# Slinky: An Adaptive Protocol for Content Access in Disruption-Tolerant Ad Hoc Networks

Vikas Kawadia, Niky Riga, Jeff Opper
{vkawadia, nriga, jopper@bbn.com}
Raytheon BBN Technologies

Dhananjay Sampath
dsampath@juniper.net
Juniper Networks

## ABSTRACT

The topology of tactical MANETs may vary from well connected to severely disconnected. Applications using the client-server paradigm often do not perform well in such environments, because they rely on the underlying MANET routing protocols to provide connectivity between arbitrary endpoints, which is often infeasible in tactical MANETs. We consider content-based networking as an alternative paradigm to operate tactical MANETs. Content based networking causes information to flow based on its content rather than the identities of endpoints. The network assumes responsibility for positioning information so that it is easily accessible. This model, rather than the classical client–server model, is naturally suited to dynamic and disruption-tolerant tactical MANETs.

We present Slinky, an adaptive protocol for content access in disruption-tolerant ad hoc networks, in particular tactical MANETs. Slinky relies on the community structure inherent in most networks. It consists of a distributed scheme to detect dynamic communities in the network, and a scheme to replicate content across the communities. Slinky does not use any underlying network-wide routing, global topology or geographical information that is often unavailable in tactical MANETs. Slinky is an adaptive and efficient content access protocol that can adapt to a wide range of mobility scenarios. We demonstrate the performance of Slinky in a few tactically relevant scenarios.

## 1. INTRODUCTION

Tactical networks, whether used for military or for disaster relief operations, are required to operate in areas without access to fixed network infrastructure. The network topology can vary from well connected to severely disconnected. Protocols in tactical MANETs need to be tolerant to such disruptions in network connectivity, and cannot rely on end-to-end connectivity for transferring data. Legacy network protocols that rely on end-to-end connectivity clearly do not work well in such tactical MANETs. Disruption tolerant routing protocols [19] that allow for long term storage at intermediate nodes alleviate only part of the problem. This is because they still try to use the network as simple *bit-pipes* – agnostic to the meaning of bits they carry. Applications written over such networks often use the client-server paradigm. Such mode of operation creates severe inefficiencies in tactical MANETs. For example, rare contact opportunities may be wasted trying to transmit multiple copies of essentially the same information but the network has no ability to see that. The fundamental primitive assumed in traditional network design – the ability to communicate with end-points – is hard to provide in tactical MANETs and DTNs because connectivity cannot be guaranteed.

Content based networking is an alternative paradigm [8] which is particularly appropriate for tactical MANETs. The fundamental primitive provided by content networks is the ability to access a piece of content. Information is routed and stored based on its content rather than the identities of endpoints. The network—not the endpoints—manage the topological location and storage of the information. What matters is that a user can get the content and be sure of its validity, the intermediaries are irrelevant. The content is decoupled from its producer and it can be retrieved from any node that has a valid copy. This benefit is particularly noticeable in tactical MANETs and DTNs where the producer is often not even reachable. Content networking also allows efficient usage of network resources. Since every router in the network is content-aware, a set of bytes generally does not have to travel a given link more than once. A content network is in essence a distributed data store, which is an appropriate model for developing novel applications, both tactical and civilian, that can work even in the face of disruptions. Examples include a real-time traffic information system using a network of vehicles, a system to find available parking spots, a network to assist in disaster recovery etc.

In this work we present Slinky, a content networking protocol that is designed to operate in DTNs and tactical MANETs. Slinky exploits the fact that real ad hoc networks often have an inherent community structure, i.e., the network can be divided into groups of nodes such that intra-group connections are much stronger than inter-group connections [13]. This is especially true in tactical MANETs where although the network

is often partitioned, it usually has islands of well connected nodes corresponding to military hierarchy. Our scheme is based on discovering such "community structure" in the network, and placing copies of the content in these communities ensuring high availability of the content to all nodes, as shown in Figure 1. A content generated by the dark node in the lower left is replicated throughout the network so that each community stores a copy. The communities overlap, like rings of a slinky, so that content can spread from one community to the next. Note that the communities reflect the topological structure of the network graph and not the social connections between nodes. Given that the network graph is time-varying, we are interested in communities that reflect the history of connectivity rather than the instantaneous topological snapshot.

Our decentralized dynamic community formation algorithm uses only local topological knowledge, i.e., every node needs to know only the network graph within a small number of hops around it. Global topology knowledge is not required. It is robust to nodes disappearing since there is no node which exclusively holds the community information. The community structure is dynamic and adapts to the network topology as described in Section 3.3. If the network remains static for a long period of time, the number of communities approaches the number of disconnected components. On the other hand, extreme random mobility causes communities to be of very small size.

Another key component of our solution is the placement of content in the communities and responding to queries. Each node in a community stores a fraction of all published content, we use a consistent hash function to assign content to nodes (see Section 3.4.1). Given a key, any node can determine the nodes storing the corresponding content in the community with a local computation. If the community structure changes due to mobility, nodes responsible for storing a content may change and content may need to be re-distributed. The use of a consistent hash function reduces the movement required as explained in Section 3.4.1.

Slinky is a novel content access protocol, especially suited for tactical MANETs. In Section 2, the basic problem is described. Slinky is presented in detail in Section 3. Simulation results provided in Section 4 confirm that Slinky achieves high availability for a low overhead in a range of tactically relevant scenarios.

### 1.1 Related work

Content Distribution Networks (CDNs), such as Akamai, and peer-to-peer systems(P2P), such as Bittorrent, are overlays deployed in the Internet enabling content delivery. CCNX [8] is an effort towards a network that will natively support content. Distributed Hash Tables [1] are an enabling technology for content networks, popular in the Internet. DHTs for ad hoc networks

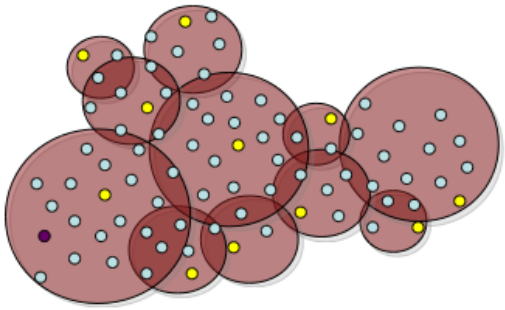


**Figure 1: High level illustration of Slinky functionality showing the discovered community structure and replication of a content item published in the lower left to all communities.**

have been studied previously. Ekta [16] integrates the Pastry DHT and the DSR routing protocol for ad hoc networks while CrossROAD [5] integrates Pastry with the OLSR routing protocol. MADPastry [23] considers physical proximity when integrating Pastry with ad hoc network routing. Virtual ring routing [2] provides both point-to-point routing and DHT functionality. Another set of approaches tackle the problem of object location in ad hoc networks using geographical information, examples of which include [17]. We do not assume that geographical information is available, which is the case if some of the nodes are indoors. GPS systems are also vulnerable to jamming in tactical environments. Most of the approaches summarized above do not explicitly address network disruptions and disconnections, which are common in tactical MANETs.

Several schemes have been proposed for routing in DTNs; see [19] for a survey. Even though content access can be implemented as an overlay on a routing protocol, its performance will probably be worse than schemes explicitly designed for content dissemination. We focus on such schemes and associated community detection schemes relevant to our work. Hui et.al. [7] present algorithms for distributed community detection in DTNs for a range of scenarios. Polat et. al [15] formalize the message ferrying capability as a generalized connected dominating set problem. Chuah et. al [3] study the impact of connector nodes in a DTN pub-sub system. Costa et. al. [4] present a socially aware pub-sub system for DTNs. Data replication protocols in dynamic ad hoc networks such as [9, 12] are also relevant to our work. Slinky does not make any assumptions about query patterns or available content servers, unlike many of the schemes surveyed here. For further details on data replication in ad hoc networks, we refer the reader to the exhaustive survey [6].

## 2. PROBLEM DESCRIPTION

Slinky is a content networking protocol that provides two basic primitives: `publish` and `query`. The `pub-`

`lish(content, key)` operation stores the content at one or more nodes in the network, and the `query(key)` operation fetches it from one of those nodes. For simplicity, we assume that a key uniquely identifies a content. In other words, content and queries are matched by exact comparison of keys. Thus our content network provides a service similar to that provided by a Distributed Hash Table [1]. The problem is in essence one of designing a cost effective mapping (potentially time-varying) that assigns every piece of content to a set of nodes so that it is available to any node that might query for it. A piece of content is considered to be available to a node if it is stored at a node that is currently reachable. In effect, we are assuming that the time to retrieve a content from a node in the same connected component is negligible compared to the latency of retrieving from a node in another component. Availability is obviously maximized by flooding content to all nodes. However, in order to scale to large networks and to be able to handle a lot of content, it is important to use the network resources efficiently. Thus, we are interested in a mapping that reduces the use of network resources, while guaranteeing content availability.

The fundamental problem is to place replicas so as to minimize *access latency*. Versions of this problem have been studied for wireless ad hoc networks by Tang et. al. [21] and Nuggehalli et. al [14] and even for networks without disruptions, the problem is NP-hard. We consider a simplification by focusing on the *disconnected latency*, which we define as the time needed to access a piece of content from a different component by waiting for partitions to heal, at least temporarily. The other component of latency, *connected latency*, is the time to retrieve the content from a node that is in the same component as the querying node. Our approximation is that connected latency is much smaller than the disconnected latency, which is a reasonable assumption in many practical scenarios.

## 2.1 Potential Approaches

We describe some straight-forward approaches for the content placement problem. In a *pure push* scheme content is pro-actively disseminated to all nodes. This is also known as stateful flooding or epidemic routing. Given sufficient contact opportunities between nodes and channel bandwidth, availability is high. However, the overhead in terms of bits transmitted in the network is also high. On the other extreme are *pure pull* schemes, where content is requested directly from the content publisher. This obviously depends on an underlying routing protocol with global topology information and content location knowledge at each node. Thus, the cost of control messages is high as the nodes exchange information to keep their topology and content location information up-to-date. Moreover, this scheme is not robust to network disconnections. *Hybrid push-pull schemes* combines the push-based behavior of epidemic with the pull-based strategy. Here, content from the publisher is replicated onto some subset of other nodes. Then a querying node needs to only discover and contact the closest node which has a replica of the content. Availability and the bits transmitted depend on the strategy used to spread the content during the push step. The Slinky protocol is such a hybrid scheme.

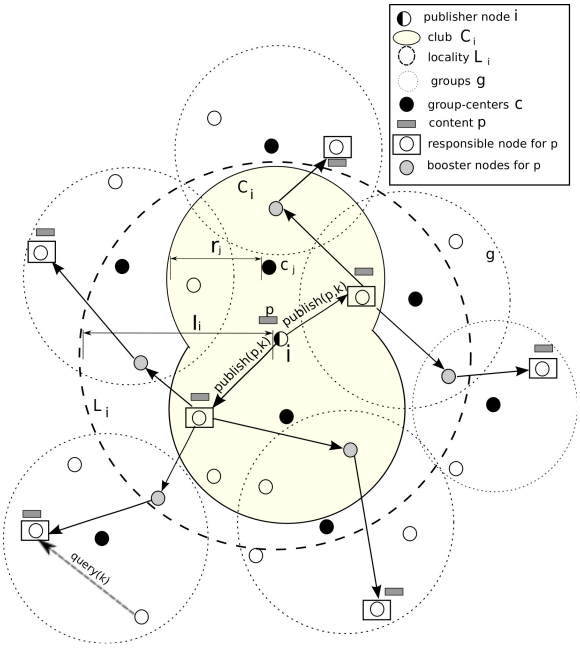



**Figure 2: Overview of Slinky forwarding**

# 3. THE SLINKY PROTOCOL

Slinky has two main components: community detection and content placement in the communities. We first describe our community detection algorithm. In this algorithm nodes form groups and the groups adapt their size based on the underlying community structure. Thus, the groups reflect the long-term community structure of the evolving network and not the structure of the current snapshot only. The group formation algorithm (Section 3.2) runs on a faster timescale than the group adaptation algorithm (Section 3.3). Both protocols rely on a scoped topology discovery mechanism which we describe in Section 3.1. We place a copy of each published content in every group. The copy is assigned to a node within each group using consistent hashing as described in Section 3.4.1. Before we describe our scheme, we introduce some necessary definitions and concepts.

A group $g_j$ is defined by a reference node called the *group-center* ($c_j$), and a group radius, $r_j$. Any node at most $r_j$ hops away from the group-center $c_j$ is defined to be a member of group $g_j$. A group-center announces itself and its group-radius to nodes in its $l_j$-hop neighborhood, where $l_j$ is called the locality radius and $l_j > r_j$. Nodes determine membership in groups by listening to these messages and using their local view of the net-

work topology. We use the term *club* ($C_i$) to denote the set of all the groups that node $i$ belongs to. These definitions (along with others to be introduced later in Section 3.4) are illustrated in Figure 2 which shows the network around node $i$. In the picture, node $i$ belongs to two groups, the union of which is its club, $C_i$. We also show the locality, $L_i$, of node $i$ which is defined as the $l_i$-hop neighborhood of $i$. Notice that node $i$ can see other group-centers in its locality but does not belong to their groups. This is because for such a group-center $c_k$, node $i$'s distance form $c_k$ is more than $r_k$ hops. Figure 2 also shows the publishing mechanics of an item $p$ at node $i$. Consistent hashing is used for content placement inside a group (Section 3.4), which requires every node in a group to know of all other members and a way to reach them. In Slinky, we achieve this by requiring each node $i$ to build and maintain a view of its $l_i$ hop topology, where $l_i$ is big enough to include all members of its club $C_i$. Since two members of a group $g_j$ could be up to $2r_j$ hops away from each other, this implies that node $i$ needs to set its $l_i$ to be at least twice the maximum $r$ of all the groups in its club, i.e., $l_i \geq max(2r_j)$ for each $g_j$ in $C_i$. We set $l_i = max_j(2r_j + 1)$ to enhance content propagation from nodes at the edge of the network.

## 3.1 Locality discovery

As described above, each node needs to construct its $l_i$ hop view of the network topology, called the *locality* $L_i$. Formally, $L_i$ is the induced subgraph of nodes at most $l_i$ hops away from node $i$. Global topology discovery in link state routing protocols in ad hoc networks works by having each node send information about its one-hop neighbors to all nodes in the network. This information is sent in link state advertisements (LSAs) when a link to a neighbor changes. To limit scope and prevent looping, LSAs include a time to live (TTL) value which is decremented at every hop and the LSA is not forwarded after its TTL reaches 0. We can do much better (in terms of overhead) than global topology discovery since each node needs to discover only its locality, however note that the diameter of the locality graph is node dependent. We have designed a topology discovery protocol which enables each node to specify the number of hops of the surrounding network that it needs to discover, which in our case is a node's locality radius, $l_i$. We call our scheme locality discovery. The key insight in locality discovery is that a node $i$ has to send its LSAs far enough so that it reaches all nodes who want to discover node $i$. In other words, a node has to set the TTL value based on the locality radii of other nodes. Thus our LSAs include $l$ along with the TTL. Also, a node has to ensure that its LSAs reach all nodes in its locality graph. A node $i$ sets its $TTL_i$ based on its $l_i$ and received $l_j$s as per the rule given in the UpdateTTL procedure in Algorithm 1, which provides the pseudocode for locality discovery.

**Algorithm 1** Locality Graph Discovery

```
procedure LocalityDiscovery(l_i)
    UpdateLocalTTL()
    if set of 1-hop nbrs or TTL_i changed then
        OriginateLSA(l_i, TTL_i)
    end if
    if LSA received then
        LSA.TTL ← LSA.TTL − 1
        if LSA.TTL > 0 then
            OneHopBroadcast(LSA)
        end if
        UpdateLocalityGraphDatabase(LSA)
    end if
end procedure
procedure UpdateLocalTTL
    TTL_i ← l_i
    for node j in my locality graph do
        if Distance(i, j) is unknown or i = j then
            TTL_i ← max(TTL_i, l_j)
        else if Distance(i, j) ≤ l_j then
            TTL_i ← max(TTL_i, Distance(i, j))
        end if
    end for
end procedure
```

## 3.2 Group formation

The goal of the group formation algorithm is to ensure that every node belongs to at least one group while minimizing the total number of groups in the network since fewer groups means fewer copies in the network. The group-radii are chosen so that each group encompasses a "stable" region. Notions of stability and strategies to choose and adapt $r$ are described in Section 3.3. In this section we assume, that $r$ is given and potentially different for different groups.

A formal definition of this problem is: *Given the set of nodes $V$ in the network, and a radius $r$ per node, compute the minimum set of nodes $D \subseteq V$, so that $\forall v_i \in V$, $\exists d_j \in D$ s.t. distance$(v_i, d_j) \leq r_j$.*

Nodes in set $D$ are the group-centers we defined earlier. Also, notice that in the case where $r$ is the same for all nodes this problem reduces to the *minimum $r$-dominating set* problem, which involves finding the minimum set of nodes such that every other node is at most $r$ hops away from at least one node in the set. The problem is known to be NP-complete. There are solutions [20, 11] for approximate distributed computation of the minimum $r$-dominating set, but most of them require synchronous communication where messages are exchanged in rounds. In a dynamic environment these type of algorithms yield a high message overhead especially when there are frequent topology changes that require dynamic adaptation of the dominating set. Slinky employs a distributed adaptation of the greedy algorithm described below.

### 3.2.1 Greedy algorithm

This is a centralized algorithm where the whole graph is known. This greedy algorithm based on [22] uses the notion of coverage – a node $v_i$ is *covered* if it is a member

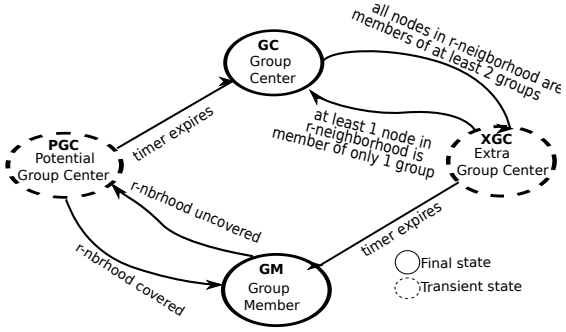



**Figure 3: State transition diagram for group formation**

of at least one group, else the node is *uncovered*. Let $U_{r_i}(v_i)$ be the set of nodes in $V_{r_i}(v_i)$ that are uncovered where $V_{r_i}(v_i)$ is the $r_i$-hop neighborhood of node $v_i$. These are the set of nodes that will be covered if $v_i \in D$.

At every step, the greedy algorithm adds the node that will cover the maximum number of uncovered nodes, to the set $D$. After all nodes are covered, $D$ is not guaranteed to be minimal because of non-uniform radii. As a last step, the algorithm removes nodes from $D$, until the set is minimal.

The greedy algorithm maintains five sets of nodes. $U$: the set of all uncovered nodes in the network. $PGC$: the set of potential group-centers, i.e., nodes with $U_r(v_i) > 0$; $GC$: the set of group-center nodes, it is initially empty. $XGC$ (Extra group-centers): the set of group-centers that do not affect the coverage of other nodes. That is, even if they were not group-centers all the nodes would still be covered. $GM$ (Group Members): the set of nodes that are covered but are not group-centers.

At every step, the greedy algorithm computes all the sets and chooses the node in $PGC$ with the highest value for $|U_{r_i}(v_i)|$ to move it to $GC$. All nodes with $U_{r_i}(v_i) = \emptyset$ are moved to $GM$. It also moves the node with the smallest group from $XGC$ to $GM$. Notice that at most one node can be moved from $XGC$ to $GM$ for $U$ to remain unaffected. The algorithm terminates when both $U$ and $XGC$ are empty. This algorithm will converge after at most $2|V|$ steps, since at each step the set $U$ becomes smaller by at least one element. After $U$ is empty $XGC$ can have at most $|V|$ elements, and in each step $XGC$ also looses at least one element. In the end $GC$ is the desired set $D$.

### 3.2.2 Distributed version of the greedy algorithm

Slinky uses a distributed version of the greedy algorithm described above. In the distributed algorithm, every node makes a decision about its own state as depicted in Figure 3, where the states represent the corresponding sets in the greedy algorithm. The decision is based on states of nodes in its locality graph. In the centralized version, the algorithm is executed in steps, i.e., only one node decides on its group-center status at each step. Also note that the order of choosing group-centers nodes does not matter, even if we were to choose nodes at random the algorithm would converge and result in a minimal $GC$. In order to guarantee convergence in the distributed version, only one node at a time should decide on its state and this decision should be relayed to its neighbors, before the next node makes a decision. In Slinky we probabilistically guarantee this with the use of random timers.

All nodes start in the $PGC$ state and set a timer, $T_{pgc}$, based on the value of $|U_{r_i}(v_i)|$. If the timer expires and $|U_{r_i}(v_i)| > 0$, the node moves to the $GC$ state and sends out an announcement to nodes in its locality. Nodes annotate their local graph with the group-center status of other nodes. Every node uses its view of its $l_i$ hop topology, to compute $|U_{r_i}(v_i)|$ locally. However, the node cannot determine the group membership for all nodes in $V_{r_i}(v_i)$, which might result in overestimating $|U_{r_i}(v_i)|$, which can only lead to increased $|D|$, but does not affect the convergence, or the correctness of the protocol.

The value of the timer, $T_{pgc}$, is inversely proportional to $|U_r(v_i)|$, so that nodes with high $|U_r(v_i)|$ are likely to decide sooner: $T_{pgc} = \frac{\alpha}{|U_{r_i}(vi)|} + rand(0, 2\beta)) * \tau_{r_i}$, where $\alpha$ is a constant that is proportional to the maximum degree in the network, $\beta$ is a random jitter, and $\tau_{r_i}$ is the $r_i$-hop propagation delay. Every node in the $XGC$ state also sets a timer, $T_{xgc}$, where $T_{xgc} = \frac{\alpha}{|V_r(v_i)|} + rand(0, 2\beta)) * \tau_{r_i}$, If the timer expires, the node moves to $GM$ state and sends out an announcement to nodes in its locality. The value of the timer pushes nodes with larger $|V_{r_i}(v_i)|$ to decide sooner. Here we diverge from the greedy algorithm since choosing nodes with higher $|V_{r_i}(v_i)|$ will achieve faster convergence but not necessarily smaller number of groups. The timers are proportional to the propagation delay to probabilistically ensure that the messages will be delivered to the $r$ hop neighborhood before the next timer expires and thus providing statistical guarantees of convergence.

We point out a few properties of the group formation protocol which make it suitable for decentralized operation in DTN environments. The group-centers do not keep track of their members, rather the group membership is implicitly determined by nodes listening to announcements from group-centers. There are no explicit join/leave messages. Group-centers are reference nodes to define a group, and do not keep any special state.

The group formation algorithm does not cause network wide adjustments or ripples in node states in the case of a local change. To see this, note that when the network has stabilized all nodes are in one of the final states ($GC$ or $GM$), Figure 3. If a topology change happens, some nodes might transition to one of the temporary states ($XGC$ or $PGC$). A node in the $XGC$ state

will either go back to $GC$ which does not cause a new announcement, or it will go to the $GM$ state and send out an announcement. The $XGC \rightarrow GM$ transition does not affect the coverage of any node. Thus, the only transition the announcement can cause at other nodes is $XGC \rightarrow GC$ which does not create new announcements and hence no state changes. Hence, nodes that are in $XGC$ state cannot cause a ripple effect. On the other hand, a node in the $PGC$ state will either move back to the $GM$ state which does not cause an announcement or it will move to the $GC$ state and send out an announcement. Then nodes in $GM$ and $XGC$ are not affected by the new group-center. Nodes in $PGC$, if affected, will go back to GM without sending an announcement. Nodes in $GC$ might move to $XGC$ but we have already proven that nodes in $XGC$ won't cause a ripple effect. Hence, there are no network wide adjustments in state due to a topology change.

## 3.3 Group adaptation

In this section, we present the algorithm used by group centers to dynamically decide their group radius, $r$. There are various trade-offs to consider since different group sizes provide different benefits. Increasing $r$ reduces the replication in the network while increasing the topology maintenance cost as the nodes have to maintain their locality graph for larger $l$. Although the topology maintenance cost is constant per node if the network is static, it increases sharply with mobility. Also if there is high level of inter group mobility, i.e., nodes moving in and out of groups, the overhead of maintaining a copy of each content per group increases (Section 3.4.3). Thus, the benefits attained from higher values of $r$ are quickly outweighed by the cost of maintaining a larger topology and more dynamic groups. These observations lead to an adaptation scheme that is based on the change in the locality of a group-center.

Every node measures the stability of it's $m$-hop neighborhood, for each $m$ between 0 and $l$. This computation is based only on the node's locality graph. Let $\sigma_t(m)$ be a function that estimates the stability of a nodes' $m$-hop neighborhood at time $t$. Let $N_t(m)$ be the set of nodes in the $m$-hop neighborhood at time $t$. The stability is computed as follows:

$$\sigma_t(m) = \alpha J(N_t(m), N_{t-1}(m)) + (1-\alpha)\sigma_{t-1}(m), \quad \text{where } J(A,B) = \frac{A \cap B}{A \cup B} \quad (1)$$

The function $J(A, B)$, called the Jaccard Index [18], is a measure of the similarity between two sets. We thus maintain an exponentially weighted moving average of the similarity of the nodes in the locality over time. Each node then adjusts its group radius to the maximum value of $m$ for which the stability is above a given threshold. To dampen oscillations in group radius adaptation, we use two thresholds, $w_{high}$ and $w_{low}$, where $w_{high} > w_{low}$. The threshold $w_{high}$ is used when increasing the value of $r$, while $w_{low}$ is used when decreasing the values of $r$. The use of thresholds reduces frequent changes in the group radii.

Group adaptation occurs at a much slower timescale than group formation. This timescale separation allows group formation to converge between group changes. The stability metric also provides feedback to the group formation algorithm to converge faster or defer changes when the topology is unstable. It does so by providing a sensible initial value of group radius $r$ for nodes that become group-centers. The stability is also considered when a group-center is deciding whether to stop being one. If the locality is rapidly changing then it is sensible for the group to maintain its structure until things settle down. Group adaptation relieves the network designer of the tricky problem of figuring out a good value of $r$ for all nodes in the network at all times.

## 3.4 Content placement and querying

In this section, we describe how content is placed in the groups and how the queries are performed.

### *3.4.1 Content placement and forwarding*

A straightforward scheme for content placement is to designate a node as the content-server for the group which stores all content. However, the content-server is a critical point of failure and also a bottleneck for communication and storage. Instead, we distribute the content storage over all nodes in a group by mapping content to nodes using consistent hashing [10]. Consistent hashing has the important property that addition or removal of a bucket — each node is a bucket in this case — does not significantly change the mapping of keys to buckets. Thus, the use of consistent hashing helps reduce the redistribution of content due to mobility. Consistent hashing also provides load balancing as the distribution of content over nodes approaches uniform as the amount of content stored increases.

Consistent hashing within a group is similar to that of overlay DHT systems like Chord [1]. Essentially, both keys and node identifiers are mapped into a common virtual space and the node whose identifier is closest to the key in this space is responsible for storing the corresponding content. In most overlay DHTs, the responsible node is reached by routing in the virtual space; each node forwards the request for a key to a node whose identifier is closest to the key. The problem is simpler in Slinky since each node can compute the responsible node locally and reach it via the shortest path based on the locality graph. When a content is published at a node, it forwards it for storage to the responsible nodes in the groups it belongs to. However, in order to maximize availability the content needs to reach the responsible nodes in all groups in the network. Since

a node may have only a partial view of the network, it is probably unable to determine the responsible nodes in groups outside its club. In our scheme, responsible nodes not only store the content, but they also forward it to a few nodes in their locality. These nodes, called *booster nodes*, forward only to responsible nodes in their groups. As seen in Figure 2, when not $i$ publishes a piece of content $p$, booster nodes help in propagating $p$ to all responsible nodes in the network.

The pseudocode for content forwarding is given in Algorithm 2.

**Algorithm 2** Content Forwarding and Query Processing

```
procedure ProcessContent(c)
    if AmResponsibleFor(c) then
        if c is not already in my Store then
            Store(c)
            Send(c, GetBoosterNodeList)
            Send(c, GetSourceOfMatchingQueries(c))
        end if
    else
        Send(c, GetResponsibleNodes(c))
    end if
end procedure
procedure ProcessQuery(q)
    if AmResponsibleFor(q) then
        Send(GetMatchingContent(q), Source(q))
    else
        Send(q, GetResponsibleNodes(q))
    end if
end procedure
```

A responsible node selects booster nodes from its locality in such a way that the content is likely to spread in all directions. A heuristic which achieves this is choosing an $\frac{l}{2}$ dominating set from the nodes in the locality which are at least $\frac{l}{2}$ hops away from the responsible node as booster nodes. This ensures that booster nodes are spread apart and are in the "outer" part of the locality to be able to see more neighboring groups. A responsible node forwards the content to booster nodes only if it is not already storing it. The spreading terminates once every node responsible gets the content. We formalize the argument in Lemma 1.

Lemma 1. *Slinky content forwarding is free of infinite loops.*

Proof. In stateful flooding, nodes store an item and forward it to their neighbors only when they first receive it. They also synchronize their stores with new neighbors as the topology changes. Thus any item goes on any edge at most once. The storage provides the obvious loop-freedom.

Now consider the case when only a subset of nodes have storage. Denote by $S$ the set of nodes that have storage and by $\overline{S}$ the set of nodes that do not have storage. Modify the forwarding rule so that a node in set $\overline{S}$ can forward messages only to nodes in $S$. Only nodes in $S$ synchronize content among themselves when there is mobility. Since nodes in $\overline{S}$ are allowed only to forward to nodes in $S$, the nodes in $\overline{S}$ can be considered as edges from the perspective of forwarding. Thus each content is transmitted at most once between nodes in $S$ and the modified strategy is free of infinite loops.

In the case of Slinky, the sets $S$ and $\overline{S}$ are dependent on the content. The responsible nodes for a content form the set $S$ and the booster nodes are the nodes without storage, i.e., $\overline{S}$. Thus, forwarding for any given content is free of infinite loops. □

### 3.4.2 Querying

We now describe how the stored content is retrieved. Content is only retrieved from the local club. Since each group uses consistent hashing this simply involves sending the query to the nodes responsible for that content. The responsible node stores the query until it gets the content or it discovers that the source of the query has moved away. The source also needs to store and regenerate the query if it becomes a member of new groups as the topology changes. The pseudocode for querying is give in Algorithm 2. Since queries are only forwarded to responsible nodes within the club and are not propagated network wide, there is no issue of them getting into infinite loops.

**Algorithm 3** Adjusting to Topology changes

```
procedure TopoAdjustment(i)
    if a node j joins group g in C_i then
        SynchronizeStore(j, g)
    end if
    if a node j leaves group g in C_i then
        Delete(queries sourced by j)
    end if
    if a group-center c_k appears in L_i then
        if i belongs g_k then
            SynchronizeStore(members of g_k, g_k)
            ProcessQuery(queries sourced by i)
        else
            ProcessContent(all stored content)
        end if
    end if
    if node i leaves group g then
        SynchronizeStore(members of g, g)
    end if
end procedure
procedure SynchronizeStore(nodelist, g)
    for n in nodelist and all stored content c do
        if IsResponsibleForInGroup(n, c, g) then
            Synchronize(c, n)
        end if
    end for
end procedure
```

### 3.4.3 Adjusting to topology changes

Topology changes may cause the responsible nodes in a group to change necessitating movement of some content and queries. Nodes are often able to exchange content with new responsible nodes since they might still be in the locality which is larger than a node's club. Algorithm 3 provides the pseudocode for content and query exchanges that may be required. In our scheme, content is offered only by nodes who are no longer responsible for it. Note that due to the use of consistent

hashing, most of the content that a node is not responsible for after a topology change is reassigned to just two nodes: its prior neighbors in the virtual space. Nodes exchange summary vectors before actually exchanging the content to save transmissions if the target node already has the content. As an optimization, when storage is available nodes store content even if they are not responsible for it. Regarding queries, only the source forwards queries to responsible nodes when it joins a new group. The adjustment procedure described in this section tries to ensure that all responsible nodes have the content they should even as the topology changes. We emphasize that exchanges needed are limited due to the use of consistent hashing as a consistent hash function changes minimally upon changes in its range [10].

### 3.5 Heterogeneous networks

A group is a well-connected set of nodes where the intra-group communication cost is low. In heterogeneous networks however, where there might be high-delay or low data-rate links, mere connectivity is insufficient to identify the best groups. Examples include tactical networks with satellite links, or with links formed via expensive high power radios. To respect these operational constraints, frequent use of these links should be avoided. Slinky achieves this by excluding such links from the group formation and adaptation algorithms, ensuring that they are not used for intra-group communication while still using them for content forwarding. Section 4.1 shows how Slinky efficiently uses these links.

## 4. EVALUATION

We evaluate Slinky using a custom discrete-event, packet-level simulator. The network is modeled as a time varying graph. We model network details like link data-rates, medium access contention, radio propagation models etc. by link delays and loss rates. The propagation delay of links is set to $0.05s$ with a jitter of 0.001. Content and queries are generated randomly on the nodes. A content is 100 KB on average, whereas all other messages are 1 KB. We do not implement all the details of the locality discovery for simplicity. We assume that the timescale of local topology discovery is much faster than the timescale that the network changes and estimate the cost of topology discovery.

**Table 1: Description of evaluation scenarios**

| Parameters | Lakehurst | UAV |
|---|---|---|
| Nodes | 40 | 101 |
| Duration (secs) | 10200 | 3000 |
| Trace type | Real | Synth. |
| Content items published | 1000 | 1000 |
| Number of queries | 3000 | 3000 |
| Avg. content generation interval (s) | 3 | 2 |
| Avg. query generation interval (s) | 1.5 | 0.5 |
| Content generation start time | 100 | 100 |
| Query generation start time | 100 | 100 |

We compare Slinky with Epidemic and Single-copy schemes on diverse scenarios of tactical importance. Epidemic is a pure-push scheme that tries to place a copy of all content to each node; similar to Slinky with group radius of 0. Single-copy stores one copy of each content in every connected component; similar to Slinky but with group radius fixed to the visible network diameter.

### 4.1 Lakehurst scenario

This is a tactical scenario with heterogeneous links based on real mobility traces. The traces were collected during experiments conducted for the DARPA Control-Based MANET program. The data set represents 40 nodes organized into three company teams and is mapped to terrain areas in Lakehurst, NJ.

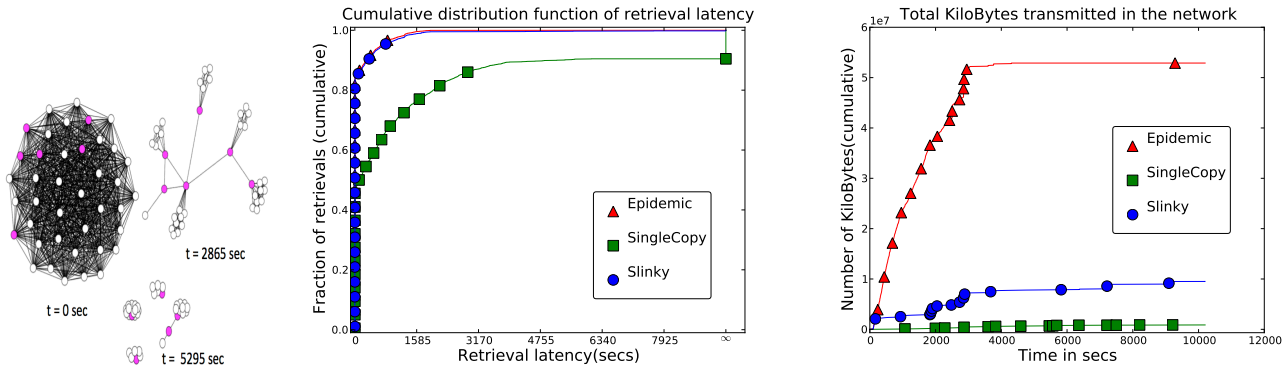
In the scenario the company teams maneuver from a rally point to an objective using two primary routes over a three-hour period. There are six *leader* nodes which have high range radios and can communicate with other leaders, as shown in Figure 4a. The leaders are colored darker. The links between the leaders have a delay which is 1000 times the delay of a regular link. This heterogeneous network has some *long-links*, which Slinky can handle as described in Section 3.5. The nodes start out as a well connected component and over the course of the experiment disburse into teams, often disconnected from each other, and re-gather at the end of the experiment. We add random content and query generation on to the real mobility trace. The parameters chosen are listed in Table 1.

Figure 4b shows the cumulative distribution function of the retrieval latency for the three algorithms. Retrieval latency is defined as the time needed to get a published content to a node who has generated a query for that content. Slinky does almost as well as Epidemic, retrieving almost 80% of the content almost instantly. Single-copy takes much longer as it may need to access the content across the long links. Figure 4c shows that the overhead of Slinky is far less than that of Epidemic. Slinky is able to achieve the latency of Epidemic at an overhead only slightly larger than that of Single-copy.

### 4.2 UAV scenario

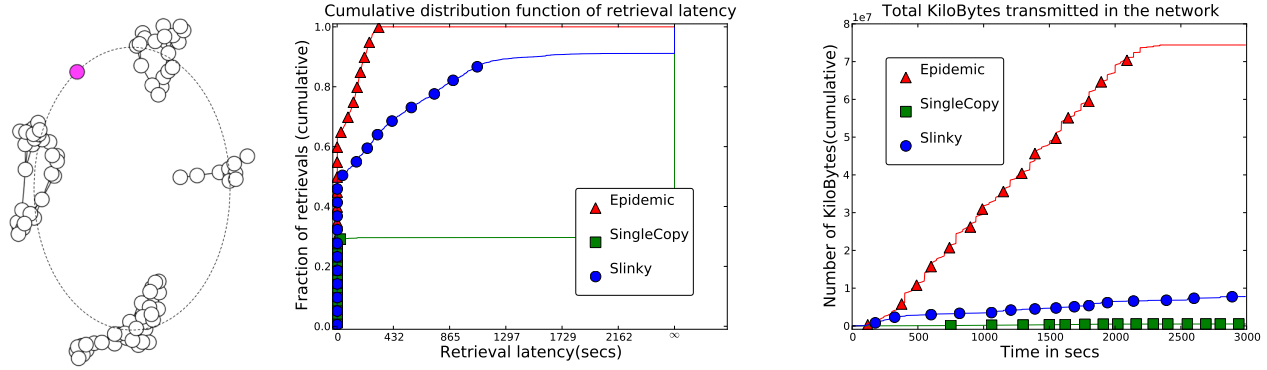
Next we evaluate the performance of Slinky on another common tactical scenario where an unmanned aerial vehicle (UAV) is flying between groups of soldiers which are otherwise disconnected, as shown in Figure 6a. The UAV acts as a data-mule carrying data between the components. There are four components of sizes 10, 20, 30 and 40 nodes respectively. We used synthetic traces for this scenario as real traces are unavailable. The parameters chosen are listed in Table 1.

The performance of Slinky's group adaptation scheme is illustrated by time evolution of the stability metric




(a) (b) (c)

**Figure 4: Performance comparison of Slinky, Epidemic and Single-copy for the Lakehurst scenario.**




(a) (b) (c)

**Figure 6: Performance comparison of Slinky, Epidemic and Single-copy for the UAV scenario.**

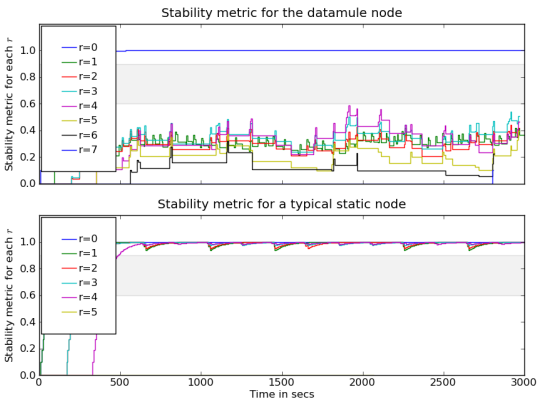



**Figure 5: Stability metric for the UAV scenario.**

for the data-mule node and for a typical static node as shown in Figure 5. The grey band depicts the thresholds, $w_{high}$ and $w_{low}$, described in section 3.3, and there is a line for the stability metric for each value of $r, 0 \leq r \leq l$, where $l$ is the locality radius. Note that for $r = 0$, the stability metric is always 1 as the set of nodes is the node itself. For the static nodes, the stability metric remains above the high threshold for all $r$ whereas for the data-mule it is much lower, hence its group-radius stays 0 and it remains a group by itself.

We compare the retrieval latency and overhead of the three schemes. Epidemic incurs huge overhead as shown in Figure 6c, which would be problematic in bandwidth-limited scenarios. Single-copy, on the other hand, does not propagate the data between components and manages to retrieve only about 30% of the queried content as shown in Figure 6b. Slinky gets the best of both worlds by satisfying almost 90% of the queries while keeping the overhead almost as low as Single-Copy; Figure 6c. Slinky's capability to adapt makes it useful in diverse scenarios of practical utility.

## 5. CONCLUSIONS

The content networking model allows one to be robust to disruptions in network connectivity because of the decoupling it provides between producers and consumers of information. We have presented, Slinky, a scalable protocol for content access in tactical MANETs and disruption-tolerant ad hoc networks. The key feature of the protocol is its ability to adapt to disruptions and topology changes in the network. As we have demonstrated, adaptation enables Slinky to run efficiently on diverse network scenarios, without any prior configuration. Slinky has good scalability properties since it relies only on local topology information and localized

changes in topology do not cause network wide ripples. The core content networking layer that we provide enables powerful new applications for tactical MANETs and other mobile ad hoc networks that need to be disruption tolerant.

## Acknowledgements

This work was sponsored in part by DARPA 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.

Research was also sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. 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 Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

## 6. REFERENCES

[1] H. Balakrishnan, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.

[2] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by dhts. In *SIGCOMM*, pages 351–362, 2006.

[3] M. Chuah and A. Coman. Identifying connectors and communities: Understanding their impacts on the performance of a dtn publish/subscribe system. In *CSE '09*, pages 1093–1098. IEEE Computer Society, 2009.

[4] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco. Socially-aware Routing for Publish-Subscribe in Delay-tolerant Mobile Ad Hoc Networks. *IEEE Journal On Selected Areas In Communications (JSAC)*, 26(5):748–760, 2008.

[5] F. Delmastro. From pastry to crossroad: Cross-layer ring overlay for ad hoc networks. *IEEE PerCom*, pages 60–64, 2005.

[6] A. Derhab and N. Badache. Data replication protocols for mobile ad-hoc networks: a survey and taxonomy. *IEEE Communications Surveys & Tutorials*, 11(2):33–51, 2009.

[7] P. Hui, E. Yoneki, S. Y. Chan, and J. Crowcroft. Distributed community detection in delay tolerant networks. In *MobiArch*, pages 1–8, 2007.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT*, pages 1–12, 2009.

[9] Z. Jing, W. Yijie, L. Xicheng, and Y. Kan. A dynamic adaptive replica allocation algorithm in mobile ad hoc networks. In *IEEE PerCom*, pages 65–69, 2004.

[10] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, pages 654–663, 1997.

[11] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *PODC*, pages 25–32, 2003.

[12] C. A. La, P. Michiardi, C. Claudio, C. Carla-Fabiana, and M. Fiore. A lightweight distributed solution to content replication in mobile networks. In *IEEE WCNC 2010, Sydney, Australia*, 2010.

[13] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[14] P. Nuggehalli, V. Srinivasan, and C. Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *MobiHoc*, page 34, 2003.

[15] B. Polat, P. Sachdeva, M. Ammar, and E. Zegura. Message ferries as generalized dominating sets in intermittently connected mobile networks. In *MobiOpp*, 2010.

[16] H. Pucha, S. Das, and Y. Hu. Ekta: an efficient dht substrate for distributed applications in mobile ad hoc networks. *Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 163–173, 2004.

[17] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *Proc. of the 1st ACM Intl. workshop on Wireless sensor networks and applications*, pages 78–87, 2002.

[18] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-neighbor methods in learning and vision: theory and practice*. MIT Press, 2005.

[19] J. Shen, S. Moh, and I. Chung. Routing Protocols in Delay Tolerant Networks: A Comparative Survey. In *ITC-CSCC*, pages 1577–1580, 2008.

[20] M. Spohn. *Using dominating sets to improve the performance of mobile ad hoc networks*. PhD thesis, UC Santa Cruz, 2005.

[21] B. Tang, H. Gupta, and S. Das. Benefit-based data caching in ad hoc networks. *IEEE Trans. on mobile computing*, 7(3):289–304, 2008.

[22] V. V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[23] T. Zahn and J. Schiller. DHT-based unicast for mobile ad hoc networks. *IEEE PerCom*, 2006.