On the Goodput of Flows in Heterogeneous Mobile Networks

Anand Seetharam, Arti Ramesh
Department of Computer Science, SUNY Binghamton, USA
aseethar@binghamton.edu, artir@binghamton.edu

Abstract

In practice heterogeneous networks comprising of diverse nodes need to operate efficiently under a wide range of node mobility and link quality regimes. In this paper, we propose algorithms to determine the goodput of flows in heterogeneous mobile networks. We consider a scenario where some network nodes operate as routers while others operate as flooders, based on the underlying forwarding policy. When a node operates as a router, it forwards packets based on the routing table as determined by the underlying routing algorithm and when it operates as a flooder, it broadcasts packets to all its neighbors. We begin with the case of a single network flow and demonstrate that the problem of determining the goodput is challenging even for this simple setting. We construct a Bayesian network, and propose an algorithm based on the sum-product algorithm to determine the exact goodput. We extend the proposed Bayesian network model for exact goodput calculation to feed forward networks with multiple flows. For a general network with multiple flows, the problem becomes more challenging. The difficulty of the problem stems from the fact that node pairs can forward traffic to one another, resulting in cyclical dependencies. We propose a fixed-point approximation to determine the goodput in this case. Finally, we present an application scenario, where we leverage the fixed-point approximation to design a forwarding strategy *adaptive-flood* that adapts seamlessly to varying networking conditions. We perform simulations and show that *adaptive-flood* can effectively classify individual nodes as routers/flooders, achieving performance equivalent to, and in some cases significantly better than that of network-wide routing or flooding alone.

1. Introduction

The emergence of the Internet of Things has seen rapid growth and deployment of heterogeneous networks comprising of diverse nodes (e.g., mobile nodes, static nodes, sensor nodes) with varying capabilities to connect to the Internet (e.g., WiFi, 4G). Uncertainty and change in network connectivity due to node mobility and wireless link quality are fundamental characteristics of such networks. Analytically determining the *goodput* of flows in heterogeneous networks is an important, yet unsolved problem. A quick search yields plethora of research papers that determine the throughput of wireless networks, each subject to its own set of assumptions and constraints [1, 2, 3]. However, limited attention has been devoted to determining the goodput of flows in a mobile and wireless setting. In comparison to throughput, goodput only takes the number of unique packets reaching the destination into account.

In wireless networks, a variety of forwarding strategies have been proposed, ranging from stateful routing protocols [4] to flooding [5, 6]. The pros and cons of maintaining state in dynamic wireless networks has also been investigated [7, 8]. Manfredi *et al.* [9] show that in mobile networks with *homogeneous* node mobility and link characteristics, stateful routing protocols such as OLSR [10] 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 highly dynamic links, there are also well-connected nodes forming sizable connected components [11, 12]. In heterogeneous networks with both stable and dynamic components, it is likely that neither routing nor flooding alone may perform particularly well in a given scenario. In such scenarios, performance can be improved by operating nodes as routers or flooders, depending on the network characteristics. However, flooding by network nodes can result in duplicate packets reaching the destination, thus making it necessary to analyze goodput instead of throughput to evaluate performance.

In this paper, an extension of our prior work [13], we consider a heterogeneous network where some nodes operate as routers while others operate as flooders, as specified by the underlying forwarding policy. *Our objective is two-fold in this paper. We propose algorithms to compute the goodput of flows in a heterogeneous network based on local link characteristics as well as the forwarding behavior of particular nodes (i.e., router or flooder). Secondly, we leverage this goodput calculation to design a forwarding strategy that seamlessly adapts itself to changing network conditions (i.e., mobility, connectivity) and provides superior performance.*

Our first goal is to design algorithms that determine the overall goodput based on network characteristics such as connectivity and mobility, given that a subset of nodes operate as routers, while others operate as flooders. This problem is challenging primarily due to two reasons - *i)* flooding can result in duplicate copies of a packet reaching the destination along multiple overlapping paths, *ii)* when there are multiple flows in a general network, nodes can forward traffic to each other, thereby

influencing their incoming and outgoing rates. For networks with a single flow and feed forward networks with multiple flows where we only encounter the first challenge, we adopt a Bayesian network model and perform exact inference to determine the goodput. For general networks with multiple flows, the second challenge complicates exact inference and therefore, we propose a fixed-point approximation to determine the overall network goodput.

Having determined the overall network goodput for a given set of routers and flooders, our next objective is to leverage this calculation to design a forwarding strategy that classifies individual nodes as routers or flooders to maximize the overall network goodput. Our work in driven by the intuition that in a wireless network with heterogeneous mobility and connectivity characteristics, neither routing nor flooding alone may perform well. Despite its apparent simplicity, this router/flooder classification is a challenging problem. 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 operating as a flooder may affect the usefulness of turning another node into a flooder, implying subtle dependencies in the decision process.

Our contributions in this paper are as follows.

- We demonstrate via an example of a four node network that determining the goodput exactly, even for a network with a single flow, is non-trivial. We construct a Bayesian network and design an algorithm that leverages the sum-product algorithm to infer the exact goodput for a network with a single flow. We extend the Bayesian network model for inferring the exact goodput to feed forward networks with multiple flows. For a general network with multiple flows, the problem becomes more challenging as node pairs can forward traffic to one another resulting in cyclical dependencies. For this scenario, we propose a fixed-point approximation for determining the goodput.

- We leverage the goodput calculation to design a forwarding strategy that determines which nodes should operate as routers and which ones should operate as flooders so as to maximize overall network goodput. Our approach *adaptive-flood* assumes that an underlying native routing protocol is available and greedily selects those nodes as candidate flooders that maximize the overall network goodput. The algorithm picks nodes as flooders in decreasing order in which they contribute to maximizing network-wide goodput; it stops when converting any of the remaining routers into a flooder would result in a decrease in goodput. From an implementation perspective, 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.

- We show via simulation that *adaptive flood* outperforms network-wide routing or flooding. In particular, at low network loads flooding outperforms routing, while at high network loads, performance is reversed. In contrast, *adaptive flood* matches or outperforms both approaches over most or all of the range of loads in both homogeneous as well as heterogeneous scenarios. From these results, we conclude that *i)* the proposed fixed-point algorithm is an effective way for determining the overall network goodput, and *ii)* 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 discuss related work in Section 2. We formalize the problem and the underlying network model in Section 3. We first demonstrate the inherent challenges involved in exact goodput calculation via a simple example and then propose a Bayesian network model based algorithm for exact inference in Section 4. For general networks with multiple flows where exact inference is difficult, we propose a simple fixed-point approximation technique for determining the goodput in Section 5. We describe the *adaptive-flood* algorithm for classifying nodes into routers and flooders and present simulation results evaluating the performance of *adaptive-flood* in Section 6. We conclude the paper and provide an outlook at future work in Section 7.

2. Related Work

In this section, we list some of the most influential work studying the capacity of wireless networks and highlight how this paper differs from existing work. A significant amount of past research has been devoted to demonstrate the capacity scaling of both flat and hybrid mobile networks [1, 2, 14, 15]. Gamal *et al.* study the delay-throughput scaling in mobile wireless networks [16]. In comparison to research on throughput analysis and measurement, there is minimal work on goodput in wireless networks [17, 18, 19]. In contrast to prior work, we propose algorithms for computing the goodput (and not throughput) of flows in wireless networks. We also utilize this goodput calculation to design a forwarding strategy that adapts seamlessly to changing network conditions.

We next present work related to network classification in wireless networks. Several past research efforts [9, 20, 21] have addressed the challenge of classifying mobile wireless networks based on connectivity and predictability. For example, Manfredi *et al.* [9] propose 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 network based on connectivity and unpredictability so as to maximize goodput. In contrast to [9], where classification has been done for the network as a whole, the *adaptive-flood* algorithm developed in this paper adopts a per-node classification strategy (route or flood) in order to maximize goodput.

Additionally, 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 [4, 22, 23, 24, 25]. Epidemic routing

Notation	Definition
V	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
$\mathbb{N}$	list of N_i, $\forall i \in V$
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 1: Notation

[5, 26] and multicopy routing [27] are designed for sparsely-connected networks and use a store-carry-and-forward mechanism and packet replication to battle poor connectivity. [23, 28] make assumptions on the mobility pattern and network topology to design forwarding protocols for intermittently connected networks. Tie *et al.* [12] propose 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). A survey of different forwarding strategies designed for mobile wireless networks is available in [29, 30]. None of this past research, however, investigates the question related to which nodes should flood/route in a network with time-varying connectivity, which is the cornerstone of our forwarding strategy.

3. Network Model

We consider a network with $|V|$ nodes. Let $\mathscr{F}$ be the set of flows in the network. The source and destination for any flow f is denoted by f_s and f_d respectively. Let us consider two nodes i and j and p_{ij} as the probability of successfully transmitting a packet from node i to node j. The link quality can vary both due to the wireless channel and mobility of the nodes, 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 represent the network as a graph $G(V, E)$ where E denotes the set of all edges in the graph; an edge e_{ij} exists if $p_{ij} > 0$.

We assume that the network has a native routing algorithm and H_{if} denotes the next hop neighbor for node i for flow f, as obtained by the routing algorithm. We consider a simple case where each node in the network can operate either as a router or a flooder, but cannot perform both actions preferentially based on the destination of the packet. We assume there are F flooders in the network. 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 node i (node j is said to be a neighbor of node i if $p_{ij} > 0$). If node 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.

We assume that time is divided into slots and packet transmissions take place at the beginning of each time slot. We assume that node i transmits data at a rate of C_i packets per time slot. Therefore, all outgoing links from node i can carry data at the maximum rate of C_i. C_i and p_{ij} together capture the capacity constraint for the link e_{ij}. Φ_{ij} is the amount of traffic originating at node i and destined for node j; Φ is the corresponding traffic matrix. A summary of our notation is available in Table I.

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,

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

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 assume that $\mathbb{E}[R_i]$ in a_i is calculated over some large interval of time.

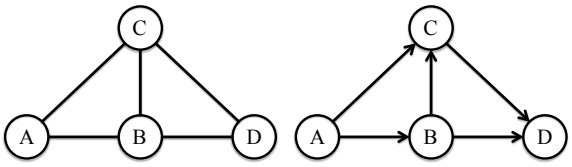
We define the *goodput* of a flow as the number of unique packets received at the destination for the flow per time slot. The overall network goodput is thus the sum of goodput for the different flows. Our first goal in this paper is to design algorithms to determine the goodput of flows in a heterogeneous network. We then use this goodput calculation to design forwarding strategies for these networks.

4. Exact Goodput Calculation

In this section, we construct a Bayesian network model to determine the exact goodput of flows in a mobile wireless network. We start with a simple example of a single flow in a four node network to help the reader appreciate the need for a Bayesian network for computing the goodput. We then proceed to the more complicated scenarios of a single flow in a general network and multiple flows in feed forward networks, design algorithms to construct a Bayesian network and then use the sum-product algorithm to determine the exact goodput.

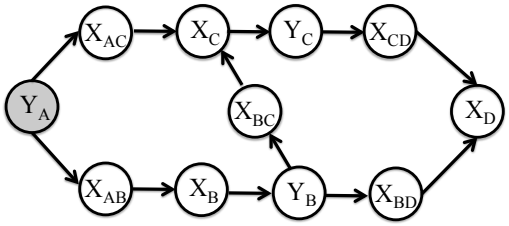
4.1. Simple Four Node Network

We first consider a simple four node network as shown in Figure 1(a) with a single flow f from A to D. We assume that all links have the same success probability (i.e., $p_{ij} = p$). We assume that the shortest path from A to D is (A, B, D). For sake of simplicity, we assume $\Phi_{AD} = C_A$. We next determine the exact goodput (in expectation) for this flow for three different



(a) Four Node Network (b) DAG for Four Node Network

Figure 1: Four Node Network and DAG construction



(a) Bayesian Network

Y_C	$P[X_{CD}=0\|Y_C]$	$P[X_{CD}=1\|Y_C]$
0	1	0
1	(1-p)	p

p is known apriori

(b) X_{CD}: conditional probability table

X_C	$P[Y_C=0\|X_C]$	$P[Y_C=1\|X_C]$
0	1	0
1	$(1-a_c)$	a_c

a_c is determined at runtime

(c) Y_C: conditional probability table

Figure 2: Bayesian Network for Four Node Network

scenarios. This example will demonstrate that determining the goodput exactly is non-trivial and help appreciate the complexity of this problem. The methods outlined here serve as building blocks for determining the exact goodput.

Scenario 1: All nodes are routers

A sends packets at a rate C_A. The expected number of packets reaching B in a time period T is pC_AT. The rate-limiter at B allows packets according to equation 1 and is given by $a_B = \min(1, \frac{C_B}{pC_A})$. The goodput ($\mathscr{G}_f$) is given by,

$$\mathscr{G}_f = C_A p^2 a_B \tag{2}$$

Scenario 2: A is a flooder; B, C, D are routers

In this scenario, A has two independent paths to D. Therefore, one needs take into account the fact that the packet might reach D along either or both paths, (A, B, D) and (A, C, D). In this scenario, $a_B = \min(1, \frac{C_B}{pC_A})$ and $a_C = \min(1, \frac{C_C}{pC_A})$. Therefore, the goodput $\mathscr{G}_f$ is given by,

$$\mathscr{G}_f = C_A(1 - (1 - p^2 a_B)(1 - p^2 a_C)) \tag{3}$$

Scenario 3: A, B are flooders; C, D are routers

In this scenario, both A and B operate as flooders. Therefore, A sends packets to both B and C. B being a flooder also broadcasts packets, that can be received by C and D. Figure 1(b) shows the directed acyclic graph (DAG) traversed by the flow originating from A and destined to D. In this case, as C can receive packets from both A and B, the probability of C receiving a packet via A is p and via B is $p^2 a_B$. Hence $a_C = \min(1, \frac{C_C}{C_A(1-(1-p^2 a_B)(1-p))})$.

Determining the exact packet reaching probability at D is more involved in comparison to the previous scenarios, because the probability of receiving the packet from B and C are not independent.

For ease of analysis, we introduce a few variables. Let X_i be the indicator variable denoting that a packet reaches node i. Let Y_i be the indicator variable denoting that the packet is available at the outgoing interface of node i. X_i and Y_i might be different because node i drops the incoming packets probabilistically. Further, let X_{ij} be the indicator variable denoting that the packet is successfully transmitted over link e_{ij}. All indicator variables are 1 if the packet is available and 0 otherwise. The probability of the packet reaching D is $P[X_D|Y_A]$. Determining $P[X_D|Y_A]$ can be treated as an inference problem by constructing a Bayesian network comprising of the indicator variables X_i, Y_i and X_{ij} as vertices.

Figure 2(a) shows the Bayesian network with the above mentioned random variables. It is clear that each node i (except source and destination) in Figure 1(b) is represented by two vertices X_i and Y_i, while each edge e_{ij} is represented by vertex X_{ij} in Figure 2(a). Note that as node A is transmitting a packet, Y_A is observed. The Bayesian network elegantly captures the interdependence between the different indicator variables. For example, given X_{AC} and X_{BC}, X_C is conditionally independent of all the other nodes in the network, i.e., given whether a packet got through links e_{AC} and e_{BC}, the packet being received by C is independent of all the other random variables in the network. Using Figure 2(a), $P[X_D|Y_A]$ can be inferred from equation (4), where $\mathbb{Z}$ is the set of all the random variables. As Y_A is observed, $P[Y_A] = 1$ and thus $P[X_D, Y_A] = P[X_D|Y_A]$. Equation (4) is derived using the classic sum-product algorithm [31].

There is a subtle, but important point that we illustrate using Figures 2(b) and 2(c). The Bayesian network captures two kinds of dependencies, one where the conditional probabilities are known apriori and another where the probabilities have to inferred using equation 1. The values in the conditional probability tables of nodes X_i and X_{ij} are known apriori. However, the conditional probability tables of nodes Y_i are not known apriori and have to be inferred at runtime. For example, in Figure 2(b), X_{CD}'s conditional probability table can be easily determined. But Y_C's conditional probability table (Figure 2(c)) is dependent on a_c. The value of a_c is governed by equation 1 and can only be determined once $P[X_C|Y_A]$ has been inferred. Therefore, the probabilities have to be inferred in topological order, thereby making this approach amiable to the sum-product algorithm.

We note that the sum-product algorithm is applied in stages. For example, since the conditional probability table for Y_C has missing entries, to solve $P[X_D|Y_A]$ exactly and determine the goodput, we first apply the sum-product algorithm to infer $P[X_C|Y_A]$ and then using this, infer $P[Y_C|X_C]$.

$$\begin{aligned} P[X_D|Y_A] = \sum_{(\mathbb{Z}-Y_A-X_D)} & P[X_D|X_{BD}, X_{CD}]P[X_{CD}|Y_C]P[X_{BD}|Y_B] \\ & P[Y_C|X_C]P[X_C|X_{AC}, X_{BC}]P[X_{BC}|Y_B] \\ & P[Y_B|X_B]P[X_B|X_{AB}]P[X_{AB}|Y_A]P[X_{AC}|Y_A] \end{aligned} \tag{4}$$

The goodput $\mathscr{G}_f = C_A \mathrm{P}[X_D|Y_A]$ and is given by

$$\mathscr{G}_f = C_A(p^5 a_B a_C - 3p^4 a_B a_C + p^3 a_B a_C + p^2(a_B + a_C)) \quad (5)$$

From this simple example, it is clear that due to the heavy dependence of Y_i on X_i, the probability of a packet reaching the destination and thus the goodput is dependent on a large number of random variables. To determine the packet reaching probability at a node, it is necessary to express the joint probability distribution as a product of conditional probabilities and then sum over the variables that are not of interest.

4.2. Single Flow in a General Network

In this subsection, we extend the approach described in the previous subsection to a general network with a single flow. From the simple four node example, we observe that when one or more nodes operate as flooders, the set of nodes and links traversed by a given flow's packets form a directed acyclic graph (DAG) instead of 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.

We first determine the DAG $G_f(V_f, E_f)$ traversed by flow f from $G(V, E)$. We propose an algorithm to construct the Bayesian network G'_f from G_f and then use the sum-product algorithm for exact inference to determine the goodput $\mathscr{G}_f$. A simple description of the inputs and steps needed to calculate the exact goodput are given in the beginning of Algorithm 1.

The DAG construction function *determine-DAG()* in Algorithm 1 maintains two lists: the *explored* list V_f and the *to-be-explored* list S_f. For flow f, DAG construction begins from source f_s. Initially S_f contains only f_s while V_f is empty. The while loop then iterates until S_f is empty. At every iteration of the while loop, the node s at the head of S_f is considered (in the first iteration the node is f_s). Recall that there are F flooders in the network. Therefore, s can be either a flooder or a router. Depending on the role of s, the algorithm either adds each neighbor or next hop neighbor to the to-be-explored list, provided it is not already in the to-be-explored or explored lists. The algorithm also adds the corresponding edges to E_f. Nodes are checked before being added to the to-be-explored list to avoid loops in the DAG. The function finally returns the DAG $G_f(V_f, E_f)$.

The next step is to construct the Bayesian network $G'_f(V'_f, E'_f)$ from G_f. The *construct-Bayes-Net()* function outlines the different steps for constructing the Bayesian network. The algorithm first iterates through V_f and for each vertex i in V_f (apart from f_s and f_d), adds vertices X_i and Y_i in V'_f. For f_s and f_d, the algorithm adds Y_{f_s} and X_{f_d} respectively. The algorithm also adds the edge $e_{X_i Y_i}$ to E'_f. The algorithm then iterates through E_f and for each edge e_{ij} adds vertex X_{ij} to V'_f and edges $e_{Y_i X_{ij}}$ and $e_{X_{ij} X_j}$ to E'_f. Let $\mathbb{X}$ and $\mathbb{Y}$ denote the sets containing the vertices X_i and Y_i in the Bayesian network respectively.

Having constructed the Bayesian network, the final step is to determine the exact goodput of the flow. Our goal is to infer the $\mathrm{P}[X_{f_d}|Y_{f_s}]$ from G'_f. Note that Y_{f_s} is observed. We first perform a topological sort on the graph G'_f. For a DAG, the topological

Algorithm 1 Single flow: exact goodput calculation

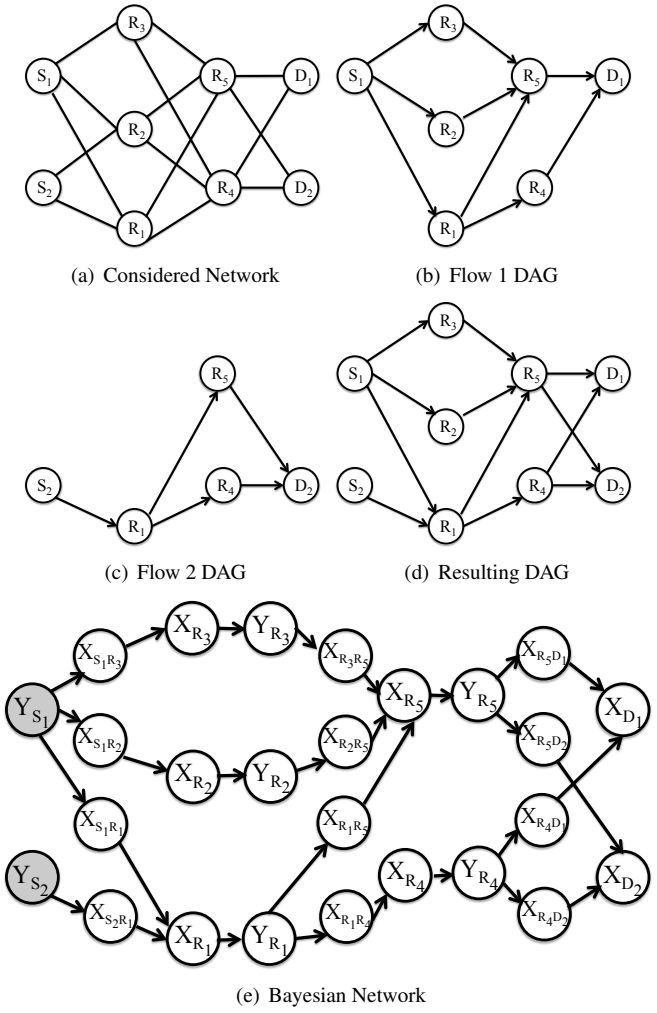
Input: Source-destination of flow, set of routers and flooders, network graph, list of neighbors, router forwarding tables, link success probability matrix, traffic matrix
Output: Exact goodput of the flow

Step 1: Determine the DAG traversed by the flow
Step 2: Construct Bayesian network from the DAG
Step 3: Topological sort the Bayesian network
Step 4: Determine goodput via exact inference on the Bayesian network by iteratively applying sum-product algorithm

function G_f = *determine-DAG*($f, F, G, \mathbb{N}, \mathbb{H}$)
  $S_f = [f_s]$, $V_f = [\,]$, $E_f = [\,]$
  while $S_f \neq [\,]$ **do**
    $[s] = head(S_f)$
    if $s \neq f_d$ **then**
      $V_f = V_f + [s]$
      if $s \in F$ **then**
        for all $i \in N_s$ **do**
          if $i \notin V_f$ **then**
            $E_f = E_f + e_{si}$
            if $i \notin S_f$ **then**
              $S_f = S_f + [i]$
      else
        if $H_{sf} \notin V_f$ **then**
          $E_f = E_f + e_{sH_{sf}}$
          if $H_{sf} \notin S_f$ **then**
            $S_f = S_f + [H_{sf}]$
    $S_f = S_f - [s]$
  return $G_f(V_f, E_f)$

function G'_f = *construct-Bayes-Net*(G_f)
  $V'_f = [\,]$, $E'_f = [\,]$
  for all $i \in V_f$ **do**
    if $[i] == f_s$ **then**
      $V'_f = V'_f + Y_{f_s}$
    else if $[i] == f_d$ **then**
      $V'_f = V'_f + X_{f_d}$
    else
      $V'_f = V'_f + X_i + Y_i$
      $E'_f = E'_f + e_{X_i Y_i}$
  for all $e_{ij} \in E_f$ **do**
    $V'_f = V'_f + X_{ij}$
    $E'_f = E'_f + e_{Y_i X_{ij}} + e_{X_{ij} X_j}$
  return $G'_f(V'_f, E'_f)$

function $\mathscr{G}_f$ = *single-flow-exact-goodput*(G'_f, $\mathbb{P}$, Φ)
  U'_f = *topological-sort*(G'_f)
  for all $u \in U'_f$ **do**
    if $u \in \mathbb{X}$ **then**
      Determine $\mathrm{P}[X_i|Y_{f_s}]$ by sum-product algorithm {*u denoted by* X_i}
      $a_i = \min(1, \frac{C_i}{\Phi_{f_s f_d} \mathrm{P}[X_i|Y_{f_s}]})$
    else if $u \in \mathbb{Y}$ **then**
      $\mathrm{P}[Y_i|X_i] = a_i$ {*u denoted by* Y_i}
    else
      $\mathrm{P}[X_{ij}|Y_i] = p_{ij}$ {*u denoted by* X_{ij}}
  $\mathscr{G}_f = \Phi_{f_s f_d} \mathrm{P}[X_{f_d}|Y_{f_s}]$
  return $\mathscr{G}_f$



(a) Considered Network (b) Flow 1 DAG

(c) Flow 2 DAG (d) Resulting DAG

(e) Bayesian Network

Figure 3: Bayesian Network Construction for a Feed Forward Network

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. Y_{f_s} and X_{f_d} are the first and last nodes in this ordering U'_f. As noted earlier, $\mathrm{P}[Y_i|X_i] = a_i$ can only be determined after $\mathrm{P}[X_i|Y_{f_s}]$ is inferred. This is possible as Y_i always appears after X_i in the topological ordering. The topological traversal of nodes thus makes the sum product algorithm the ideal candidate for inferring $\mathrm{P}[X_{f_d}|Y_{f_s}]$.

4.3. Feed Forward Networks with Multiple Flows

In the previous subsection, we studied the single flow case and observed that flooding can result in the same packet traversing multiple overlapping paths to the destination. For the multiple flow case, the problem is compounded by the fact that nodes can forward traffic to one another, resulting in interdependencies. In this subsection, we demonstrate how the exact goodput calculation algorithm developed for the single flow case can be extended for the special case of feed forward networks with multiple flows. We investigate the multiple flow general network case in Section 5.

In a feed forward network, the network is arranged in the form of layers and all flows move in one direction (i.e., say from left to right). If there are M layers in the network, we assume that all sources are in layer 1 and all destinations are in layer M. Additionally, we assume that nodes in layer k can only receive data packets from nodes in layer $(k-1)$ and can send data packets to nodes in layer $(k+1)$.

We consider the example of a simple four-layer feed forward network (Figure 3(a)) to illustrate the goodput calculation. In this figure, there are two flows, one from S_1 to D_1 and the other from S_2 to D_2. We assume that nodes S_1 and R_1 act as flooders, while all other nodes act as routers. It is evident that similar to the single flow case, the path traversed by each flow is a DAG. The DAGs traversed by flow 1 and flow 2 are shown in Figures 3(b) and 3(c) respectively. The *determine-DAG()* function in Algorithm 1 can be used to determine the DAGs for the individual flows.

An additional step is required for feed forward networks, which is to combine the DAGs traversed by the individual flows. We observe from Figures 3(a), 3(b) and 3(c) that as all flows are moving in the same direction and as nodes in layer $(k-1)$ can only send data packets to nodes in layer k, a situation where a pair of nodes forward packets to each other cannot arise. Therefore, the resulting graph obtained by merging the different DAGs together will also be a DAG. This scenario is depicted in Figure 3(d).

Once the DAG is created, one can utilize the *construct-Bayes-Net( )* function in Algorithm 1 to construct the Bayesian network. The Bayesian network for the resulting DAG is shown in Figure 3(e). However, there are some significant differences from the single flow case. The values taken by a node (i.e., X_i, Y_i, X_{ij}) will be an $\mathscr{F}$ tuple, with each item in a tuple denoting an indicator variable for a particular flow. A value of 1 for an indicator variable denotes that the packet for that flow has been received correctly, while 0 denotes otherwise. For example, in Figure 3(e), the observed nodes Y_{S_1} and Y_{S_2} will be represented by the tuples [1, 0] and [0, 1] respectively. Let p be the probability of successfully transmitting a packet over any link in Figure 3(a). Let us now consider $X_{S_1R_1}$. As S_1 is observed, therefore, $X_{S_1R_1}$ takes values [0, 0], [0, 1], [1, 0] and [1, 1] with probability $(1-p)$, 0, p and 0 respectively. Similarly, $X_{S_2R_1}$ takes values [0, 0], [0, 1], [1, 0] and [1, 1] with probability $(1-p)$, p, 0 and 0 respectively. Another major difference from the single flow case lies in the calculation of the packet-passage probability a_i. For the multiple flow case, to calculate a_i, it is essential to sum the incoming traffic from all flows. Therefore, the total incoming flow at R_1 is $p(\Phi_{S_1D_1} + \Phi_{S_2D_2})$. Hence, the value of a_{R_1} in this case is $\min\left(1, \frac{C_{R_1}}{p(\Phi_{S_1D_1}+\Phi_{S_2D_2})}\right)$.

Having constructed the Bayesian network, the next step is to determine the goodput. A function similar to *single-flow-exact-goodput( )* in Algorithm 1 can be used to perform exact inference to determine the goodput. To determine the overall network goodput, one needs to add the goodput of the individual flows.

5. General Networks: Approximate Goodput Calculation

In a general network with multiple flows, cyclical dependencies can arise from node pairs forwarding traffic to one another. For example, it is possible that some node j receives traffic from node i for some flow f_1 while node i receives traffic from node j for some other flow f_2. In this case, since a_i depends on R_i which includes traffic arriving from node j, and a_j depends on R_j which includes traffic arriving from node i, we will need to compute the packet-passage probabilities via a set of simultaneous equations to determine the exact goodput. Therefore, for general networks with multiple flows we propose a fixed-point approximation.

Before developing the fixed-point iteration, we revisit Scenario 3 for the four node network in Section 4. To determine the goodput approximately in this scenario, we apply the following assumption - probabilities of a packet reaching a node along its incoming links are independent. In Figure 1(b), the probability of the packet reaching B and C is given by $\alpha_B = p$ and $\alpha_C = 1-(1-p^2 a_B)(1-p)$ respectively. Assuming that the probability of a packet reaching D through B and C are independent, the approximate goodput at D is given by,

$$\mathscr{G}_f \approx C_A(1-(1-\alpha_B a_B p)(1-\alpha_C a_C p)) \tag{6}$$

We note that the approximate goodput for a single flow in a general network or multiple flows in feed forward networks can be easily computed by using the approach outlined above. For achieving this, the topological ordering for the DAG traversed by the flow(s) is determined. The nodes are then navigated in topological order and the packet reaching probabilities at a node are computed using the packet-passage probabilities and assuming independence of incoming links.

Algorithm 2 shows the steps of the algorithm for determining the approximate goodput for a single flow in a general network. The algorithm starts by determining the DAG G_f traversed by flow f using the *determine-DAG()* function. It then obtains a topological sort U_f from G_f. U_f thus consists of a linear ordering of vertices. The algorithm next evaluates the probability of a packet reaching the nodes in U_f in order. 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 node i receiving that packet on its incoming DAG links are independent of one another. As discussed earlier, this is clearly an approximation, since two input links at node i may share common upstream nodes in the DAG. Let α_{if} be the probability that a packet reaches node i for flow f. For any node i in U_f, let U_{if} denote the list of the nodes in U_f appearing before node i in this ordering. We approximate the probability of a packet reaching i by

$$\alpha_{if} = 1 - \prod_{j \in U_{if}} (1 - \alpha_{jf} p_{ji} a_j) \tag{7}$$

Equation (7) 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 U_f in order ensures that when the algorithm calculates α_{if} for node i, α_{jf} of all nodes j in U_{if} has already been computed.

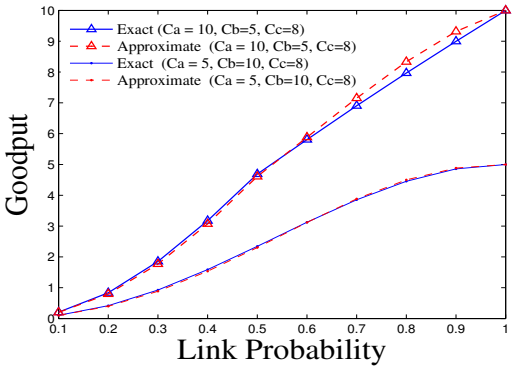


Figure 4: Comparison between exact and approximate goodput

Algorithm 2 Single Flow: Approximate Goodput Calculation

1: **function** $\mathscr{G}_f$ = *single-flow-approx-goodput*$(f, F, G, \mathbb{P}, \mathbb{N}, \mathbb{H})$
2: G_f = *determine-DAG*$(f, F, G, \mathbb{N}, \mathbb{H})$
3: U_f = *topological-sort*(G_f)
4: **for** $i \in U_f$ **do**
5: $\alpha_{if} = 1 - \prod_{j \in U_{if}} (1 - \alpha_{jf} p_{ji} a_j)$
6: $a_i = \min\{1, \frac{C_i}{\Phi_{fs f_d} \alpha_{if}}\}$
7: $\mathscr{G}_f = \Phi_{fs f_d} \alpha_{f_d f}$
8: **return** $\mathscr{G}_f$

We numerically verify the 'closeness' of the approximate approach to the exact method for the network in Figure 1(a) for different link capacities in Figure 4. We observe from the figure that the approximation is close to the exact value of the goodput for this simple scenario.

5.1. Fixed Point Approximation

In this subsection, we leverage the approach for determining the approximate goodput for a single flow for designing an algorithm for determining the goodput of a general network with multiple flows. To address the primary challenge of determining the packet-passage probabilities $\vec{a}$, we propose a fixed-point iteration. The details of the fixed-point approximation is shown in Algorithm 3.

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 a node i (I_{fi}). I_{fi} is calculated using equation 7 that uses the independence assumption between incoming links at a node. The fixed point iteration then uses the incoming rates to compute the packet-passage probabilities to be used in iteration $l+1$, $\vec{a}^l$. 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. Once the fixed-point converges, the goodput for each individual flow f is calculated. The overall goodput over all flows is then computed by summing the goodput for the individual flows in the network.

Algorithm 3 General Network: Approximate Goodput Calculation

function $\mathcal{G}_F$ = *general-approx-goodput*$(F, G, \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
  $\vec{a}^0 = 0, l = 1$
  for all $f \in \mathscr{F}$ **do**
    G_f = *calculate-DAG*$(f, F, G, \mathbb{N}, \mathbb{H})$
    U_f = *topological-sort*(G_f)
  while (true) **do**
    for all $f \in \mathscr{F}$ **do**
      for $j \in U_f$ **do**
        $\alpha_{jf} = 1 - \prod_{i \in U_j}(1 - \alpha_{if} p_{ij} a_i)$
        $I_{fj} = \Phi_{fsfd} \alpha_{jf}$
    for all $s \in V$ **do**
      $R_s = \sum_{f \in \mathscr{F}} I_{fs}$
      $a_s^l = \min\{1, \frac{C_s}{R_s}\}$
    if $\max |\vec{a}^l - \vec{a}^{l-1}| < \tau$ **then**
      return false
    $l = l + 1$
  for all $f \in \mathscr{F}$ **do**
    $\mathcal{G}_f = \Phi_{fsfd} \alpha_{fdf}$
  $\mathcal{G}_F = \sum_{f \in \mathscr{F}} \mathcal{G}_f$
  return $\mathcal{G}_F$

Algorithm 4 *adaptive-flood* router/flooder classification

function F = *adaptive-flood*$(G, \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
  $F = [\,]$
  $\mathcal{G}_F$ = *general-approx-goodput*$(F, G, \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
  while $|F| \neq |V|$ **do**
    $F' = [\,]$
    for all $s \notin F$ **do**
      $T = F + [s]$
      $\mathcal{G}_T$ = *general-approx-goodput*$(T, G, \mathbb{P}, \Phi, \mathbb{N}, \mathbb{H}, \mathscr{F}, \tau)$
      if $\mathcal{G}_T > \mathcal{G}_F$ **then**
        $F' = s, \mathcal{G}_F = \mathcal{G}_T$
    if $F' == [\,]$ **then**
      return F
    else
      $F = F + F'$
  return F

6. Application Scenario: Adaptive Forwading

In this subsection, we leverage Algorithm 3 to design a superior forwarding strategy for mobile wireless networks that improves goodput. In a heterogeneous network with varying link characteristics, neither routing nor flooding alone may perform well. *Our goal is to propose a forwarding strategy for heterogeneous mobile networks that adapts dynamically and seamlessly to changing network conditions (i.e., mobility, connectivity) and provides superior goodput.* We propose a forwarding strategy *adaptive-flood*, based on local link characteristics as well as network-wide considerations, that determines which nodes should forward traffic according to the forwarding table computed by the native routing protocol, and which nodes should broadcast their traffic to all neighbors.

Intuitively 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. We first describe *adaptive-flood* and then demonstrate its superior performance via simulation.

6.1. The adaptive-flood algorithm

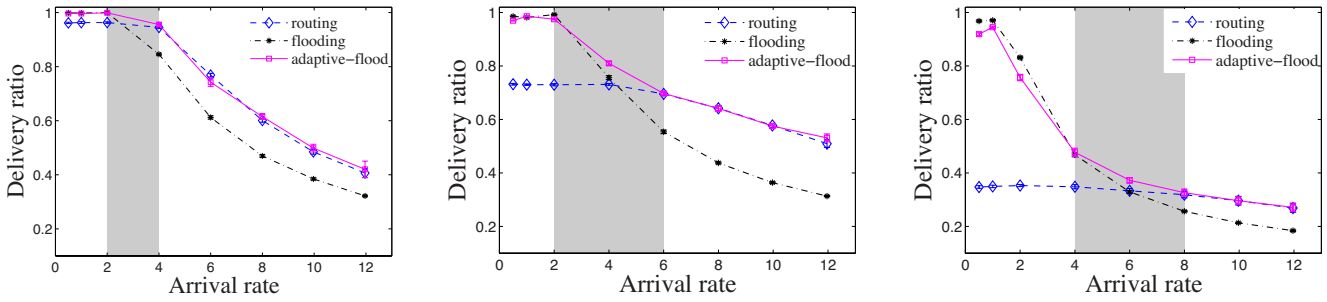
Our approach, the *adaptive-flood* algorithm is an iterative greedy algorithm that turns one router into a flooder at each iteration. In *adaptive-flood*, the router that is turned into a flooder is that router whose change in status would greedily maximize overall network goodput. This decision of converting a router to a flooder is primarily based on Algorithm 3. Algorithm 3 thus lies in the core of the *adaptive-flood* algorithm.

adaptive-flood begins with all network nodes initially classified as routers (Algorithm 4). It then computes the total (over all source/destination pairs) goodput for the $\mathscr{F}$ flows by calling the *general-approx-goodput()* function outlined in Algorithm 3. In Algorithm 4, $\mathcal{G}_F$ is the total network goodput when there are F flooders. During each iteration in *adaptive-flood*, the total goodput is calculated assuming 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 total goodput if it were to be turned into a flooder. 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 total goodput. Note that 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.

As the *adaptive-flood* algorithm is a greedy algorithm and leverages the fixed-point iteration in Algorithm 3 to determine the goodput for a given set of flooders, it converges quickly. Additionally, as *adaptive-flood* runs on top of the underlying routing algorithm, and classifies nodes as routers or flooders, it can be easily implemented in existing wireless networks with limited effort and provide superior performance. *Adaptive-flood* also does not require additional inputs in comparison to a standard link-state algorithm. We note that similar to stateful routing protocols such as OLSR, *adaptive-flood* algorithm can also be run locally given broadcast link state updates.

6.2. Simulation Results

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



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

Figure 6: Delivery ratio with different sets of flows for an 18-node network

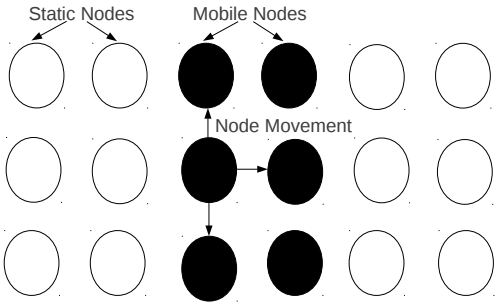


Figure 5: Topology: 18 Node Network

6.2.1. Simulation Setup

The simulator is written primarily in C++ and interfaced with MATLAB. Our simulations are conducted on a grid topology with r rows and c columns. The heterogeneous network primarily used for simulation consists of 18 nodes as shown in Figure 5 (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 this paper.

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. For example, if an epoch consists of 10 intervals and nodes i and j are in the same or adjacent grid positions in 3 of these intervals, then the value of p_{ij} is 0.3. The link success probability matrix $\mathbb{P}$ is populated at the beginning of the epoch, before Dijkstra's algorithm is executed.

As *adaptive-flood* contains both routing and flooding nodes, we compare it against two baseline approaches: pure routing and pure flooding to demonstrate its performance gains. To conduct a fair comparison, the pure routing algorithm considered here also calculates routes according to Dijkstra's algorithm. In pure flooding, each node forwards traffic to all its neighbors. Flooders perform duplicate suppression so as to not forward the same packet twice.

We make several simplifying assumptions in the simulator. We do not model the effect of interference in the network, assuming that a node can receive multiple packets in the same time slot (one along each of its incoming links); this would be possible when a node has multiple interfaces operating on different channels, when a node has multiple directional antennae, and in some CDMA settings. A node can, however, send only one packet in one time slot. If a node is a router, the packet will be received and processed only by the designated next hop; if it is a flooder, its transmitted packet can be received by all its neighbors present in adjacent grid positions. In our simulation, we model data plane forwarding only; since *adaptive-flood* and routing, all take advantage of common link state control information, we do not explicitly simulate link state transfer.

All nodes have a single finite buffer of size 300 packets, and packets arriving to a full buffer are dropped. Hence our simulator can simulate packet loss, a phenomenon that can occur at high exogenous packet arrival rates. Each data point in our simulation is obtained for the same number of total exogenous packet arrivals (15000 packets). The number of intervals is 30 and the number of slots per interval is 10. The number of epochs is adjusted so that the expected number of arrivals is 15000 exogenous packets.

We report results for the 18-node network in Figure 5 for different sets of network flows. In each case, we vary the exogenous arrival rate and study two different performance metrics: overall network goodput and delivery ratio. As discussed earlier, a flow's goodput is the average number of unique packets delivered to the destination per time slot; 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. The arrival rate determines the traffic matrix Φ used in our models. For each flow, each source node has the same probability of generating an exogenous packet arrival at the beginning of a time slot. We multiply this probability by the total number of flows to obtain the exogenous packet arrival rate. We increase the arrival rate by increasing the probability of an exogenous packet arrival. To ensure a fair comparison, in each run of the experiment, when simulating the different algorithms, the same seed values are used so that the sequence of the exogenous packet arrivals is unchanged. We report results as mean values obtained after multiple runs; the length of the error bars denotes twice the standard deviation.

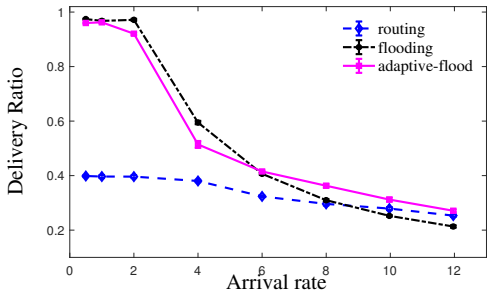
- **Scenario 1:** We consider mostly short (2-hop) flows. Every node in the static region has a single 2-hop flow destined to a randomly chosen other node in the same static region (12 flows in all). There are also 3 single-hop flows in the mobile region.
- **Scenario 2:** We have 12 flows in the static 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.

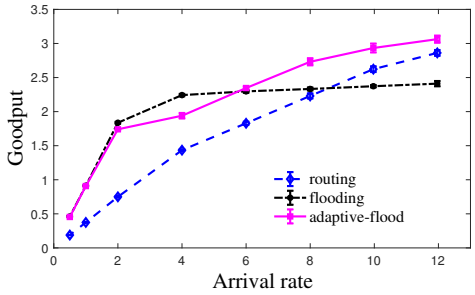
6.2.2. *Comparing routing, flooding, and adaptive-flood*

Figure 6 shows the delivery ratio of the different algorithms for the above three scenarios. For scenario 1 (Figure 6(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 6(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



(a) Delivery Ratio



(b) Goodput

Figure 7: Delivery Ratio and Goodput for a 18-node network

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 6(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.

Next, we turn our attention to the performance of our *adaptive-flood* algorithm[1]. The shaded regions in Figure 6 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 6(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 6(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 note that *adaptive-flood* also often selects stationary nodes as flooders. Turning stationary nodes into flooders can

[1] In some cases, packet-passage probabilities in the fixed point iteration of the *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.

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.

6.2.3. 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 7 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.

Comparing goodput for the different schemes (Figure 7(b)), 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.

7. Conclusion

In this paper, we investigated the problem of determining the goodput of flows in a heterogeneous mobile network. We started with the simple case of a single flow and designed a Bayesian network model that leveraged the sum-product approach for determining the exact goodput. We extended the Bayesian network model for determining the exact goodput to feed forward networks with multiple flows. For a general network with multiple flows, where nodes forward traffic to each other, we developed a fixed-point approximation to determine the goodput. We studied an application scenario where the fixed-point approximation can be utilized to design a forwarding strategy for heterogeneous mobile networks, comprising of both stable as well as highly dynamic components, in which neither uniform routing nor flooding at all network nodes performs well. We developed an iterative algorithm *adaptive-flood* that individually determines for each node whether it should operate as a router or a flooder. This decision of operating a node as router or flooder is based on considerations such as the quality of its links, the amount of traffic traversing it, and the effect of turning routers into flooders on overall goodput. Via simulation, we showed that *adaptive-flood* yields performance equivalent to, and often significantly better than, that of baseline routing or flooding alone. In future work, we plan to investigate the performance gains achievable by preferentially routing or flooding packets based on their destination.

References

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

[2] M. Grossglauser and D. Tse, "Mobility increases the capacity of ad-hoc wireless networks," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1360–1369.

[3] Y. Chen, Y. Shen, J. Zhu, X. Jiang, and H. Tokuda, "On the throughput capacity study for aloha mobile ad hoc networks," *IEEE Transactions on Communications*, vol. 64, no. 4, pp. 1646–1659, 2016.

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

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

[6] R. Rab, A. Rahman, and F. TuzZohra, "Analytical modeling of self-pruning and an improved probabilistic broadcast for wireless multihop networks," *Ad-hoc Networks*, vol. 52, 2016.

[7] U. G. Acer, S. Kalyanaraman, and A. A. Abouzeid, "Weak state routing for large-scale dynamic networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1450–1463, 2010.

[8] U. G. Acer, S. Kalyanaraman, and A. A. Abouzeid, "Weak state versus strong state: an analysis of a probabilistic state mechanism for dynamic networks," *Wireless networks*, vol. 20, no. 4, pp. 639–654, 2014.

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

[10] 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*.

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

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

[13] A. Seetharam, S. Heimlicher, J. Kuorse, and W. Wei, "Routing with adaptive flooding in heterogeneous mobile networks," in *COMSNETS*. IEEE, 2015.

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

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

[16] A. E. Gamal, J. Mammen, B. Prabhakar, and D. Shah, "Throughput-delay trade-off in wireless networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1. IEEE, 2004.

[17] L. Le, S. Albayrak, M. Elkotob, and A. C. Toker, "Improving tcp goodput in 802.11 access networks," in *Communications, 2007. ICC'07. IEEE International Conference on*. IEEE, 2007, pp. 4494–4499.

[18] D. Zhou, W. Song, and P. Ju, "Goodput improvement for multipath transport control protocol in cooperative relay-based wireless networks," *IET Communications*, vol. 8, no. 9, pp. 1541–1550, 2014.

[19] D. Qiao, S. Choi, and K. G. Shin, "Goodput analysis and link adaptation for ieee 802.11 a wireless lans," *IEEE transactions on Mobile Computing*, vol. 99, no. 4, pp. 278–292, 2002.

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

[21] 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.

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

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

[24] T. Meng, F. Wu, Z. Yang, G. Chen, and A. V. Vasilakos, "Spatial reusability-aware routing in multi-hop wireless networks," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 244–255, 2016.

[25] A. Bhorkar, M. Naghshvar, and T. Javidi, "Opportunistic routing with congestion diversity in wireless ad hoc networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 24, no. 2, pp. 1167–1180, 2016.

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

[27] K. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient routing in intermittently connected mobile networks: The multiple-copy case,"

IEEE/ACM Trans. Networking, 2008.

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

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

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

[31] D. Koller and N. Friedman, *Probabilistic Graphical Models*. The MIT Press, 2009.