to as *joint custody* in [6]. Joint custody allows more than one custodian for a given message. In this situation, migration causes message duplication that must be resolved at another custodian receiving both messages or the destination.

## 3. STORAGE ROUTING

Storage Routing (SR) is a set of algorithms invoked by a DTN custodian when congestion becomes evident and when it abates. With the onset of congestion, the algorithms provide a selection of messages to migrate and a set of alternative custodian(s) to which the selected messages are to be migrated. The custodian then invokes the conventional DTN custody transfer mechanism [13] to transfer the selected messages to alternative storage locations using the push operation. When a custodian has sufficient storage resources to take custody for messages, it invokes a retrieval algorithm to migrate, or pull, messages for which it may have had custody earlier toward their destination(s). In summary, the algorithms comprising SR include the following:

1. message selection - selects which messages to migrate to storage nodes
2. node selection - produces a set of nodes where messages should be migrated to
3. retrieval selection - selects which messages to retrieve (take custody for) and from whom

We now discuss further details influencing the design of each algorithm.

## 3.1 Message Selection

The choice of message(s) to migrate using the push custody transfer operation (also called the *push policy*) is analogous to the choice of a packet to drop (or mark in the case of active queue management approach such as RED gateways [7]) when an IP router exhausts its internal queues. Generally, either an arriving or a stored message may be selected. Push policies can be categorized by the types of factors they use to select messages. Three selection factor types considered are temporal, size, and priority.

A temporal policy selects a message to be pushed based on time. Example of such policies are pushTail (most recent arrival), pushHead (least recent arrival), pushOldestNetworkAge, where network age may be defined as time-to-live or a real-time timestamp, and pushLatestRouteAvailability, which pushes messages that utilize routes that will become available furthest in the future. A size policy selects a message to be pushed based on the size of the messages. Examples of such policies are pushSmallest and pushLargest. A priority policy selects a message to be pushed based on priority level. An example of this policy is pushLowestPriority. These examples are shown in Table 1 and do not represent a comprehensive set of options.

Push policies are important because they dictate the options a custodian has for handling congestion. A DTN custodian will vigorously attempt to avoid discarding any messages or fragments for which it has taken custody, as doing so would imply the likely total loss of the corresponding message. Although the DTN architecture suggests the idea that when a custodian becomes congested it may cease accepting custody for subsequent messages (a form of flow control), we do not pursue this particular approach further in this work because we are primarily interested in networks containing traffic sources which themselves cannot be flow controlled due to continuous generation of application data (e.g., sensor networks).

For sources which continuously produce data, flow control would only cause loss or additional delay for data being forwarded through the network. Instead, we focus on mitigating storage-based congestion using underutilized storage resources found off the selected routing path(s). In some sense, our approach could be characterized as a form of congestion management implemented using distributed storage load balancing.

The push policy selects a message currently in storage or the newly arrived message to be pushed. It is similar to a drop policy in that the policy selects a particular message, or packet, that must be removed from storage but is different in terms of the action taken. In a push policy, it is possible to delete messages, but this action is selected as a last resort, only when there is no available storage at any neighboring node.

## 3.2 Node Selection

Any custodian node taking custody of a message is required to store the message until it receives a custody acknowledgment (CACK) from another custodian node or the node itself is the destination. Once a message selection algorithm has been invoked to select one or more messages for migration, alternative custodians ("targets") for the messages must also be selected. We are usually interested in selecting targets among nodes that are near to the congested custodian in the topological (hop count) sense. Although other metrics are possible, this is the simplest to begin our discussion.

For static graphs $G = (V, E)$, we define the $k-neighborhood$ $N_v(k)$ a node $v$ as the set of nodes within minimum distance $k$ hops of $v$. More precisely: $N_v(k) = \{w | d(v, w) \leq k, w \in V\}$ where $d(x, y)$ is the smallest number of hops among all paths $x \rightarrow y$. For DTN graphs, where edges come and go, we consider an edge $e_{i,j}$ to be in $E$ if ever there exists a link between $i$ and $j$ with capacity greater than zero. The value $k$ is also called the *hop radius*; it alone does not fully reflect the cost involved in migrating a message to from a custodian to its $k$ neighborhood. Obviously, it is possible for a node in $N_v(2)$, for example, to have a lower migration cost than some other node in $N_v(1)$, especially for DTNs with highly diverse and intermittent links.

The node selection algorithm we pursue determines which nodes are good candidate targets based on an aggregate *migration cost* metric. This metric is sensitive to the following network parameters:

***Storage Cost*** : The amount of available storage used for accepting migrated messages. As in Zebranet [10], each node decides how much of its own storage to make available for storing other nodes' messages.

***Transmission Cost*** : A sum based on the constituent costs of transferring messages from the present custodian to one or more alternative custodians. This value is dependent on the latency, bandwidth, and up/down schedules of the links along the path from the current custodian to the target set.

Migration cost, $C_{c,v}(l)$, from custodian node, $c$, to a k-neighborhood node, $v$, of the selected message with length, $l$, is the weighted summation of the normalized storage cost, $S_v(l)$, and normalized transmission cost, $T_{c,v}(l)$. The minimum value for $C_{c,v}(l)$ is 0 and the maximum value is 1.

$$C_{c,v}(l) = T_{c,v}(l)\omega_T + S_v(l)\omega_S$$

The transmission cost is a function of the latency, $L_{c,v}$, on the path $c \rightarrow v$, bandwidth, $B_{c,v}$, on the path node $c \rightarrow v$, and the message length, $l$. It is defined as follows:

$$T_{c,v}(l) = \log((L_{c,v} + (l \div B_{c,v})) \div (10^{-6})) \div 10$$

In this equation, $B_{c,v}$ is the minimum bandwidth on the path from node $c$ to node $v$, and $L_{c,v}$ is the total latency on the path from node $c$ to node $v$. For normalization, a value of 1$\mu$sec as the minimal transmission cost and a value of 1000 sec as the maximum transmission cost is used. All normalized transmission costs lower than 1$\mu$sec are 0, and all normalized transmission costs greater than 1000 sec are 1. If no path exists between node $c$ and node $v$, the normalized transmission cost is infinite. A logarithm base 10 is used in this equation to highlight differences in order of magnitude rather than small deviations in nominal values.

The storage cost is a function of the available storage for migration at node $v$. For normalization, the storage cost is a ratio of the available storage, $A_v$, and maximum node storage, $Max_v$. The normalized storage cost, $S_v(l)$ is shown below.

$$S_v(l) = \begin{cases} A_v \div Max_v & \text{for } l \leq A_v \\ +\infty & \text{for } l > A_v \end{cases}$$

The weights, $\omega_T$ and $\omega_S$, are selected based on the DTN application and must sum to 1. When $\omega_T > 0.5$, migration is based more on transmission cost than storage cost. This

| *PolicyName* | *PolicyDescription* | *PolicyUsage* |
|---|---|---|
| pushTail | Selects the most recent arriving message | Used in applications where keeping messages together and in order is important such as serialized data streams (e.g., voice or video) |
| pushHead | Selects the least recently arriving message | Used in applications where locally newer data is more important than older data such as highly refreshed data (e.g., caching applications) |
| pushOldestNetworkAge | Selects the message with the highest network age | Used in applications where globally newer data is preferred over older data such as highly refreshed data (e.g., web browsing applications) |
| pushLatestRouteAvailability | Used with scheduled intermittent links to select the message that will be waiting for an available route for the longest period of time from now | Used in applications where link resource management is important such as satellite applications |
| pushSmallest | Selects the message with the smallest size | Used in data or file transfers applications where large files are more important such as in scientific data collection |
| pushLargest | Selects the message with the largest size | Used in interactive applications where small messages are more important (e.g., text messages, chat, or remote login) |
| pushLowestPriority | Selects the message with the lowest priority | Used when high-priority data is more important to deliver than low-priority data |

**Table 1: Push Policy Description and Usage**

value prefers a migration node with less available storage that is *closer* to the custodian node rather than a migration node with more available storage that is *further*. The opposite is true when $\omega_S > 0.5$

To find targets, a form of *expanding ring search* (ERS) [2] may be employed from a current custodian $c$. Starting with $k = 1$, a value of $C_{c,v}$ is computed for each $v \in N_c(k)$ according to the formula given above. The minimum cost among all nodes within radius $k$ is formed as $C_{min}(k) = \min_{v \in N_c(k)} C_{c,v}$ The target node set comprises all nodes with minimum cost: $T = \{v | C_{c,v} = C_{min}\}$. If this set is of cardinality greater than one, ties are broken by lexical order of the node ID.

## 3.3 Message Retrieval

The message retrieval algorithm determines which message is selected for migration to a custodian using a custody pull operation initiated by the custodian. In relatively simple topologies, this may be a custodian retrieving a message back it has temporarily stored in an adjacent alternative custodian.

A custodian may send a custody request when it transitions from a state of high congestion to low congestion. This is similar in spirit to a form of routing advertisement, but for reasons discussed earlier does not affect the DTN routing state. The custody request message may be sent to either a single custodian (e.g. when the sender knows the location of messages it previously migrated) or might instead use a form of group query to find other nearby custodians with messages to migrate. The decision of which messages to migrate at this point mirror the push decision described earlier, except in this case the decision is made by the receiver rather than the sender.

# 4. SIMULATION SETUP

To study the potential benefits of Storage Routing, we implemented a simple version of each of the message selection and retrieval algorithms along with the node selection algorithm described earlier. The message selection policy is pushTail and the node selection algorithm uses the procedure described in section 3.2. The message retrieval algorithm selects a message from all sufficiently small messages available in $N_c(1)$. Using simulation on a topology constructed to illustrate the behavior of SR, we evaluate the relationship between storage, link intermittency, and their effect on message completion rate (MCR). For purposes of evaluation, we make several assumptions about the network that apply to all simulations:

- No separate flow control mechanisms are in use - sources are unregulated and transmit at any time
- Nodes must take custody when forwarding messages - Nodes may refuse to take custody, but never accept data for forwarding without also accepting custody
- Custodian nodes drop as a last resort - data is only dropped when no other migration options are available
- Nodes are symmetric with respect to storage - each has a storage capacity made available to hold either their own or others' messages
- Each link has the same bandwidth in both directions
- Each link is either "up" or "down" – whenever a link is up, its capacity is fixed

## 4.1 The Simulator

A publicly-available simulation package was considered for DTN simulation, namely dtnsim [4]. Dtnsim contained a only a small subset of DTN mechanisms and in particular is missing custody transfer. Therefore, a custom simulation environment was developed based on the generic YACSIM [14] discrete event simulation engine and augmented with custom simulation software supporting intermittent links, custody transfer, and the other DTN mechanisms.

## 4.2 Network Topology

Figure 1 indicates a projection of the (dynamic) network topology used to obtain the simulation results. An edge $e_{i,j}$ is included in the topology graph if there ever exists in an interval of time during the simulation when an edge incident to $i$ and $j$ has strictly positive capacity. Using this approach, the topology includes 18 nodes, with a set of well-connected "core nodes" having average node degree 4.125. It also includes many degree-1 leaf nodes. A significant number of high degree nodes in the network core allows our simulations to realize the effect of cross-traffic while the presence of low degree nodes at the periphery supports exploration of end-user performance.

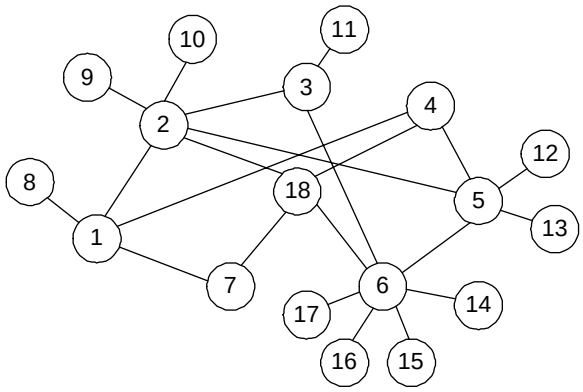


**Figure 1: Network Topology**

This topology contains a fixed schedule of intermittent links, where each intermittent link has a duty cycle of 0.25 (25% uptime and 75% downtime). Each link in the topology has a bandwidth of 10 kbps and a latency of 1 ms. At simulation time $t = 0$, all links in the network topology are up. The intermittent link schedule is shown in Table 2. Intermittent links were selected to affect a significant percentage of network traffic; therefore, several links connecting relatively high-betweenness nodes (core nodes in this case) were selected as opposed to links connected to leaves that only carry a small percentage of the overall network traffic. 75% of the intermittent links connect two core nodes together. The link intermittency schedule is arranged such that at some points in time the graph is partitioned.

| $IntermittentLink$ | $Uptime(s)$ | $Downtime(s)$ |
|---|---|---|
| (1,4) | 17.0 | 51.0 |
| (2,5) | 13.0 | 39.0 |
| (3,6) | 57.0 | 171.0 |
| (6,18) | 38.0 | 114.0 |
| (6,15) | 7.0 | 21.0 |
| (1,7) | 12.5 | 37.5 |
| (5,12) | 47.0 | 141.0 |
| (2,18) | 4.5 | 13.5 |

**Table 2: Intermittent Link Schedule**

In addition to the nominal uptime and downtime, it is interesting to note the aggregate number of intermittent links down and up as a function of time. This information is somewhat difficult to ascertain by studying the intermittent link schedule, so we present a time series in Figure 2. The graph shows the number of intermittent links at any given time assuming all links listed in Table 2 are used. This graph shows how the number of down links oscillates between 4 and 8 during steady-state operation.

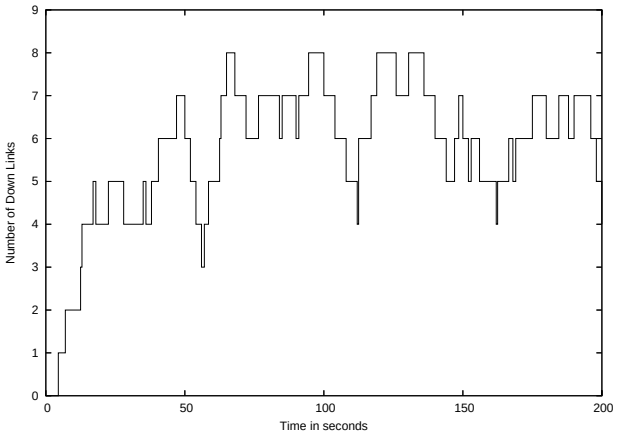


**Figure 2: Number of Down Links in Time**

## 4.3 Network Traffic

Traffic sources are modeled as periodic on/off sources with two parameters: off time and on time, where the off time and on time are fixed for each traffic source. The burst rate is fixed for a single but different among traffic sources. Given a fixed off time, a short on time reduces the traffic intensity, which tends to allow the network to continue operating without loss despite a number of links being down (ie., because previous bursts are buffered). Longer on times tend to stress the network, and over time will produce congestion and eventual loss. The burst length can be derived from the on time and nominal link bandwidth, and is expressed in terms of a number of fixed-sized messages.

The input parameters jointly affect the overall performance as they together determine both the instantaneous and average traffic intensity. Traffic arriving at the beginning of a link downtime requires node storage for the duration of the downtime, assuming no other path to the destination becomes available. Conversely, traffic arriving at the beginning of an uptime has a high probability of arriving at the next hop node right away, assuming that the transmission time is negligible in comparison to the uptime of the link. The strong joint sensitivity to delay between bursts, burst length, and intermittent link schedule requires careful consideration of these parameters when evaluating performance.

The specific traffic model selected for simulation contains a fixed off time of 5 seconds for each traffic source. In addition, each traffic source transmits 1000 byte messages. All other details, which are variable based on each traffic source, are shown in Table 3. The network traffic was run for 200.0 seconds for each simulation run to produce results discussed in the next section.

| $Src$ | $Dest$ | $StartTime(sec)$ | $Uptime(ms)$ | $DataRateDuringBurst(KB/s)$ |
|---|---|---|---|---|
| 5 | 8 | 0.1 | 750.0 | 2000 |
| 3 | 18 | 0.3 | 105.0 | 1666 |
| 14 | 9 | 0.5 | 175.0 | 2000 |
| 13 | 1 | 0.7 | 420.0 | 1666 |
| 9 | 13 | 1.1 | 900.0 | 1666 |
| 13 | 10 | 1.3 | 87.5 | 2000 |
| 6 | 3 | 1.5 | 210 | 1666 |
| 4 | 1 | 1.7 | 750 | 2000 |

**Table 3: Traffic Source Information**

# 5. SIMULATION RESULTS

To evaluate the efficacy of SR, we calculate the Message Completion Rate (MCR) of the network traffic. The MCR compares the amount of traffic sent into the network with the amount of unique traffic that emerges from the network:

$$MCR = \frac{AggregateReceivedData}{AggregateSentData}$$

We study the sensitivity of MCR as a function of node storage and link intermittency level as independent variables. For each simulation, we compare MCR results for three SR variations: no SR, SR with $k = 1$, and SR with $k = 2$. Our goal is to determine whether SR is sufficiently compelling to pursue further, and if so whether the hop radius distance has a strong influence on its performance.

## 5.1 Node Storage

The amount of storage at each node in a congested DTN has a strong effect on the MCR of network. With increasing link downtimes, larger amounts of storage are required to cope with intermittent links while avoiding loss. For our node storage simulation results, we configure four links to be intermittent, which are the first four links listed in table 2 and let node storage range from 32 KB to 2048 KB. The results are shown in Figure 3. With 32KB of storage, SR with $k = 1$ shows a 39% improvement and with $k = 2$, a 48% improvement in MCR over similar circumstances when SR is not in use.

There are two other points of interest on the graph. When the node storage reaches 1000 KB, there is no further difference between SR with $k = 1$ versus $k = 2$. When node storage is 2048 KB, it is no longer a limiting factor on the value of MCR, implying the network is no longer congested (in which case SR offers no benefit).

The largest burst of network traffic introduced to this simulated network is 1500 messages. Since each message 1 KB in size, the largest burst from one source entering the network is 1500 KB. Given one traffic source, the maximum amount of storage necessary at a given next hop node to store all messages is 1500 KB, and it could be expected this is the maximum node storage necessary to alleviate all drops due to lack of storage. As the results for 1500 KB show, this is not the case due to cross-traffic from other traffic sources. Competition exists for next hop storage between all traffic sources traversing the node.

## 5.2 Intermittent Links

The number of intermittent links (and their respective up and down times) can have a large impact on the ability of

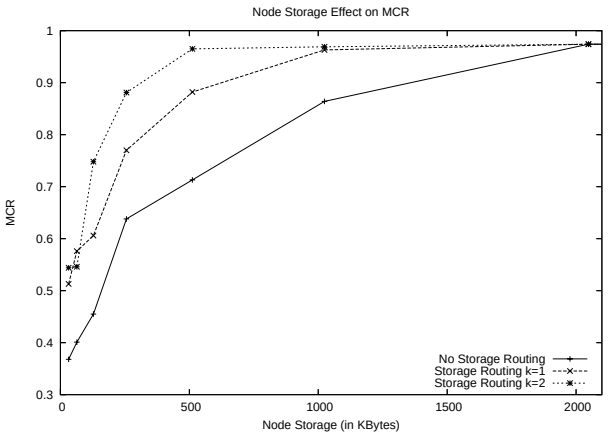


**Figure 3: MCR as a function of node storage. With 1MB of storage, SR using either $k = 1$ or $k = 2$ is sufficient to combat congestion. 2MB is required to achieve a similar result if SR is not used.**

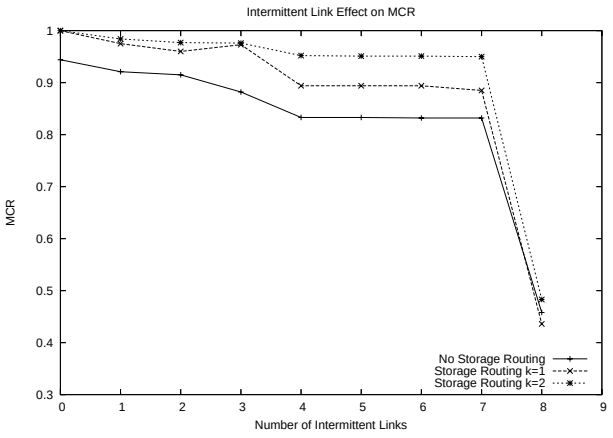


**Figure 4: Intermittent Link Effect on MCR**

a DTN to deliver messages. To explore the effect of link intermittency on MCR, we fix each node storage at 1MB and vary the number of intermittent links between zero and 8 by making links intermittent according to the order and parameters of Table 2. Results are shown in Figure 4.

Intermittent links were introduced one-by-one in each subsequent simulation run in the order in which they are listed in the table. For example, the first simulation run contained no intermittent links. The second simulation run contained (1,4) as the only intermittent link. The third simulation run contained (1,4) and (2,5) as intermittent links and so on. A total of 9 simulation runs were necessary to produce a set of results for each configuration (no SR, SR with k=1, SR with k=2). The results show that increasing the number of intermittent links either decreases the MCR or holds the MCR constant.

From 0 to 3 intermittent links, there is no difference in MCR between using SR with $k = 1$ or $k = 2$ and only a 10% difference from using no SR at all. With 4 intermittent links, all such differences become more pronounced. When the number of links is between 4 and 7, the difference in MCR increases by approximately 5% when going from no SR to SR/$k = 1$ to SR/$k = 2$. This result is moderate yet significant because, as previously shown in Figure 2, the number of intermittent links is between 4 and 7 for over 85% of the time.

When the 8th link becomes intermittent, the MCR severely decreases. At this point, the MCR not only significantly drops due to partitioning, but the MCR for the three cases studied nearly merge because none of them can make significant forward progress. When the network becomes partitioned, in addition to the severe decline in MCR, SR begins to oscillate, an effect not shown in the graph. The oscillation is a consequence of SR searching for alternate storage locations but routing being unable to carry messages to their destinations. We can expect SR to ultimately fill every potential custodian in one connected component of the topology graph until message discarding is required (or the partition heals). A subject for future research is to study this phenomenon in more detail and to develop a solution to mitigate its negative effects.

Results for the intermittent link simulations differ from those of the storage simulations due to the presence of the sharp MCR threshold for intermittent links. This is because the effect of a link becoming intermittent has a dependency on several other factors. The number of core network intermittent links introduced matters because alternative paths to the same destination or nearby node storage may be different. For example, if the order had been different and all core network intermittent links had been introduced first, the network topology would have partitioned into two subgraphs earlier in the simulation causing the large drop in MCR to occurs at number of intermittent links equals 6 (there are 6 core network links on the intermittent link schedule) instead of 8.

A link transitioning from up to down may have no effect on the MCR in cases where no traffic was using the link during its down period. This is shown in the graph by comparing MCR data points when the number of intermittent links is 4, 5, 6, and 7. Conversely, the graph also shows that an intermittent link may have a drastic effect on the MCR as well which is shown by comparing the MCR at number of intermittent links equals 7 and 8.

# 6. FUTURE WORK

The concepts introduced in this paper begin to develop the framework for addressing DTN congestion without interfering with global routing. We also introduce the symmetric push-pull custody transfer that provides a wider range of options to mitigate congestion. We now consider some variations on our work to date that may be fruitful avenues to explore.

Message migration may not need to involve transferring entire messages among custodians. The DTN architecture[13] supports a *proactive fragmentation* mechanism that can divide a message into smaller fragments, each of which can be handled by custodians individually. Proactive fragmentation was originally designed to take advantage of multiple available paths towards the same destination, but in this case it can be used to take advantage of multiple buffers of available storage to store one message. Techniques such as erasure coding, as explored in [8] for dealing with path loss, may also be applicable for encoding fragments across multiple custodians. Supporting this efficiently would require modifications to the message and node selection algorithms.

In this paper, we have not considered message priority as input to the message, node, and retrieval algorithms. In many circumstances, messages with a higher priority would be preferentially selected for transmission before being pushed to more distant storage. Modification of SR to support this capability should be straightforward, but may cause difficulties when large messages of high priority need to be handled in a highly resource-constrained network.

As mentioned earlier in this paper, we have chosen to perform message migration independently from path selection. Although joining these two functions is challenging, doing so may result in a more stable global network (fewer control loops). In particular, doing so may allow some alternative custodians to be identified that have better routes to the destination of messages.

During our exploration of the behavior of SR, we found that when the network is partitioned, our simple implementation can begin to operate in an oscillatory fashion where messages are moved among custodians but little forward progress of messages is made. There are numerous techniques for avoiding such forms of oscillations in routing protocols (hold times, split horizon, etc). To what extent these techniques or others could be employed to address the oscillation issue remains to be explored.

Our description of the push-pull custody model involves identifying messages largely by identifier. It would be possible to construct a more general and capable matching function supporting the ability to express a custody request containing a form of content-based query. Such a system is a form of publish/subscribe, and custodians could be modified to act not only as intermediate destinations for messages but also as long-term repositories where other DTN nodes could visit to obtain messages matching certain criteria. Such a capability could be especially useful for content distribution where significant temporal locality is present.

# 7. CONCLUSION

Storage congestion in DTN represents a significant design challenge for several reasons. Lack of reliable, timely feedback regarding the state of the network makes traditional techniques like rate based and window based congestion con-

trol nearly ineffective. In addition, the reliability model of custody transfer, where a custodian essentially promises not to discard data for which it has taken custody, leaves few options for a congested custodian to remedy its situation. While a global DTN routing protocol could be modified to take storage into account, doing so may disturb desirable stability properties of routing protocols such a loop-free operation. As a consequence, we have developed an approach to congestion control that can be decoupled from path selection decisions.

Although DTN nodes may be constructed with large amounts of persistent storage, many DTN applications are likely to employ nodes that are constrained in one or more of its parameters (power, storage, etc) such as sensors or small embedded systems. With networks of such devices in mind, we introduce Storage Routing (SR) to employ available storage in neighborhoods of congested custodians to mitigate blocking due to storage exhaustion without directly affecting route selection. In support SR, a push-pull custody transfer model is introduced to support the initiation of message migration from either a sending or receiving custodian. This mechanism allows custodian nodes with limited storage to push messages to other custodian nodes for storage when deemed necessary to avoid message loss. It also allows custodians with available storage to request custody of messages from other, more congested, custodians.

In evaluating whether SR is a promising approach to DTN congestion, we measured the influence of node storage and number of intermittent links on the Message Completion Rate (MCR) using simulation. It most cases, use of SR improves over its non-use, in some cases significantly, especially when the amount of node storage is limited in comparison to the network traffic volume. Also, we find SR provides benefits when the number of intermittent links is moderate but may not provide a significant benefit when the network is frequently partitioned, although more investigation of this topic would be appropriate.

Although we find SR is capable of providing significant benefits in terms of MCR, we also observed that the degree of benefit can vary significantly depending not only on the amount of link intermittency but also on the particular order and combination of intermittent links. Despite these concerns, we believe SR can be an effective technique for many DTN application scenarios, is relatively easy to implement, and avoids the concerns related to modification of a global routing protocol.

# 8. REFERENCES


[1] J. Alonso and K. Fall. A Linear Programming Formulation of Flows over Time with Piecewise Constant Capacity and Transit Times. Technical Report IRB-TR-03-007, Intel Research Berkeley, 2003.

[2] N. Chang and M. Liu. Revisiting the ttl-based controlled flooding search: optimality and randomization. *Mobicom*, 2004.

[3] D. D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, pages 106–114, Stanford, CA, Aug. 1988. ACM.

[4] DTN Research Group. `http://www.dtnrg.org/`.

[5] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *ACM SIGCOMM*, 2003.

[6] K. Fall, W. Hong, and S. Madden. Custody Transfer for Reliable Delivery in Delay Tolerant Networks. Technical Report IRB-TR-03-030, Intel Research Berkeley, 2003.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[8] S. Jain, M. Demmer, R. Patra, and K. Fall. Using redundancy to cope with failures in a delay tolerant network. *Proc. SIGCOMM*, 2005.

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

[10] P. Juang, H. Oki, Y. Wang, M. Margaret, P. Li-Shiuan, and R. Daniel. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS-X*, 2002.

[11] K. Scott and S. Burleigh. Bundle Protocol Specification. *Internet Draft draft-irtf-dtnrg-bundle-spec-05, work in progress*, 2006.

[12] J. Saltzer and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4), 1981.

[13] V. Cerf et al. Delay Tolerant Network Architecture. *Internet Draft draft-irtf-dtnrg-arch-05, work in progress*, 2006.

[14] YACSIM. `http://www.owlnet.rice.edu/~elec428/`.