Routing With Adaptive Flooding in Heterogeneous Mobile Networks

Anand Seetharam[1], Simon Heimlicher[2], Jim Kurose[2], Wei Wei[2]
[1]Computer Science and Information Technology Program, California State University, Monterey Bay
[2] School of Computer Science, University of Massachusetts, Amherst, USA
aseetharam@csumb.edu, {heimlicher, kurose, weiwei}@cs.umass.edu

***Abstract*—In practice, Mobile Ad Hoc Networks (MANETs) must operate efficiently under a wide range of node mobility and link quality regimes. Stateful protocols such as OLSR are suitable for networks connected by stable paths, but are outperformed by stateless flooding in sparse and rapidly changing networks. Neither routing nor flooding alone perform well in heterogeneous networks comprised of both stable and highly dynamic components. Rather than design a new protocol for routing in heterogeneous mobile networks from scratch, we use an approach that leverages prior work by operating nodes individually as routers or flooders and switching mode in response to changing network conditions. We present two greedy algorithms, k-flood and adaptive-flood, that dynamically classify nodes as routers or flooders. Our simulations show that nodes can effectively adapt their individual operation as routers/flooders, achieving performance equivalent to, and in some cases significantly better than, that of network-wide routing or flooding alone.**

I. Introduction

Uncertainty and change in network connectivity are fundamental characteristics of mobile ad hoc networks (MANETs). A variety of forwarding strategies have been proposed for such scenarios, ranging from stateful routing protocols [1] to flooding [2]. Manfredi *et. al* [3] show that in mobile networks with *homogeneous* node mobility and link characteristics, stateful routing protocols such as OLSR [4] perform well in dense and stable networks, whereas flooding is preferable in sparse and rapidly changing networks. However, mobility and connectivity characteristics observed in real-world measurements are often *heterogeneous*: while some nodes may have few or highly dynamic links, there are also well-connected nodes forming sizable connected components [5], [6]. In such networks with both stable and dynamic components, it is likely (and we will see) that neither routing nor flooding alone may perform particularly well in a given scenario.

We propose a simple approach towards forwarding in heterogeneous mobile networks: based on local link characteristics as well as network-wide considerations, determine which nodes should forward according to the forwarding table computed by the native routing protocol, and which nodes should broadcast their traffic to all neighbors. Our work is driven by the intuition that nodes with particularly reliable and stable links should be well-suited to operate as routers, since the next-hop toward a given destination determined by the native routing algorithm would continue to work well in the future. Conversely, a node with highly dynamic or unreliable links might better operate as a flooder, exploiting the broadcast nature of common omni-directional antennas to efficiently forward a packet to all neighbors in a single transmission; packet copies can then be forwarded from one or more of those neighbors (either via routing or flooding by that neighbor) toward the destination. Practically speaking, such a simple approach leverages the vast amount of past research on both routing and flooding in mobile networks with minimal changes to existing protocols. From a performance standpoint, we will show that our simple approach not only matches the performance of network-wide routing or flooding in stable or dynamic settings, respectively, but also performs better than either of them in heterogeneous scenarios. Unlike [6], which presents a baseline routing scheme that uses flooding only as a fallback when a node lacks a valid path to a destination, we treat flooding as a first-class citizen.

In our approach, we have a single decision to make for each node — should it operate as a router or flooder — in such a way as to maximize global network goodput. Despite its apparent simplicity, this is a challenging problem. First, available link state information may be stale due to mobility and variability inherent in wireless links. Second, while it is tempting to think that classifying a node as a router or flooder only requires local information, flooding at one node increases network traffic at downstream nodes and may ultimately reduce overall goodput due to congestion. In addition, one node being a flooder may affect the usefulness of turning another node into a flooder, implying subtle dependencies in the decision process.

We present two simple greedy algorithms to determine which nodes should operate as routers and which nodes should operate as flooders. Both algorithms assume that an underlying native routing protocol is available and then determine the set of nodes that should operate as flooders such that overall network goodput is maximized. Practically, this means that each node needs to determine only one piece of information, namely whether to unicast packets to the next-hop neighbor specified in its forwarding table, or to locally flood each packet to all neighbors.

k-flood, our first algorithm, is based on a simple utility function indicating the usefulness of turning a node into a flooder. It assigns a utility to each node, giving preference to nodes with fewer and lower quality links and more potential incoming traffic. Given the flooding parameter k, this algorithm iteratively calculates the utility of each router and turns the router with the highest utility into a flooder until k flooders have been selected. While *k-flood* performs better than network-wide routing or flooding for most settings, it ignores

978-1-4244-8953-4/11/$26.00

Notation	Definition
N	set of all nodes
E	set of all edges
$\mathscr{F}$	set of flows
F	list of flooders
N_i	neighbors of node i
f_s, f_d	source and destination node of flow f
$\mathbb{N}$	is the list of N_i, $\forall i \in N$
H_{if}	next hop forwarder for node i for flow f
$\mathbb{H}$	next hop forwarder matrix
p_{ij}	link success probability between nodes i and j
$\mathbb{P}$	link success probability matrix
Φ_{ij}	traffic originating from i and destined for j
Φ	traffic matrix

TABLE I. NOTATION

capacity constraints and may lead to congestion.

adaptive-flood is a more sophisticated classification algorithm based on observations we have made with *k-flood.* This algorithm attempts to iteratively select those nodes as candidate flooders which maximize the *overall expected network goodput.* It picks nodes as flooders in decreasing order in which they contribute to maximizing expected network-wide goodput; it stops when converting any of the remaining routers into a flooder would result in a decrease in expected goodput.

This paper makes the following contributions: First, we present two greedy algorithms to determine which nodes should operate as routers and which nodes should operate as flooders. Second, we show via simulation that our classification algorithms outperform network-wide routing or flooding. In particular, at low network loads flooding outperforms routing while at high network loads, performance is reversed. In contrast, our classification algorithms match or outperform both baseline approaches over most or all of the range of loads in both homogeneous as well as heterogeneous scenarios. From these results, we conclude that routing combined with adaptive flooding is a promising solution to solve the challenges inherent in mobile networking.

The rest of this paper is organized as follows. We formalize the problem and the underlying network model in Section II, and describe the greedy algorithms for classifying nodes into routers and flooders in Section III. Simulation results evaluating the performance of the greedy algorithms are presented in Section IV. In Section V, we discuss related work. We conclude the paper and provide an outlook at future work in Section VI.

II. NETWORK MODEL

Let us begin by defining our network scenario and the router/flooder classification problem. We consider a network with $|N|$ nodes. Let $\mathscr{F}$ be the set of flows in the network. The source and destination for any flow f are denoted by f_s and f_d respectively. A summary of our notation is available in Table I.

Time Periods. Time is slotted and we introduce two intervals beyond the minimal interval defined as one time *slot*. Specifically, packet transmissions occur at each time slot, node mobility takes place at the beginning of each *interval,* which is a period of several time slots, and finally, routing tables are updated at the beginning of every *epoch,* which is a period of multiple intervals.

Link State Information. We assume that during each epoch, state information characterizing the connectivity between nodes is collected. Let us consider any two nodes i and j and informally consider p_{ij} as the probability of successfully transmitting a packet from node i to node j (we will substantially sharpen this definition of p_{ij} in Section IV where we use simulation to assess the performance of our greedy algorithms). The link quality can vary both due to the wireless channel and mobility of the nodes during the epoch, but we abstract away these details via the p_{ij} link characterization. $\mathbb{P}$ is the matrix of p_{ij}'s and is referred to as the link success probability matrix.

We next represent the network as a graph $G(N, E)$ where E denotes the set of all edges in the graph; an edge exists if $p_{ij} > 0$.

Routing and flooding. We consider a simple case where $\mathbb{P}$ is the only state information available at the beginning of an epoch and is used for determining routes. At the beginning of each epoch, we assume that routes are calculated using Dijkstra's shortest path algorithm where an edge between node i and j has link weight $1/p_{ij}$. Note that any other routing algorithm could be used; we use Dijkstra's algorithm for simplicity. H_{if} denotes the next hop neighbor for node i for flow f, as obtained by the routing algorithm.

Each node in the network can either act as a router or a flooder, but cannot perform both actions preferentially based on the destination of the packet. If node i operates as a router, it forwards packets according to H_{if}; otherwise it floods all packets. Let N_i be the list denoting the neighbors of i (node j is said to be a neighbor of i if $p_{ij} > 0$). If i operates as a flooder, it sends the same packet to every node in N_i. To prevent packets from circulating in the network in loops, nodes perform duplicate packet transmission suppression.

Traffic and Capacity. We assume that node i transmits data at a rate of C_i packets per time slot. Therefore all outgoing links from i can carry data at the maximum rate of C_i. C_i and p_{ij} together capture the capacity constraint for the link ij. Φ_{ij} is the amount of traffic originating at node i and destined for node j; Φ is the corresponding traffic matrix.

Overall network goodput. We define the *goodput* for a flow as the number of unique packets received at the destination for the flow per time slot. The overall network goodput is thus the sum of goodputs for the different flows.

Based on this model, we define the problem to be solved by the classification algorithm *at the beginning of every epoch* as follows.

Given the above network model and the forwarding tables from the routing algorithm, classify certain routers as flooders so as to maximize overall network goodput. We note that our algorithms for solving this problem pre-suppose the presence of a native routing algorithm and will execute periodically, following the execution of the native routing algorithm; our work thus fall squarely in the network control plane.

III. ALGORITHMS FOR ROUTER/FLOODER CLASSIFICATION

In this section, we propose two simple greedy algorithms for router/flooder classification. Our algorithms, which operate

in the MANET control plane (e.g., would execute following the periodic execution of the network's native stateful routing algorithm), classify each network node as a router or as a flooder. Nodes classified as routers will unicast-route packets according to forwarding tables computed by the MANET's native stateful routing algorithm; nodes classified as flooders will *locally* flood a packet to all one-hop neighbors, who will then in turn unicast-route or flood (depending on their own classification) that packet. We note that similar to stateful routing protocols such as OLSR, our router/flooder classification algorithms can be run *locally* given broadcast link state updates, as discussed shortly.

The first algorithm — referred to as *k-flood* — iteratively re-classifies one of the routing nodes to be a flooder, according to a simple utility function that takes link transmission success probabilities (p_{ij}) and the amount of traffic flowing through network nodes into account. *k-flood* requires an input parameter k, stopping when k nodes have been selected as flooders. The second algorithm - known as *adaptive-flood* - iteratively re-classifies one of the routing nodes to be a flooder, selecting that router whose change to a flooder would result in the maximum increase in expected network goodput, and stopping when turning any remaining router into a flooder would result in a decrease in expected total goodput.

We then compare the performance of *k-flood* and *adaptive-flood* with two baseline approaches: *routing* (where all network nodes operate as routers and forward packets to next hop neighbors based on the MANET's native routing algorithm, which we will assume to be based on Dijkstra's algorithm) and *flooding* (where all network nodes operate as flooders and forward every packet to all their neighbors). Flooders perform duplicate suppression so as to not forward the same packet twice.

A. The k-flood Algorithm

We begin by describing the *k-flood* algorithm for router/flooder classification. The algorithm, given an input parameter k, stops when k nodes have been classified as flooders. The details of the algorithm are given in Alg. 1. Initially, all nodes are classified routers (i.e., $F = [\,]$; see Table I for notation), with forwarding tables $\mathbb{H}$, which are determined by the native unicast routing algorithm assuming that all nodes are classified as routers. The greedy algorithm then iteratively selects a node (F') to be turned into a flooder that maximizes a utility function. In Alg. 1, the *calculate-utility* function returns $\vec{U}$ — a vector containing the utilities of all nodes. After each iteration, the size of the list of flooding nodes, F, increases by one. Note that the algorithm re-evaluates the utilities of all routers during each iteration, since a packet locally flooded by the newly classified flooder to its neighbors can now traverse additional paths to its destination, thus changing the utility of that node from the previous iteration).

We next describe the details of the *calculate-utility* function (Alg. 1) used by *k-flood* to calculate the utility of turning a router into a flooder. Let F be the set of nodes classified as flooders when the *calculate-utility* function is called. Initially, when all nodes are routers, the computed forwarding tables are such that traffic for each flow is routed along the simple path (containing no flooders) computed by the native unicast routing algorithm. When one or more network nodes are classified as flooders, however, the set of nodes and links traversed by a given flow's packets will form a directed acyclic graph (rather than a path) between the flow's source and destination nodes. As a node never forwards the same packet twice, loops are avoided, thereby ensuring that packets for a particular flow move along a DAG.

In *k-flood,* the utility function for router i is given by $U_i = \sum_{j \in K_i} \frac{w_{ij}}{p_{ij}}$. The weight w_{ij} corresponds to the total traffic (from all flows), for the router and flooder lists computed at this iteration, that will cross link ij assuming loss-free links and no capacity constraints. w_{ij} considers both the exogenous traffic arriving at node i and the endogenous traffic that i relays/forwards towards a destination for flows exogeneously arriving at other source nodes. Recall that p_{ij} refers to the successful packet transmission probability between nodes i and j. The intuition behind the utility function is the following: if a node is forwarding a high volume of traffic (exogenous and endogenous) and has outgoing links with low packet transmission success probabilities, then it might be useful to turn that router into a flooder so that its packets are more likely to be correctly received by at least one of its neighbors, and then find their way from that neighbor to the destination. On the contrary, if a node has highly reliable, stable links to neighbors, it might be better for it to operate as a router, as converting it into a flooder would create unnecessary duplicate packets in the network.

To calculate the weights, *calculate-utility()* considers the graph $G'(N, E)$ whose edges are all initially unweighted i.e., $w_{ij} = 0, \forall i, j$. For every flow $f \in \mathscr{F}$ the algorithm begins by constructing the DAG traversed by flow, increasing the weight of the corresponding edges of G' by the traffic rate for that flow, $\Phi_{f_s f_d}$. The DAG construction algorithm maintains two lists: the observed list - O and the explore list - X. For every flow f, DAG construction begins from the source f_s. X and O initially contain only f_s and f_d respectively (line 11). The while loop in line 12 then iterates until the explore list is empty. At every iteration of the while loop, the node (m) at the head of X (line 13) is considered (in the first iteration the node is f_s). Recall that when the *calculate-utility* function is called, there are F flooders in the network. Therefore m can be either a router or a flooder. In either case (lines 15–19 for a flooder; lines 21–23 for a router), we update the weights of all links from m to its one or more neighbors and add each neighbor to the explore list if it is not already on the explore or observed list. Nodes are checked before being added to the observed list to avoid loops in the DAG. The outer for loop in line 10 terminates when all flows $f \in \mathscr{F}$ in the network have been considered. Having computed the values of all w_{ij}, the algorithm then computes the utility function for all routers and returns this value to the *k-flood* algorithm.

k-flood is based on a simple intuition for classifying nodes as routers/flooders; it takes neither capacity constraints nor network congestion into account. Nevertheless it helps illustrate the challenge involved in assessing the value of turning a router into a flooder — one needs to take into account link quality as well as the interactions among nodes (a node's own traffic as well as traffic passing through it) when designing an algorithm for router/flooder classification. The above algorithm

also requires a stopping criteria, k. One could iterate over all values of k to determine the best value of k. However, a natural and rather straightforward stopping criteria is to stop changing routers into flooders when doing so would not increase (and could possibly decrease) overall network performance. This insight then leads to our next algorithm.

Algorithm 1 *k-flood* router/flooder classification

1: **function** F = *k-flood*$(k, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F})$
2: $F = [\,]$
3: **while** $|F| \neq k$ **do**
4: $\vec{U}$ = *calculate-utility*$(F, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F})$
5: $[F'] = \text{argmax}\,\vec{U}$
6: $F = F + [F']$;
7: **return** F

8: **function** $\vec{U}$ = *calculate-utility*$(F, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F})$
9: $\vec{U} = 0$, $w_{ij} = 0, \forall i, j$
10: **for all** $f \in \mathscr{F}$ **do**
11: $X = [f_s]$, $O = [f_d]$
12: **while** $X \neq [\,]$ **do**
13: $[m] = head(X)$
14: $O = O + [m]$
15: **if** $m \in F$ **then**
16: **for all** $i \in N_m$ **do**
17: $w_{mi} = w_{mi} + \Phi_{f_s f_d}$
18: **if** $i \notin O$ **and** $i \notin X$ **then**
19: $X = X + [i]$
20: **else**
21: $w_{mH_{mf}} = w_{mH_{mf}} + \Phi_{f_s f_d}$
22: **if** $H_{mf} \notin O$ **and** $H_{mf} \notin X$ **then**
23: $X = X + [H_{mf}]$
24: $X = X - [m]$
25: **for all** $i \notin F$ **do**
26: $U_i = \sum_{j \in N_i} \frac{w_{ij}}{p_{ij}}$
27: **return** $\vec{U}$

B. The adaptive-flood algorithm

In this subsection we present the *adaptive-flood* algorithm. *adaptive-flood*, like *k-flood* is an iterative greedy algorithm that turns one router into a flooder at each iteration. In *adaptive-flood*, the router that is turned into a flooder is that router whose change in status would greedily maximize overall network goodput. The details of the algorithm are given in Alg. 2.

As with *k-flood*, *adaptive-flood* also begins with all network nodes initially classified as routers. It then computes the expected total (over all source/destination pairs) goodput for the $\mathscr{F}$ flows by calling the *total-goodput()* function. In Alg. 2, $\mathscr{G}_F$ is the expected total network goodput when there are F flooders. During each iteration in *adaptive-flood* (lines 4–14), the expected total goodput is calculated if router s were to be turned into the flooder, given the current list of routers and flooders. The algorithm then selects that particular router (F') that gives the maximum increase in expected total goodput if it were to be turned into a flooder (lines 6–10). This router is then added to the list of flooders F. The algorithm terminates when either all routers have been classified as flooders or if converting each of the remaining routers into a flooder (individually) results in a *decrease* in expected total goodput. As in *k-flood,* when a router is added to F, the usefulness (in terms of goodput) of converting some other router into a flooder can change, since converting a node into a flooder can change the incoming traffic rates at other network nodes.

Algorithm 2 *adaptive-flood* router/flooder classification

1: **function** F = *adaptive-flood*$(G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
2: $F = [\,]$
3: $\mathscr{G}_F$ = *total-goodput*$(F, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
4: **while** $|F| \neq |N|$ **do**
5: $F' = [\,]$
6: **for all** $s \notin F$ **do**
7: $T = F + [s]$
8: $\mathscr{G}_T$ = *total-goodput*$(T, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
9: **if** $\mathscr{G}_T > \mathscr{G}_F$ **then**
10: $F' = s$, $\mathscr{G}_F = \mathscr{G}_T$
11: **if** $F' == [\,]$ **then**
12: **return** F
13: **else**
14: $F = F + [F']$
15: **return** F

16: **function** $\mathscr{G}_F$ = *total-goodput*$(F, G', \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
17: $\vec{a}^0 = 0$, $l = 1$
18: **for all** $f \in \mathscr{F}$ **do**
19: D_f = *calculate-DAG*$(f, F, G', \mathbb{N}, \mathbb{H})$
20: V_f = *topological-sort*(D_f)
21: **while** (true) **do**
22: **for all** $f \in \mathscr{F}$ **do**
23: $\vec{I_f}$ = *calculate-incoming-rate*$(D_f, V_f, \Phi, \mathbb{P}, f, \vec{a}^{l-1})$
24: **for all** $s \in N$ **do**
25: $R_s = \sum_{f \in \mathscr{F}} I_{fs}$
26: $a_s^l = \min\{1, \frac{C_s}{R_s}\}$
27: **if** $\max|\vec{a}^l - \vec{a}^{l-1}| < \tau$ **then**
28: **return** false
29: $l = l + 1$
30: **for all** $f \in \mathscr{F}$ **do**
31: g_f = *calculate-goodput*$(D_f, V_f, \Phi, \mathbb{P}, f, \vec{a}^l)$
32: $\mathscr{G}_F = \sum_{f \in \mathscr{F}} g_f$
33: **return** $\mathscr{G}_F$

Calculating the effect of a router-to-flooder change on total network goodput

The *total-goodput()* function computes the overall (over all flows) goodput, given a list of routers and a list of flooders and the next-hop forwarding matrix $\mathbb{H}$ computed via Dijkstra's algorithm. Doing so is challenging for two reasons. First, link capacities are finite and buffer overflows will occur when the incoming traffic rate exceeds a node's capacity to send that traffic on its going link(s). Second, given the presence of flooding nodes in the network, multiple copies of the same packet may be received at a node, and via duplicate suppression, only a single copy of that packet will be forwarded. Thus, traffic input rates to nodes need not equal their output rate, even in the absence of congestion losses due to limited link capacities.

Modeling the effects of finite buffer overflow. We model buffer overflow by adopting a fluid model in which nodes probabilistically drop packets if the expected incoming traffic rate exceeds that node's outgoing transmission capacity, C_i. Let a_i denote the probability that a packet is successfully received and forwarded through node i, assuming no losses due to transmission errors. We refer to a_i as the *packet-passage probability* at node i. Let R_i be the incoming traffic rate at node i. Then:

$$a_i = \min\{1, \frac{C_i}{\mathbb{E}[R_i]}\} \tag{1}$$

Thus, when the expected incoming traffic rate is less than link capacity, all arriving packets are successfully forwarded by that node, $(a_i = 1)$. When the expected incoming traffic rate exceeds the outgoing rate, arriving packets are successfully forwarded by that node with probability $\frac{C_i}{\mathbb{E}[R_i]}$. We note that this simple model of congestion is used *only* for calculating goodput in our control plane algorithm, *adaptive-flood.* The MANET's data plane itself performs packet dropping due to buffer overflow according to its native policy; our model calculations of goodput only affect the control-plane router/flooder classification. We next use our model to determine the overall network goodput.

The *total-goodput()* function presented at the bottom of Alg. 2 returns the total goodput for the $\mathscr{F}$ flows in the network, given the list of flooders (F). The total network goodput is calculated by summing the goodput for the individual flows in the network (Lines 30–32). To calculate the goodput of individual flows, we need to know the packet-passage probabilities at all network nodes. Even though packets for a given flow traverse a DAG, when there are multiple flows in the network, it is possible that some node j will receives traffic from node i and vice versa. In this case, since a_i depends on R_i which includes traffic arriving from j, and a_j depends on R_j which includes traffic arriving from i, we'll need to compute the packet-passage probabilities via a set of simultaneous equations.

Lines 21–29 in Alg. 2 are a fixed point iteration for calculating the packet-passage probabilities, $\vec{a}$. The algorithm begins with an initial feasible packet-passage probability $(\vec{a}^0)$ (in our case 0). For each flow f, the fixed-point iteration then uses $\vec{a}^{l-1}$ at iteration l to calculate the incoming rate at every node (line 23). $\vec{I_f}$ is a vector of the incoming rates at different nodes for flow f, while I_{fs} denotes the incoming rate for flow f at node s. The fixed point iteration then uses the incoming rates to compute the packet-passage probabilities to be used in iteration $l+1$ (line 26), $\vec{a}^l$. The fixed-point iteration converges when the maximum absolute difference between the packet-passage probabilities in two successive iterations are all below a threshold τ (line 27). *total-goodput()* then computes the total goodput and returns this value to the *adaptive-flood* algorithm. Note that the goodput for flow f is simply the incoming traffic for flow f at node f_d.

Modeling multiple copies of a packet, duplicate suppression. All that remains to be discussed is how $\vec{I_f}$, the incoming traffic rate at node i, is determined in line 23 of *total-goodput().* The primary complication here is that multiple copies of the same packet in flow f may arrive at a node due to upstream flooders in flow f's DAG. Duplicate copies would be suppressed, resulting in only a single copy being forwarded to the node's output interface. Since flooding is a central to router/flooder classification, our model must accommodate multiple copies of a packet. We delve into this challenge in our technical report [7].

Here, we summarize our approximate approach for computing $\vec{I_f}$, the incoming traffic rate at node i, as follows. To compute $\vec{I_f}$ it is necessary to know the DAG which each flow traverses. Therefore in Alg. 2 (lines 19–20), we first determine the DAG (D_f) traversed by that flow's packets. This can be done using the same approach as in *k-flood.* We then obtain a topological sort V_f for D_f. For a directed acyclic graph, the topological ordering provides a linear ordering of its vertices such that for every directed edge from vertex u to vertex v, u comes before v in the ordering. The source f_s and destination f_d are the first and last nodes in this ordering (V_f). We then pass D_f and V_f as parameters to the *calculate-incoming-rate()* function (line 23 in Alg. 2). Note that D_f and V_f are calculated only once in the *total-goodput()* function because the list of flooders does not change between iterations of the while loop in line 21 in Alg. 2.

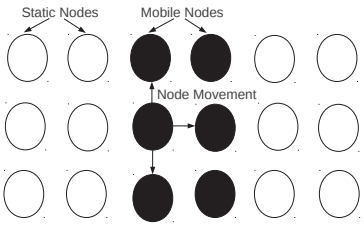


Fig. 1. Topology: 18 Node Network

We next evaluate the probability of a packet reaching the nodes in V_f in order in the *calculate-incoming-rate()* function. To determine the probability of a flow f packet (which has been flooded one or more times upstream in flow f's DAG) reaching node i, we assume that the probabilities of i receiving that packet on its incoming DAG links are independent of one another. This is clearly an approximation, since two input links at i may share common upstream nodes in the DAG. Let α_{if} be the probability that a packet reaches node i for flow f. Let us consider any node j in V_f and let U_j denote the list of the nodes in V_f appearing before j in this ordering. We approximate the probability of a packet reaching j by:

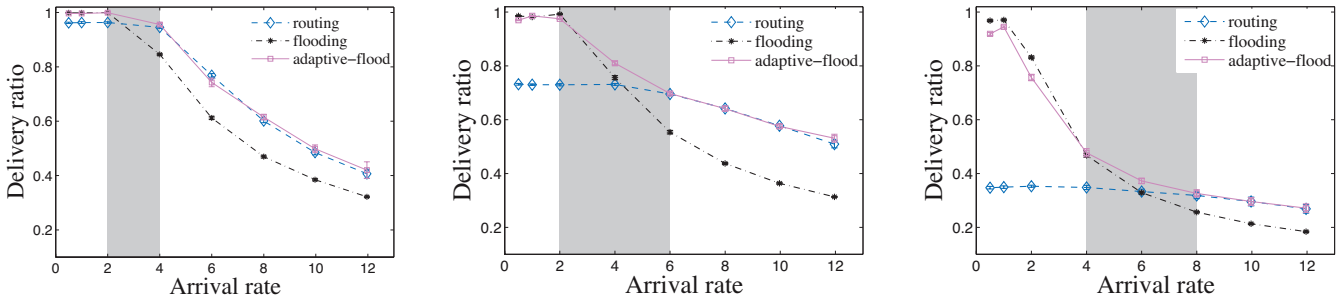
$$\alpha_{jf} = 1 - \prod_{i \in U_j} (1 - \alpha_{if} p_{ij} a_i) \tag{2}$$

Equation (2) takes into account the fact that the packet can be received along multiple incoming links. It is also takes the successful link transmission probabilities and the packet-passage probabilities into account. Traversing V_f in order ensures that when the algorithm calculates α_{jf} for node i, α_{jf} of all nodes j in U_j has already been computed.

IV. Simulation Results

In this section, we report on simulations comparing the performance of our two greedy algorithms, *k-flood* and *adaptive-flood,* with pure network-wide routing and flooding. We find that the greedy algorithms capture the best of both approaches (routing and flooding), achieving performance equivalent to (and sometimes better than) that of network-wide routing or flooding alone.

Our simulations are conducted on a grid topology with r rows and c columns. An example topology with 18 nodes is shown in Figure 1 (nodes colored white are stationary while the ones colored black are mobile). Thus there are two regions with stationary nodes separated by an intervening mobile region. This specific topology is used to generate results in Figure 2. We chose this topology in order to stress-test and study our algorithms, ensuring that we could create controlled scenarios in which source-destination flows pass through both static



(a) Delivery Ratio (Scenario 1) (b) Delivery Ratio (Scenario 2) (c) Delivery Ratio (Scenario 3)

Fig. 2. Delivery ratio with different sets of flows for an 18-node network

and mobile regions. We also present results for a larger 48-node topology later in this section. At the beginning of the simulation, there is one node per grid location.

We assume a time-slotted system and consider three time periods of different granularity - slots, intervals and epochs. A *slot* is the time taken for a packet transmission. An *interval* consists of multiple slots. A mobile node moves equi-probably to any of the adjacent positions on the grid (up, down, left or right) at the beginning of an interval. However, mobile node movement is confined to the mobile region.

The time period of the longest duration is an *epoch,* consisting of multiple intervals. Unicast routes are calculated using Dijkstra's algorithm at the beginning of each epoch. Our *adaptive-flood* algorithm also executes at this time granularity, classifying nodes as routers/flooders. In Dijktsra's algorithm, link weight values are equal to $1/p_{ij}$, where p_{ij} is the fraction of intervals in the previous epoch that node i and j were in adjacent or the same grid positions; the value of p_{ij} between two adjacent stationary nodes i and j is thus always 1. The link success probability matrix Φ is populated at the beginning of the epoch, before Dijkstra's algorithm is executed.

We study three scenarios (with different sets of flows) for the 18-node network. We study two different performance metrics: overall network goodput and delivery ratio by varying the exogenous arrival rate. The delivery ratio is the ratio of the total number of unique packets delivered to the total number of exogenous packet arrivals for the entire duration of the simulation. The arrival rate is the expected total number of exogenous packet arrivals per time slot. Due to lack of space we omit details, which are available in [7].

- **Scenario 1:** We consider mostly short (2-hop) flows. Every node in the static region has a single 2-hop flow destined to a randomly chosen other node in the same static region (12 flows in all). There are also 3 single-hop flows in the mobile region.
- **Scenario 2:** We have 12 flows in the static regions. There are also 12 short flows originating from, and destined to, the mobile region and 3 flows originating from one static region and destined to the other static region.
- **Scenario 3:** In contrast to the other two cases, we have 25 flows in all, (some destined from one static region to other, some within the mobile region and some between mobile and static regions).

In the first scenario, most flows are confined to the static region; in the second there is a mix of flows in the static and mobile region; while in the third scenario, flows either cross, or are destined to, the mobile region. Hence, the main difference among the three scenarios is that the overall reliability of routes decreases, progressing from the first scenario to the third. Consequently, one would intuitively expect routing to generally outperform flooding in the first scenario, while the opposite would occur in the third scenario.

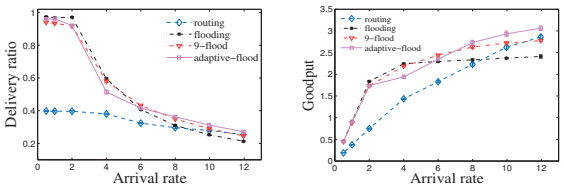
A. Simulation Results: comparing routing, flooding, k-flood, and adaptive-flood.

Comparison of pure routing and flooding. Figure 2 shows the delivery ratio of the different algorithms for the above three scenarios. For scenario 1 (Figure 2(a)), we observe that pure (i.e., network-wide) routing performs comparable to pure flooding in the low arrival rate regime but then outperforms pure flooding as the arrival rate increases. The difference in delivery ratio at low arrival rate is due to the fact that in case of pure routing, packets are dropped in the mobile region; in the case of pure flooding, packet duplication via flooding ensures that at least one copy of most packets get delivered to the destination. The reason for the relatively poorer performance of flooding at higher arrival rates is that as the network becomes congested, duplicate packets cause other packets to be discarded at intermediate routers, resulting in decreased goodput.

In scenario 2 (Figure 2(b)), we observe that pure flooding outperforms pure routing at low arrival rates, while the relative performance ordering is reversed at higher arrival rates. Since approximately half of the flows are in the mobile region (and these flows have less reliable paths), pure routing has a low delivery ratio at low arrival rates. Once again, increased congestion results in poor performance of flooding at higher arrival rates. In scenario 3 (Figure 2(c)), as end-end path reliability is low, pure routing performs poorly, marginally overtaking pure flooding as the arrival rate increases. The three scenarios thus demonstrate situations when pure routing outperforms pure flooding, and vice versa.

***adaptive-flood* outperforms both pure flooding and pure routing.** Next, we turn our attention to the performance of our *adaptive-flood* algorithm[1]. The shaded regions in Figure

[1]In some cases, packet-passage probabilities in the fixed point iteration in the *calculate-goodput* function of *adaptive-flood* algorithm do not converge to a fixed point, often oscillating between two sets of values. In such cases we select one of the sets of values and use it to determine the total goodput.



(a) Delivery Ratio (b) Goodput

Fig. 3. Delivery Ratio and Goodput for a 18-node network

2 indicate the arrival rate regime where the *adaptive-flood* outperforms both routing and flooding. It is evident from the figure that while flooding and routing perform well at low and high arrival rates respectively, the *adaptive-flood* algorithm achieves performance equivalent to (and better in the shaded regions) than that of pure routing or flooding alone. For example in Figure 2(b), the performance of *adaptive-flood* exceeds that of both routing and flooding in the shaded region, for arrival rates between 2 and 6 arrivals per time slot. The superior performance of the *adaptive-flood* algorithm can be attributed to the fact that it dynamically adapts the number of flooders selected based on the arrival rate. For example in Figure 2(b), the algorithm selects around 10 nodes as flooders when the arrival rate is 2 and selects 2.11 nodes on average as flooders when the arrival rate is 12. We noted that *adaptive-flood* also often selects stationary nodes as flooders. Turning stationary nodes into flooders can present multiple entry points into the mobile region. Also if a given stationary node is congested because of a large number of flows through it, turning other stationary nodes into flooders can help find additional paths for these flows, thus increasing goodput.

To focus on the performance of *adaptive-flood* (which performs either equivalent or better than *k-flood* for a fixed k) and to avoid cluttering the graph, we have not shown the performance of *k-flood* in Figure 2. In addition, we note that while the *k-flood* algorithm can perform better than either routing or flooding alone, it fails to capture the best of both approaches. For example in Figure 2(b) at high arrival rates, routing has higher delivery ratio than *9-flood*. This is because *9-flood* always selects exactly 9 flooders, regardless of the scenario (e.g., arrival rate). We observed that *9-flood* mainly picks mobile nodes and stationary nodes bordering the mobile region as flooders. This selection conforms to intuition because the algorithm is likely to pick nodes with poor outgoing links as flooders.

Performance with a larger number of flows. We next consider the 18-node network scenario, where each node originates flows destined to every other node. There are thus $18 * 17 = 306$ flows in the network. Figure 3 shows the delivery ratio and goodput for different arrival rates for the various algorithms. Once again we observe that when the arrival rate is low, the delivery ratio of pure flooding is higher than pure routing while the opposite holds at high arrival rate. We also observe that the performance of our greedy algorithms is equivalent or superior to either pure routing or flooding. Interestingly, unlike the previous scenarios, the performance of the *k-flood* and *adaptive-flood* algorithms are comparable to one another, with *k-flood* marginally outperforming *adaptive-flood*.

Comparing goodput for the different schemes (Figure 3(b)),

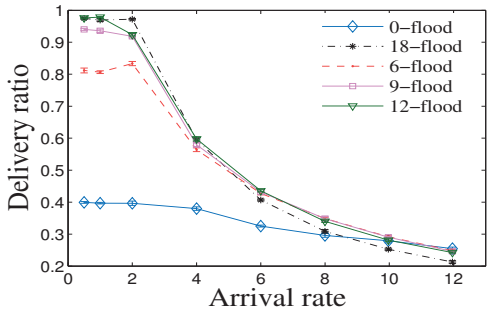
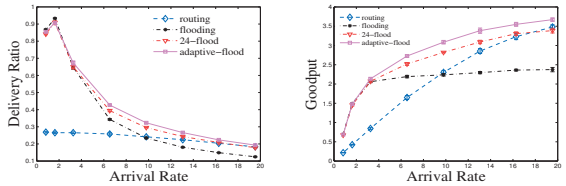


Fig. 4. Delivery Ratio: *k-flood algorithm*



(a) Delivery Ratio (b) Goodput

Fig. 5. Delivery Ratio and Goodput for a 48-node network

we find that, as expected, flooding outperforms routing in the low arrival rate regime. Interestingly, although the delivery ratio decreases with increasing exogenous arrival rate, the goodput increases since the absolute number of packets delivered increases with higher arrival rate.

Comparing performance of *k-flood* with different values of k. Thus far, all *k-flood* results have been reported with 9 flooders. Figure 4 shows the delivery ratio of the *k-flood* algorithm for different number of flooders for the 18-node network with all nodes sending packets to all other nodes. We observe in Figure 4, that (as expected) when the number of flooders is small, the behavior of *k-flood* is similar to routing and when the number of flooders is large, its behavior is closer to flooding.

Performance results for 48-node network with large number of flows. We also conducted experiments on a larger 48-node network, with nodes arranged in a grid with 4 rows and 12 columns. The static and mobile areas thus consist of 4*4 grids. Figure 5 shows the delivery ratio and goodput for this network with all nodes sending packets to all other nodes. We again observe that the greedy algorithms outperform pure routing and flooding both in terms of delivery ratio and goodput. We also observe that the *adaptive-flood* algorithm outperforms the *k-flood* algorithm.

V. Related Work

Several past research efforts [3], [8], [9] have addressed the challenge of classifying MANETs based on connectivity and predictability – concerns that are of central importance to us in this paper. [3] proposes a framework for organizing the decision space of communication strategies (i.e., determining whether the network as a whole should operate by flooding, routing, or store-carry-and-forward) in a *homogeneous* MANET based on connectivity and unpredictability so as to maximize goodput. Similar approaches for classifying networks (as connected, intermittently connected or disconnected) based on connectivity (i.e., presence of paths) and mobility

(i.e., contact time, meeting) have been investigated [8], [9]. In contrast to prior work where classification has been done for the network as a whole, we develop per-node classification strategies (route or flood) in order to maximize goodput.

A number of past efforts have sought to exploit characteristics such as connectivity, predictability and mobility of wireless networks to design forwarding protocols that enhance performance. Epidemic routing [10], [2] and multicopy routing [11] are designed for sparsely-connected networks and use a store-carray-and-forward mechanism and packet replication to battle poor connectivity. [12], [13] make assumptions on the mobility pattern and network topology to design forwarding protocols for intermittently connected networks. A survey of different forwarding strategies designed for MANETs and DTNs is available in [14], [15]. None of this past research, however, investigate the question of which nodes should flood/route in a MANET with time-varying connectivity.

Our work is closest to [6], which proposes a routing protocol, R3, that provides robust performance in diverse and varying connectivity regimes. They identify packet replication as the key factor governing performance for networks at opposite ends of the connectivity spectrum (meshes and DTN). R3 replicates packets along two paths for each flow, pruning one of the paths in the event of network congestion. They also propose the SWITCH protocol, in which nodes make decisions locally and flood packets only when the designated next hop for that packet is unavailable. SWITCH performs close to R3 in their evaluation. Our work differs from R3 in that we determine which nodes should flood/route in a network-wide context, taking multiple flows into account when making a routing/flooding decision and not relying on replication solely at the source. Our algorithm also differs from SWITCH in that we determine which nodes should flood over an epoch of time, not just flood when a next-hop neighbor towards a destination in is unavailable.

A significant amount of past research has been devoted to demonstrating the capacity scaling of both flat and hybrid MANETs [16], [17], [18]. Similarly several forwarding strategies (routing, flooding and hybrid) aimed at improving goodput have been proposed and extensively studied for both MANETs and DTN [1], [19], [12]. Our work differs from existing literature in that we are not proposing a new forwarding protocol from scratch for a particular setting; rather we study the problem of enhancing network goodput by selectively classifying a subset of nodes as flooders and the remaining as routers.

VI. Conclusion

We have studied the problem of forwarding in heterogeneous mobile networks that comprise both stable as well as highly dynamic components and in which uniform routing or flooding at all network nodes does not perform well. Instead of designing a new protocol, we leverage past efforts by exploring a simple and intuitive approach that individually determines for each node whether it should operate as a router or a flooder based on considerations such as the quality of its links, the amount of traffic traversing it, and the effect of turning a router into a flooder on overall goodput. We have proposed two greedy algorithms — *k-flood* and *adaptive-flood* — that iteratively classify nodes as routers or flooders. Via simulation, we have shown that these algorithms allow to effectively adapt the individual operation of each node to serve as a router or flooder, yielding performance equivalent to, and often significantly better than, that of baseline routing or flooding alone.

In future work, we plan to improve the computational efficiency of our algorithms and to investigate the performance gains achievable by preferentially routing or flooding packets based on their destination.

VII. Acknowledgment

This work was primarily done while all the authors were at University of Massachusetts Amherst.

References

[1] C. E. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *IEEE Workshop on Moble Computing Systems and Applications*, 1999.

[2] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," *Technical Report, Duke University*, 2000.

[3] V. Manfredi, M. Crovella, and J. Kurose, "Understanding stateful vs stateless communication strategies for ad hoc networks," in *ACM Mobicom*, 2011.

[4] P. Jacquet, Muhlethaler, A. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Hipercom Project, INRIA*.

[5] S. Heimlicher and K. Salamatian, "Globs in the Primordial Soup: The Emergence of Connected Crowds in Mobile Wireless Networks," in *ACM MobiHoc*, 2010.

[6] X. Tie, A. Venkataramani, and A. Balasubramanian, "R3: Robust replication routing in wireless networks with diverse connectivity characteristics," in *ACM Mobicom*, 2011.

[7] A. Seetharam, S. Heimlicher, J. Kurpse, and W. Wei, "Routing with adaptive flooding in heterogeneous mobile networks (technical report)," http://itcdland.csumb.edu/∼aseetharam/paper/techreport-comsnets.pdf.

[8] R. Ramanathan, P. Basu, and P. Krishnan, "Towards a formalism for routing in challenged networks," in *ACM CHANTS*, 2007.

[9] D. Antonellis, A. Mansy, K. Psounis, and M. Ammar, "Towards distributed classication for mobile ad hoc networks," in *In Proceedings of the 4th Annual International Conference on Wireless Internet*, 2008.

[10] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance modeling of epidemic routing," *Computer Networks*, 2007.

[11] K. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient routing in intermittently connected mobile networks: The multiple-copy case," *IEEE/ACM Trans. Networking*, 2008.

[12] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," in *ACM Sigcomm*, 2007.

[13] M. Demmer and K. Fall, "Dtslr: Delay tolerant routing for developing regions," in *ACM NSDR*, 2007.

[14] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Ad Hoc Networks*, 2004.

[15] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges," *IEEE Communications Surveys and Tutorials*, 2006.

[16] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Trans. on Information Theory*, vol. 26, no. 02, 2000.

[17] A. Zemlianov and G. De Venciana, "Capacity of an ad hoc wireless network with infrastructure support," *JSAC*, vol. 3, no. 25, 2005.

[18] B. Liu, Z. Liu, and D. Towsley, "On the capacity of hybrid wireless networks," in *IEEE Infocom*, 2003.

[19] T. Clausen, P. Jacquet, and L. Viennot, "Comparative study of routing protocols for mobile ad-hoc networks," in *MedHoc*, 2002.