Abstract—In this paper we introduce a topology control algorithm for increasing transport robustness and efficiency by creating two simultaneous communication paths between the source and destination nodes of a data flow. We make use of a cross-layer substrate [1], [2] that allows us to detect the flows of data in the network while also allowing for a finer control of the routing. Because information of flow traffic in the network is available, the algorithm attempts to achieve intra-flow interference reduction by exploiting node mobility.

I. INTRODUCTION

The issue of routing in mobile ad-hoc networks (MANETs) has been of interest to the research community in recent years. Several routing protocols that try to maximize network connectivity have been proposed[3], [4], [5]. However, some MANETs might have other concerns such as load balancing, fault-tolerance and/or improvement in bandwidth on top of node connectivity, and multi-path routing is one of the preferred mechanisms for addressing these issues. In this paper we present dpath, a distributed topology control algorithm designed to create, on demand, two simultaneous data communication paths between the source and destination nodes of a data flow. The algorithm creates the interference free data paths by moving nodes for the duration of the data flow. Assumming a common and fixed transmission frequency for all nodes, the algorithm adjusts the position of the nodes to create two data paths that are both link and edge disjoint, thus interference free under certain simplifying assumptions1. Given enough resources, the only two points of contention between the data paths are the source and destination nodes.

Once the disjointed path has been created, the sender node may choose to utilize both paths to improve robustness by duplicating the transmission of each data packet over the paths. Alternatively, the sender node may choose to improve throughput instead by interleaving the transmission over each link. Figure 1 shows an instance of a network with disjointed paths between nodes ‘0’ and ‘11’.

1This is strictly true for a unit disc topology. In most practical cases, however, the transmission interference range is significantly higher than the connectivity range (defined by the RSSI threshold) which would technically require the algorithm to be defined over two topologies, one representing the network connectivity and one for interference (a contention graph). Nevertheless, as currently proposed, the algorithm relies on the simplifying assumption that network propagation can be approximated to a unit-disc, but we expect that it will well tolerate small violations of the assumptions.

Fig. 1. Dual disjoint path in place, allowing sender to choose between end-to-end robustness and throughput.

II. RELATED WORK

Popular choices of multi-path routing protocols include SMR (Split Multipath Routing) [6] and AODVM (Ad hoc On-Demand Distance Vector Multipath) [7]. Both of these protocols are on-demand (reactive) routing protocols. These modified versions of DSR[3] and AODV[4], respectively, have been adapted to handle multiple node-disjoint paths. In these algorithms, the selection of the disjointed path is realized at the destination node. Such decision is based on global information gathered from and provided by route-request messages.

These algorithms, however, are bound by a common set of assumptions. First, the algorithms consider that the topology of the nodes is fixed, most likely because neither the mobility of the nodes or the transmission power of the devices can be controlled. Second, the mechanisms of route creation and selection do not take into consideration historical information of data traffic.

Improvement of bandwidth is an issue that has been attempted in [8], where the authors make use of a cross-layer approach. However, the authors try to solve the multipath routing problem using genetic algorithms, a solution that might be too complex to be performed in real-time.
III. THE DUAL-PATH TOPOLOGY CONTROL ALGORITHM

The dpath algorithm is a reactive algorithm, triggered by the presence of a data flow in the network. The algorithm is incremental in nature, in the sense that changes in topology are progressively reinforced by each packet in the data flows. The approach allows for a solution that evolves with other uncorrelated changes in the network topology or data flow. There are no requirements for the duration of the data-flow (either in number of packets or time) for triggering the protocol.

While in its current implementation the topology adaptation rate is fixed (i.e. the rate in which the topology adapts to an existing data flow is fixed and independent of the rate of the flow), a variation of the protocol can be easily conceived to increase the adaptation rate as a function of the data rate, forcing the algorithm to react differently to different types of flows.

A. Data Flows

In the context of this work, a data flow is identified by a pair \((n_s, n_d)\), where \(n_s\) is the source node, and \(n_d\) is the destination node of unicast data packets. At any given time, multiple data flows may co-exist in the network and, under the assumption that all transmissions occur over the same frequency (no spectrum allocation), to minimize interference, a given node should not participate in more than one of the paths of any given data flow. The goal of the algorithm is to separate each data flow into two disjoint data paths between source and destination.

For instance in Figure 1 there is a single data flow between nodes ‘0’ and ‘11’ going through two non-interfering data paths (represented by the lines with darker color). The current version of the algorithm defines a flow based solely on source and destination addresses, disregarding traffic type and port numbers. Multiple independent data flows between a given source and destination (for instance between different applications) will be reduced to the same flow, as they share the same source and destination IP addresses. This simplification is acceptable because a dual communication path between each node is valid regardless of the associated port numbers (or applications) involved.

The choice of constraining the protocol to unicast traffic is to ensure that no adaptation takes place over broadcast messages, which are commonly used for network management and routing protocols.

B. Node State

Every node in the network maintains a list of the data flows in which it participates (that is, the node is either the source, destination, or a forwarding node of the data packets of a flow). In addition to their identifiers (source and destination node addresses), the flows are annotated with the immediate last-hop node from which it was received \((n_p)\), the set of nodes used as immediate next-hop for that flow \((N_f)\), and a binary flag \((s)\) indicating if the data flow has been split (i.e. forwarded to more than one next-hop neighbor) by any of its upstream nodes, including the source node.

\(F_n\) constitutes the set of metadata pieces associated with all the flows going through a node. Each node creates and maintains the metadata based on information relayed with the data packets. \(F_n\) is defined as a set of tuples of type \((n_s, n_d, n_p, N_f, s)\).

C. Working Principle

The working principle of the protocol is simple. There is an algorithm that regulates the mobility of the nodes, that is, the local mobility heuristics used to create the disjointed paths. In parallel to the mobility task, there is another algorithm that makes the packet forwarding decision. Both algorithms work independently from one another but based on the same information (the flow of data packets) as feedback. This leads to a convergent solution for both mobility and packet forwarding.

The list of flows \((F_n)\) that each node \(n\) is currently handling is periodically broadcasted to all its immediate neighbors so that every node also maintains the list of flows being handled by its neighbors \((N)\). When receiving a data packet for forwarding, each node will either split the flow (if it hasn’t been done yet by one of the upstream nodes) or attempt to move away from other nodes carrying the same flow on the other path.

The proposed approach is independent from the routing protocol being used. However, dpath does require the following three capabilities from the routing service for its operation:

- **Enhanced Next-Hop Lookup:** That is, the possibility of looking up a set of best next-hop nodes for a given destination, as opposed to a single best-hop which is the conventional mode. Such information is often available in most routing protocols (both link-state and distance-vector based), but generally discarded and not included in the routing table. Minor modifications on the routing table (and possibly on the protocols themselves) to retain this information is necessary for the dual-path algorithm.

- **Packet Forward Override:** This is necessary, for instance, for the splitting of a data flow, where a copy of the data packet (or interleave packets) are forwarded to both the best next-hop and the second best next-hop of a given destination.

- **Data Packet Modification/Encapsulation:** To reduce overhead, the state-information that is shared by dpath is minimum and is appended as part of the data packets themselves. For that, dpath requires the capacity of annotating data packets with a flag indicating whether the flow has been split, together with the list of next hops, to which the packet will be sent. Therefore, the routing service must provide a facility for transmitting the annotated data packets.

In most cases, however, such capabilities can be easily implemented in a non-intrusive way that is fully compatible with the default protocol.
IV. ALGORITHM DESCRIPTION

The packet forwarding and mobility algorithms work in parallel, based on the same information being exchanged through the data packets.

In this section we introduce the pseudo-code for both the packet forwarding and mobility algorithms, together with a less formal description of each phase of the algorithms and the mobility heuristics.

A. Packet Forwarding

Algorithm 1 shows the logic for packet-forwarding in the dpath. The algorithm takes the following arguments:

- \( P_{n_s,n_d} \), the packet to be forwarded, labeled with source node \( (n_s) \) and destination node \( (n_d) \).
- \( n_p \), the previous hop of the packet.
- \( s \), the flag indicating whether the flow has already been split.

The algorithm also relies on a modified version of the \( \text{getNextHop}(\cdot) \) function. The modified version, \( \text{getNextHop}(n_d,E) \), will receive the target node \( (n_s) \), together with the list of nodes \( E \) that will be excluded from the calculation of the path. For instance, if \( E \) consists of the next best hop for a given destination address, the function \( \text{getNextHop}(n_d,E) \) will return the second best hop to that destination address.

```
Input: \( P_{n_s,n_d}, n_p, s \)
if \( s \) then
    \( n_1 \leftarrow \text{getNextHop}(n_d,\emptyset) \);
    \( E \leftarrow \emptyset \);
    \( \text{while } n_1 \neq \text{null} \)
    \( \text{and } \exists (n_s, n_d, n'_p, N'_f, s') \in F_{n_1} [n'_p \neq \text{this] do} \)
    \( E \leftarrow E \cup \{n_1\} \);
    \( n_1 \leftarrow \text{getNextHop}(n_d, E) \);
else
    \( n_1 \leftarrow \text{getNextHop}(n_d, \emptyset) \);
    \( n_2 \leftarrow \text{getNextHop}(n_d, \{n_1\}) \);
end
\( N_f \leftarrow \{n_1, n_2\} \);
if \( |N_f| > 1 \) then
    \( s \leftarrow \text{true} \);
end
if \( |N_f| > 0 \) then
    \( \text{foreach } n \in N_f \) do
        \( \text{forward } P_{n_s,n_d} \) to \( n \);
    end
else
    throw NoRouteHostException;
end
update \( F_{\text{this}} \) with \( \langle n_s, n_d, n_p, N_f, s \rangle \);
```

Algorithm 1: Packet Forwarding Logic.

The primary goal of the packet forwarding algorithm is to handle the splitting of the flow. The algorithm also maintains the flow separation by favoring (when possible) a next-hop that has not reported any participation in the current flow.

B. Mobility Algorithm

Let us define a data flow as tuple \( \langle n_s, n_d, n_p, N_f, s \rangle \). The goal of the mobility algorithm is to “observe” the data flow and move the local node to create the disjointed path. Mobility decisions are local to each node, and are based on traffic and on information reported by other nodes. There is no global state required for node mobility, and information is shared only between immediate neighbors. Nodes are location aware (e.g. via GPS, or triangulation), and the algorithm uses a node’s position and that of its neighbors to decide where to move. Currently, the mobility algorithm only works for a single flow, and this flow is identified by the tuple \( \langle n_s, n_d, n_p, N_f, s \rangle \). Algorithm 2 shows the logic for local node mobility.

The arguments of algorithm are the following:

- \( n_s \), the source node of the flow.
- \( n_d \), the destination node of the flow.
- \( n_p \), the previous hop node.
- \( N_f \), the set of next hop nodes.
- \( \text{minrssi} \), minimum acceptable RSSI.
- \( N \), the set of 1-hop neighbors
- \( M \), distance to move the node on each step.

```
Fig. 2. Example Simulation Topology for the dual-path algorithms.
```

Figure 2 shows the initial topology for an NS-2 simulation of the dual-path algorithm in a 12 node network. In the simulation, a data flow starts in node ‘0’, with node ‘11’ as the destination. Our implementation does not use a modified routing protocol. Instead, it captures the data packets at the transport level. Then, it will split the flow (i.e. duplicate the packet) if more than one next hop to the destination node is available, and if the flow has not been previously split. This will send the packet through both the first and second best next hops.

As illustrated in Figure 3, the flow takes advantage of disjointed paths already in place in the topology. The problem, however, is that without topology adaptation, the redundant data paths interfere with each other, thus potentially reducing
As opposed to improving the robustness of the flow. Furthermore, there are two points in the flow that rely on a single node. The flow will be degraded (or fail to make it through) if nodes ‘2’ or ‘3’ are removed from this topology or fail to perform well.

The dual-path topology adaptation algorithm addresses this issue by allowing intermediate nodes to move autonomously (in loosely coupled coordination) to create a path from source to destination that is non-interfering (that is, node and edge disjoint, with no direct edges between intermediate nodes involved in different paths of the flow). Each node in the path periodically broadcasts its current position and information about the data flows that it is handling. Based on the broadcasted information, each node can independently identify potential interferences and decide how to move.

Figure 4 illustrates the adaptation process. The data flow from node ‘0’ to node ‘11’ is relayed by nodes ‘4’ and ‘1’. These nodes determine that they are both interfering with each other in regards to the same data flow. Furthermore, they also identify that data is being relayed to them from the source in a non-split flow. This is an indication that node ‘2’, the immediate upstream node, is either a common vertex in the

```plaintext
Input: n_s, n_d, n_p, N_f, min_rssi, N, M
dx ← 0;
dy ← 0;
x ← getX (this);
y ← getY (this);
if n_p ≠ n_s ∧ ∃(n_s, n_d, n_p', N_f', s') ∈ F_n_p[|N_f'| > 1]
then
    | dx ← getX (n_p') - x;
    | dy ← getY (n_p') - y;
else if rssi (n_p) < min_rssi then
    | dx ← getX (n_p) - x;
    | dy ← getY (n_p) - y;
else
    n_farthest ← n_p;
    foreach n' ∈ N_f do
        | if distance (this, n_farthest) ≤ distance (this, n') then
            | n_farthest ← n';
    end
end

tx ← getY (n_farthest) - getY (this);
ty ← getX (this) - getX (n_farthest);
d ← √(tx^2 + ty^2);

if ∃n ∈ N[∃(n_s, n_d, n_p'', N_f'', s'') ∈ F_this[n ≠ n_p''] ∧ ∃(n_s, n_d, n_p'', N_f'', s'') ∈ F_n[ this ≠ n_p'']] then
    moveTo (x_n ← getX (n);
    y_n ← getY (n);
    x_n' ← x_n + M ∙ tx;
    y_n' ← y_n + M ∙ ty;
    m'' ← √((x - x_n')^2 + (y - y_n')^2);
    m ← √((x - x_n)^2 + (y - y_n)^2);
    if m'' < m then
        | dx ← -tx;
        | dy ← -ty;
    else
        | dx ← tx;
        | dy ← ty;
    end
end

if dx ≠ 0 ∨ dy ≠ 0 then
    d ← √(dx^2 + dy^2);
dx ← dx / d;
dy ← dy / d;
x ← x + M ∙ dx;
y ← y + M ∙ dy;
moveTo x, y;
end

Algorithm 2: The node mobility algorithm.
```

Fig. 3. Dual-path simulation (phase 1), data flow from nodes 0 to 11.

Fig. 4. Mobility vectors on some of the intermediate nodes.
path, or that there is a common edge leading to node ‘2’. Either way, the heuristic for node mobility in this case is to move away from interfering intermediate nodes, and towards upstream nodes that are unable to sub-divide the stream.

The composed mobility vectors are shown in Figure 4. Nodes ‘1’ and ‘4’ simultaneously move away from each other (to reduce interference) and towards their upstream neighbor (to create the conditions for a split from the source). The resulting direction is the composed vector illustrated in Figure 4. The other nodes involved in the path (with the exception of the source and target nodes) are also moving, but their mobility vectors is omitted here for simplicity.

When node ‘0’ finds two immediate paths to the target nodes, it immediately splits the flow, leading to the configuration shown in Figure 5. At this point, nodes ‘1’ and ‘4’ no longer need to move towards the source as they are each receiving a split of the flow. They are, however, still interfering with one another so they continue to move, but with in the directions illustrated in Figure 6.

At the end of the process, the flow from node ‘0’ to ‘11’ is fully split at the source node, following separate, non-interfering paths, to later converge only at the target node (Figure 7), which is a stable state for the algorithm. At this point all intermediate nodes stop moving and data continues to flow.

Although all nodes become static again at the configuration illustrated in Figure 7, the dual-path algorithm continues to run, correcting its position again if a) interference is detected, or if b) a non-split flow coming from a node other than the source node is received by an intermediate node.

V. CONCLUSIONS AND FUTURE WORK

With this preliminary study we have shown that it is possible to create non-interfering paths by modifying the topology of a MANET. However, our design only considers one concurrent data flow. Extending the algorithm to handle multiple data flows is our next step, however it is not a trivial one. The new version of the algorithm will need to deal with new models of mobility that will adapt to the multiple flows.

Also, our current implementation attempts to minimize interference by manipulating the physical position of the nodes, and this is possible thanks to the fact that the nodes are mobile and mobility can be controlled. However, we need to explore, the behavior of the algorithm in situations where node movement is just not possible. For these cases, changing the transmission power of the wireless network device might help modify the topology to a desirable state.

REFERENCES


