Loop-Free Backpressure Routing Using Link-Reversal Algorithms

ABSTRACT

The backpressure routing policy is known to be a throughput optimal policy that supports any feasible traffic demand in data networks, but may have poor delay performance when packets traverse loops in the network. In this paper, we study loop-free backpressure routing policies that forward packets along directed acyclic graphs (DAGs) to avoid the looping problem. These policies use link reversal algorithms to improve the DAGs in order to support any achievable traffic demand.

For a network with a single commodity, we show that a DAG that supports a given traffic demand can be found after a finite number of iterations of the link-reversal process. We use this to develop a joint link-reversal and backpressure routing policy, called the loop free backpressure (LFBP) algorithm. This algorithm forwards packets on the DAG, while the DAG is dynamically updated based on the growth of the queue backlogs. We show by simulations that such a DAG-based policy improves the delay over the classical backpressure routing policy. We also propose a multicommodity version of the LFBP algorithm, and via simulation we show that its delay performance is better than that of backpressure.

1. INTRODUCTION

Throughput and delay are the two major metrics used to evaluate the performance of communication networks. For networks that exhibit high variability, such as mobile ad hoc networks, the dynamic backpressure routing policy [1] is a highly desirable solution, known to maximize throughput in a wide range of settings. However, the delay performance of backpressure is poor [2]. The high delay is attributed to a property of backpressure that allows the packets to loop within the network instead of moving towards the destination. In this paper we improve the delay performance of backpressure routing by constraining the data routing along loop free paths.

To eliminate loops in the network, we assign directions to the links such that the network becomes a directed acyclic graph (DAG). Initially, we generate an arbitrary DAG and use backparessure routing over it. If the initial DAG has max-flow smaller than the traffic demand, parts of the network become overloaded. By reversing the direction of the links that point from non-overloaded to overloaded nodes a new DAG with a lower overload is obtained. Iterating over this process, our distributed algorithm gradually converges to a DAG that supports any traffic demand feasible in the network. Hence the loop-free property is achieved without the loss of throughput.

Prior work identifies looping as a main cause for high delays in backpressure routing and proposes delay-aware backpressure techniques. Backpressure enhanced with hop count bias is first proposed in [3] to drive packets through paths with smallest hop counts when the load is low. An alternative backpressure modification that utilizes shortest path information is proposed in [8]. A different line of works proposes to learn the network topology using backpressure and then use this information to enhance routing decisions. In [7] backpressure is constrained to a subgraph which is discovered by running unconstrained backpressure for a time period and computing the average number of packets routed over each link. Learning is effectively used in scheduling [9] and utility optimization [13] for wireless networks. In our work we aim to eliminate loops by restricting backpressure to a DAG, while we dynamically improve the DAG by reversing links.

The link-reversal algorithms were introduced in [4] as a means to maintain connectivity in networks with volatile links. These distributed algorithms react to topological changes to obtain a DAG such that each node has a loop-free path to the destination. In [5], one of the link-reversal algorithms was used to design a routing protocol (called TORA) for multihop wireless networks. Although these algorithms provide loop free paths and guarantee connectivity from the nodes to the destination, they do not maximize throughput. Thus, the main goal of this paper is to create a new link-reversal algorithm and combine it with the backpressure algorithm to construct a distributed throughput optimal algorithm with improved delay performance.

The main contributions of this paper are as follows:

- For a network with single commodity, we study the lexicographic optimization of the queue growth rate. We show that the lexicographically optimal queue growth rates can be used to distributedly detect links whose change of direction reduces overload.
- We develop a novel link-reversal algorithm that re-

verses link direction based on overload conditions to form a new DAG with lexicographically smaller queue growth rates.

- We show that a combination of backpressure and link reversal can be used to find an optimal DAG. We develop loop free backpressure (LFBP) algorithm, a distributed routing scheme that eliminates loops and retains the throughput optimality property.
- Our simulation results of LFBP show a significant delay improvement over backpressure in static and dynamic networks.
- We extend the LFBP algorithm to networks with multiple commodities, and provide a simulation result to show its delay improvement over backpressure.

2. SYSTEM MODEL AND DEFINITIONS

2.1 Network model

We consider the problem of routing single-commodity data packets in a network. The network is represented by a graph $G = (N, E)$, where N is the set of nodes and E is the set of undirected links $\{i, j\}$ with capacity c_{ij}. Packets arrive at the source node s at rate λ and are destined for a receiver node d. Let $f^{\max}$ denote the maximum flow from node s to d in the network G. The quantity $f^{\max}$ is the maximally achievable throughput at the destination node d.

To avoid unnecessary routing loops, we restrict forwarding along a directed acyclic graph (DAG) embedded in the graph G. An optimal DAG exists to support the max-flow $f^{\max}$ and can be found by: (i) computing a feasible flow allocation (f_{ij}) that yields the max-flow $f^{\max}$ in G (e.g. using [11]); (ii) trimming any positive flow on directed cycles; (iii) defining an embedded DAG by assigning a direction for each link $\{i, j\}$ according to the direction of the flow f_{ij} on that link. Since backpressure achieves the max-flow of a constrained graph [14], performing backpressure routing over the optimal DAG supports λ.

This centralized approach is unsuitable for mobile adhoc networks, which are based on wireless links with time-varying capacities and may undergo frequent topology changes. In such situations, the optimal embedded DAG also changes with time, which requires constantly repeating the above offline process. Instead, it is possible to use a distributed adaptive mechanism that reverses the direction of links until a DAG that supports the current traffic demand is found. In this paper we propose an algorithm that reacts to the traffic conditions by changing the directions of some links. To understand the properties of the link-reversing operations, we first study the fluid level behavior of a network under overload conditions.

2.2 Flow equations

Consider an embedded DAG $D_k = (N_k, E_k)$ in the network graph G, where $N_k = N$ is the set of network nodes and E_k is the set of directed links.[1] For each link $\{i, j\} \in E$, either (i, j) or (j, i) belongs to E_k (but not both). Each directed link (i, j) has the capacity of the undirected counterpart $\{i, j\}$, which is c_{ij}. Let $f_k^{\max}$ be the maximum flow of the DAG D_k from the source node s to the destination node d. Any embedded DAG has smaller or equal max-flow with respect to G, $f_k^{\max} \leq f^{\max}$.

For two disjoint subsets A and B of nodes in D_k, we define $\text{cap}_k(A, B)$ as the total capacity of the directed links going from A to B, i.e.,

$$\text{cap}_k(A, B) = \sum_{(i,j)\in E_k : i\in A,\, j\in B} c_{ij}. \quad (1)$$

A cut is a partition of nodes (A, A^c) such that $s \in A$ and $d \in A^c$. A cut (A_k, A_k^c) is a min-cut if it minimizes the expression $\text{cap}_k(A_k, A_k^c)$ over all cuts. By the max-flow min-cut theorem $f_k^{\max} = \text{cap}_k(A_k, A_k^c)$, where (A_k, A_k^c) is the min-cut of the DAG D_k. We remark that a cut in a DAG is also a cut in G or another DAG. However, the value of $\text{cap}_k(.,.)$ depends on the graph considered (see summation in (1)).

We consider the network as a time-slotted system, where slot t refers to the time interval $[t, t + 1)$, $t \in \{0, 1, 2, \ldots\}$. Each network node n maintains a queue $Q_n(t)$, where $Q_n(t)$ also denotes the queue backlog at time t. We have $Q_d(t) = 0$ for all t since the destination node d does not buffer packets. Let $A(t)$ be the number of exogenous packets arriving at the source node s in slot t. Under a routing policy that forwards packets over the directed links defined by the DAG D_k, let $F_{ij}(t)$ be the number of packets that are transmitted over the directed link $(i, j) \in E_k$ in slot t; the link capacity constraint states that $F_{ij}(t) \leq c_{ij}$ for all t. The queues $Q_n(t)$ are updated over slots according to

$$Q_n(t) = Q_n(t-1) + 1_{[n=s]}A(t) + \sum_{i:(i,n)\in E_k} F_{in}(t) - \sum_{j:(n,j)\in E_k} F_{nj}(t), \quad (2)$$

where $1_{[\cdot]}$ is an indicator function.

To study the overload behavior of the system we define the queue overload (i.e., growth) rate at node n as

$$q_n = \lim_{t\to\infty} \frac{Q_n(t)}{t}. \quad (3)$$

Additionally, define the exogenous packet arrival rate λ and the flow f_{ij} over a directed link (i, j) as

$$\lambda = \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1} A(\tau), \quad f_{ij} = \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1} F_{ij}(\tau),$$

where the above limits are assumed to exist almost surely. Taking the time average of (2) and letting $t \to \infty$, we have the fluid-level equation:

$$q_n = 1_{[n=s]}\lambda + \sum_{i:(i,n)\in E_k} f_{in} - \sum_{j:(n,j)\in E_k} f_{nj},\ \forall n \in N \quad (4)$$

$$0 \leq f_{ij} \leq c_{ij},\ \forall (i, j) \in E_k. \quad (5)$$

Equations (4) and (5) are the flow conservation and link capacity constraints, respectively. A network node n is said to be overloaded if its queue growth rate q_n is positive, which implies that $Q_n(t) \to \infty$ as $t \to \infty$ (see (3) and [10]). Summing (4) over $n \in N$ yields

$$\sum_{n\in N} q_n = \lambda - \sum_{i:(i,d)\in E_k} f_{id}, \quad (6)$$

[1]The notation D_k of an embedded DAG is useful in the paper; it will denote the DAG that is formed after the kth iteration of the link-reversal algorithm.

where $\sum_{i:(i,d)\in E_k} f_{id}$ denotes the throughput received at the destination d. Therefore, equation (6) states that the received throughput is equal to the exogenous arrival rate λ less the sum of queue growth rates $\sum_{n\in N} q_n$ in the network.

2.3 Properties of queue overload vector

If the traffic arrival rate λ is strictly larger than the maximum flow $f_k^{\max}$ of the DAG D_k, then some network nodes will be overloaded. It is because, from (6), we have

$$\sum_{n\in N} q_n = \lambda - \sum_{i:(i,d)\in E_k} f_{id} \geq \lambda - f_k^{\max} > 0, \qquad (7)$$

which implies that $q_n > 0$ for some node $n \in N$. Let $\boldsymbol{q} = (q_n)_{n\in N}$ be the queue overload vector. A queue overload vector $\boldsymbol{q}$ is feasible in the DAG D_k if there exist overload rates $(q_n)_{n\in N}$ and flow variables $(f_{ij})_{(i,j)\in E_k}$ that satisfy (4) and (5). Let $\mathcal{Q}_k$ be the set of all feasible queue overload vectors in D_k. We are interested in the *lexicographically smallest* queue overflow vector in set $\mathcal{Q}_k$. Formally, given a vector $\boldsymbol{u} = (u_1, \ldots, u_N)$, let $\overline{u}_i$ be the ith maximal component of $\boldsymbol{u}$. We say that a vector $\boldsymbol{u}$ is *lexicographically smaller* than a vector $\boldsymbol{v}$, denoted by $\boldsymbol{u} <_{\text{lex}} \boldsymbol{v}$, if $\overline{u}_1 < \overline{v}_1$ or $\overline{u}_i = \overline{v}_i$ for all $i = 1, ..., (j-1)$ and $\overline{u}_j < \overline{v}_j$ for some $j = 2, \ldots, N$. If $\overline{u}_i = \overline{v}_i$ for all i, then the two vectors are lexicographically equal, represented by $\boldsymbol{u} =_{\text{lex}} \boldsymbol{v}$.[2] The above-defined vector comparison induces a total order on the set $\mathcal{Q}_k$, and hence the existence of a lexicographically smallest vector is always guaranteed [12].

LEMMA 1 ([6]). *Let $\boldsymbol{q}_k^{min}$ be the lexicographically smallest vector in the queue overload region $\mathcal{Q}_k$ of the DAG D_k. We have the following properties:*

1. *The vector $\boldsymbol{q}_k^{min}$ exists and is unique in the set $\mathcal{Q}_k$ (Lemma 5 in [6]).*
2. *The vector $\boldsymbol{q}_k^{min}$ minimizes the sum of queue overload rates, i.e., it is the solution to the optimization problem:*

$$\text{minimize } \sum_{n\in N} q_n, \text{ subject to } \boldsymbol{q} \in \mathcal{Q}_k$$

(direct consequence of Theorem 1 in [6]). Due to (6), the corresponding throughput at the destination node d is maximized.

3. *A feasible flow allocation vector $(f_{ij})_{(i,j)\in E_k}$ induces $\boldsymbol{q}_k^{min}$ if and only if over each link $(i,j) \in E_k$ the following holds:*

$$\text{if } q_i < q_j, \text{ then } f_{ij} = 0, \qquad (8)$$

$$\text{if } q_i > q_j, \text{ then } f_{ij} = c_{ij} \qquad (9)$$

(Lemma 5 in [6]).

In general, there are many flow allocations that yield the maximum throughput. Focusing on those that additionally induce $\boldsymbol{q}_k^{\min}$ has two advantages. First, as shown next, these allocations lead to link-reversal operations that improve the max-flow of the DAG D_k. Second, the backpressure algorithm can be used to preform the same reversals and improve the max-flow; we will use this observation in Section 4 to combine link-reversal algorithms with backpressure routing.

[2]As an example, the two vectors $\boldsymbol{u} = (3,2,1,2,1)$ and $\boldsymbol{v} = (1,2,3,2,2)$ satisfy $\boldsymbol{u} <_{\text{lex}} \boldsymbol{v}$ because $\overline{u}_1 = \overline{v}_1 = 3$, $\overline{u}_2 = \overline{v}_2 = \overline{u}_3 = \overline{v}_3 = 2$, and $\overline{u}_4 = 1 < \overline{v}_4 = 2$.

3. LINK-REVERSAL ALGORITHMS

The link-reversal algorithms given in [4] were designed to maintain a path from each node in the network to the destination. One algorithm relevant to this paper is the *full reversal method*. This algorithm is triggered when some nodes $n \neq d$ lose all of their outgoing links. At every iteration of the algorithm, nodes n, that have no outgoing link, reverse the direction of all their incoming links. This process is repeated until all the nodes other than the destination have at least one outgoing link. When the process stops these nodes are guaranteed to have a path to the destination. The example in Figure 1, taken from [4], illustrates this algorithm at work.

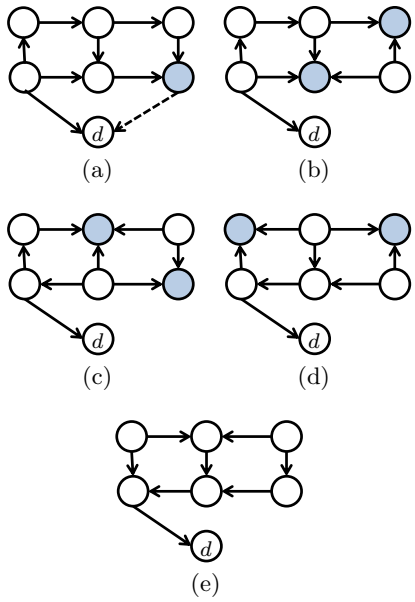


Figure 1: Illustration of the full reversal method of [4] when the dashed link in Figure 1(a) is lost. At every iteration, the algorithm reverses all the links incident to the nodes with no outgoing link (the blue nodes).

Although the full reversal algorithm guarantees connectivity, the resulting throughput may be significantly lower than the maximum possible. Hence, in this paper we shift the focus from connectivity to maximum throughput. Specifically, we propose a novel link-reversal algorithm that produces a DAG which supports the traffic demand λ, assuming $\lambda \leq f^{\max}$. We do this by quickly constructing an initial DAG and improving upon it in multiple iterations.

3.1 Initial DAG

We assume that each node in the network has a unique ID. These IDs give a topological ordering to the nodes. So, the initial DAG can be created simply by directing each link to go from the node with the lower ID to the node with the higher ID. If the unique IDs are not available, the initial DAG can be created by using a strategy such as the one given in [5].

3.2 Overload detection

Given a DAG D_k, $k = 0, 1, 2, \ldots$, we suppose that there is

a routing policy π that yields the lexicographically minimal queue overload vector $\boldsymbol{q}_k^{\min}$. [3] Then we use the vector $\boldsymbol{q}_k^{\min}$ to detect node overload and decide whether a link should be reversed.

If the data arrival rate λ is less than or equal to the maximum flow $f_k^{\max}$ of the DAG D_k, then there exists a flow allocation (f_{ij}) that supports the traffic demand and yields zero queue overload rates $q_n = 0$ at all nodes $n \in N$. By the second property of Lemma 1 and nonnegativity of the overload vector, the queue overload vector $\boldsymbol{q}_k^{\min}$ is zero. Thus, the throughput under policy π is λ according to (6), and the current DAG D_k supports λ; no link-reversal operations are needed.

On the other hand, if the arrival rate λ is strictly larger than the maximum flow $f_k^{\max}$, by the second property in Lemma 1 the maximum throughput is $f_k^{\max}$ and the queue overload vector $\boldsymbol{q}_k^{\min} = (q_{k,n}^{\min})_{n\in N}$ is nonzero because we have from (7) that

$$\sum_{n\in N} q_{k,n}^{\min} > \lambda - f_k^{\max} > 0.$$

We may therefore detect the event "DAG D_k supports λ" by testing whether the overload vector $\boldsymbol{q}_k^{\min}$ is zero or non-zero.

The next lemma shows that if DAG D_k does not support λ then it contains at least one under-utilized link (our link-reversal algorithm will reverse the direction of such links to improve network throughput).

LEMMA 2. *Suppose that the traffic demand λ satisfies*

$$f_k^{\max} < \lambda \le f^{\max}.$$

where $f_k^{\max}$ is the max-flow of the DAG D_k and $f^{\max}$ is the max-flow of the undirected network G. Then there exists a link $(i,j) \in E_k$ such that $q_{k,i}^{min} = 0$ and $q_{k,j}^{min} > 0$.

PROOF OF LEMMA 2. Let A_k be the set of overloaded nodes under a flow allocation that induces the lexicographically minimal overload vector $\boldsymbol{q}_k^{\min}$ in the DAG D_k; the set A_k is nonempty due to $\lambda > f_k^{\max}$ and (7). It follows that the partition (A_k, A_k^c) is a min-cut of D_k (see Lemma 7 in the Appendix).[4] By the max-flow min-cut theorem, the capacity of the min-cut (A_k, A_k^c) in D_k satisfies $\mathrm{cap}_k(A_k, A_k^c) = f_k^{\max} < f^{\max}$.

The proof is by contradiction. Let us assume that there is no directed link that goes from the set A_k^c to A_k in the DAG D_k. It follows that $\mathrm{cap}_k(A_k, A_k^c)$ is the sum of capacities of all undirected links between the sets A_k and A_k^c, i.e.,

$$\mathrm{cap}_k(A_k, A_k^c) = \sum_{i\in A_k,\, j\notin A_k} c_{ij},$$

which is equal to the value of the cut (A_k, A_k^c) in graph G. Since the value of any cut is larger or equal to the min-cut, applying the max-flow min-cut theorem on G we have

$$f^{\max} \le \sum_{i\in A_k,\, j\notin A_k} c_{ij} = \mathrm{cap}_k(A_k, A_k^c) = f_k^{\max},$$

which contradicts the assumption that $f_k^{\max} < \lambda \le f^{\max}$. □

[3] In Section 4, we will develop an algorithm using backpressure that does not require the computation of the lexicographically optimal overload vector. We use this vector only to prove the properties of our link-reversal algorithm.

[4] The set A_k^c contains the destination node d and is nonempty.

3.3 Link reversal

Lemma 2 shows that if the DAG D_k has insufficient capacity to support the traffic demand $\lambda \le f^{\max}$, then there exists a directed link from an underloaded node i to an overloaded one j under the lexicographically minimum overflow vector $\boldsymbol{q}_k^{\min}$. Because of property (8), we may infer that this link is not utilized. Next we show that reversing the direction of this link provides a strictly improved DAG.

We consider the link-reversal algorithm (Algorithm 1) that reverses all such links that satisfy the property in Lemma 2. This reversal yields a new directed graph $D_{k+1} = (N, E_{k+1})$.

Algorithm 1 Link-Reversal Algorithm

```
for all (i,j) ∈ E_k do
    if q_{k,i}^min = 0 and q_{k,j}^min > 0 then
        (j,i) ∈ E_{k+1}
    else
        (i,j) ∈ E_{k+1}
    end if
end for
```

LEMMA 3. *The directed graph D_{k+1} is acyclic.*

PROOF OF LEMMA 3. Recall that A_k is the set of overloaded nodes in the DAG D_k under the lexicographically minimum queue overload vector $\boldsymbol{q}_k^{\min}$. Let $L_k \subseteq E$ be the set of undirected links between A_k and A_k^c. Algorithm 1 changes the link direction in a subset of L_k. More precisely, it enforces the direction of all links in L_k to go from A_k to A_k^c.

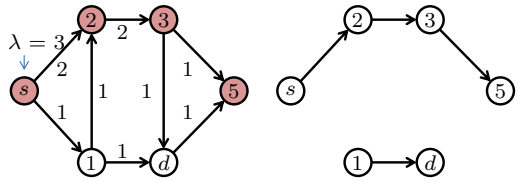
We complete the proof by construction in two steps. First, we remove all links in L_k from the DAG D_k, resulting in two disconnected subgraphs that are DAGs themselves. Second, consider that we add a link in L_k back to the network with the direction going from A_k to A_k^c. This link addition does not create a cycle because there is no path from A_k^c to A_k, and the resulting graph remains to be a DAG. We can add the other links in L_k one-by-one back to the graph with the direction from A_k to A_k^c; similarly, these link additions do not create cycles. The final directed graph is D_{k+1}, and it is a DAG. See Fig. 2 for an illustration. □

The next lemma shows that the new DAG D_{k+1} supports a lexicographically *smaller* optimal overload vector (and therefore potentially better throughput) than the DAG D_k.

LEMMA 4. *Let D_k be a DAG with the maximum flow $f_k^{\max} < \lambda \le f^{\max}$. The DAG D_{k+1}, obtained by performing Algorithm 1 over D_k, has the lexicographically minimum queue overload vector satisfying $\boldsymbol{q}_{k+1}^{min} <_{\mathrm{lex}} \boldsymbol{q}_k^{min}$.*

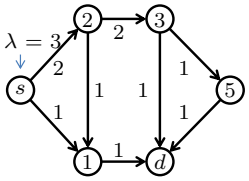
PROOF OF LEMMA 4. Consider a link $(a,b) \in E_k$ such that $q_{k,a}^{\min} = 0$ and $q_{k,b}^{\min} > 0$; this link exists by Lemma 2. From the property (8), any feasible flow allocation (f_{ij}) that yields the lexicographically minimum overload vector $\boldsymbol{q}_k^{\min}$ must have $f_{ab} = 0$ over link (a,b). The link-reversal algorithm reverses the link (a,b) so that $(b,a) \in E_{k+1}$ in the DAG D_{k+1}. Consider the following feasible flow allocation (f'_{ij}) on the DAG D_{k+1}:

$$f'_{ij} = \begin{cases} \epsilon & \text{if } (i,j) = (b,a) \\ 0 = f_{ji} & \text{if } (i,j) \neq (b,a) \text{ but } (j,i) \text{ is reversed} \\ f_{ij} & \text{if } (i,j) \text{ is not reversed} \end{cases}$$



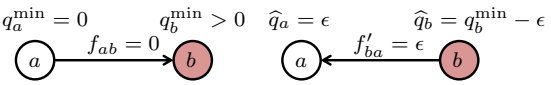
(a) The DAG D_k with $A_k = \{s, 2, 3, 5\}$.

(b) Two disconnected DAGs formed by removing all links between A_k and A_k^c.



(c) The DAG D_{k+1} formed by adding all links in L_k back to the graph with the direction going from A_k to A_k^c.

Figure 2: Illustration for the proof of Lemma 3.



(a) Link (a, b) before the link reversal.

(b) Link (b, a) after the link reversal.

Figure 3: A link $\{a, b\}$ in the network in Fig. 2 before and after link reversal. Before the reversal, the flow f_{ab} is zero on (a, b). After the reversal, an ϵ flow can be sent over (b, a) so that $(\widehat{q}_a, \widehat{q}_b) <_{\text{lex}} (q_{k,a}^{\min}, q_{k,b}^{\min})$, while the rest of the flow allocation remains the same.

where $\epsilon < q_{k,b}^{\min}$ is a sufficiently small value. In other words, the flow allocation (f'_{ij}) is formed by reversing links and keeping the previous flow allocation (f_{ij}) except that we forward an ϵ-amount of overload traffic from node b to a. Let $\widehat{\boldsymbol{q}} = (\widehat{q}_n)_{n \in N}$ be the resulting queue overload vector. We have

$$\widehat{q}_b = q_{k,b}^{\min} - \epsilon < q_{k,b}^{\min},\ \widehat{q}_a = \epsilon > q_{k,a}^{\min} = 0, \text{ and}$$
$$\widehat{q}_n = q_{k,n}^{\min},\ n \notin \{a, b\}.$$

Therefore, $\widehat{\boldsymbol{q}} <_{\text{lex}} \boldsymbol{q}_k^{\min}$ (see Fig. 3 for an illustration). Let $\boldsymbol{q}_{k+1}^{\min}$ be the lexicographically minimal overload vector in D_{k+1}. It follows that $\boldsymbol{q}_{k+1}^{\min} \leq_{\text{lex}} \widehat{\boldsymbol{q}} <_{\text{lex}} \boldsymbol{q}_k^{\min}$, completing the proof. □

THEOREM 1. *Suppose the traffic demand is feasible in G, i.e., $\lambda \leq f^{\max}$, and the routing policy induces the overload vector $\boldsymbol{q}_k^{\min}$ at every iteration k. Then, the link-reversal algorithm will find a DAG whose maximum flow supports λ in a finite number of iterations.*

PROOF OF THEOREM 1. The link-reversal algorithm creates a sequence of DAGs $\{D_0, D_1, D_2, \ldots\}$ in which a strict improvement in the lexicographically minimal overload vector is made after each iteration, i.e.,

$$\boldsymbol{q}_0^{\min} >_{\text{lex}} \boldsymbol{q}_1^{\min} >_{\text{lex}} \boldsymbol{q}_2^{\min} >_{\text{lex}} \cdots.$$

The lexicographically minimal overload vector is unique in a DAG by Lemma 1, the DAGs $\{D_0, D_1, D_2, \ldots\}$ must all be distinct. Since there are a finite number of unique DAGs in the network, the link-reversal algorithm will find a DAG D_{k^*} that has the lexicographically minimal overload vector $\boldsymbol{q}_{k^*}^{\min} = \mathbf{0}$ and the maximum flow $f_{k^*}^{\max} \geq \lambda$ in a finite number of iterations; this DAG D_{k^*} exists because the undirected graph G has the maximum flow $f^{\max} \geq \lambda$. □

3.4 Arrivals outside stability region

We show that even when $\lambda > f^{\max}$, the link reversal algorithm will stop reversing the links in a finite number of iterations, and it will obtain the DAG that supports the maximum throughput $f^{\max}$. We begin by examining the termination condition of our algorithm and show that if the algorithm stops at iteration k (which pappens when there is no link to reverse), then the DAG D_k supports the max-flow of the network.

LEMMA 5. *Consider the situation when $\lambda > f_k^{\max}$. If there is no link (i, j) such that $q_{k,i}^{\min} = 0$ and $q_{k,j}^{\min} > 0$, then $f^{\max} = f_k^{\max}$ and $\lambda > f^{\max}$. That is, if there are no links to reverse at iteration k, and $\mathbf{q}_k^{\min} > 0$, then the throughput of D_k is equal to $f^{\max}$.*

PROOF. Let A_k be the set of overloaded nodes under a flow allocation that induces the lexicographically minimal overload vector $\mathbf{q}_k^{\min}$ in the DAG D_k. We know that (A_k, A_k^c) is a min-cut of the network from Lemma 7 (in the appendix), so

$$cap_k(A_k, A_k^c) = f_k^{\max}.$$

Suppose the link reversal algorithm stops after iteration k, i.e. at iteration k there are no links to reverse. In this situation, there is no link (i, j) such that $q_{k,i}^{\min} = 0$ and $q_{k,j}^{\min} > 0$, so by property (9), all the links between A_k and A_k^c go from A_k to A_k^c. The capacity of the cut (A_k, A_k^c) is given by

$$cap_k(A_k, A_k^c) = \sum_{i \in A_k, j \in A_k^c} c_{ij}.$$

This is equal to the capacity of the cut (A_k, A_k^c) in the undirected network G. So $f^{\max} \leq cap_k(A_k, A_k^c) = f_k^{\max}$. Because $f_{\max}^k$ cannot be greater than f^{max}, $f_{\max}^k = f^{\max}$. By assumption $\lambda > f_k^{\max}$, so $\lambda > f^{\max}$. □

When $\lambda > f^{\max}$, this lemma shows that the link reversal algorithm stops only when the DAG achieves the maximum throughput of the network. Hence, if the DAG doesn't support the maximum throughput, then there exists a link that can be reversed. After each reversal, Lemma 3 holds, so the directed graph obtained after the reversal is acyclic. We can modify Lemma 4 to show that every reversal produces a DAG that supports an improved lexicographically optimal overload vector. We can combine these results to prove the following theorem.

THEOREM 2. *Suppose the traffic demand is not feasible in G, i.e., $\lambda > f^{\max}$, and the routing policy induces the overload vector $\boldsymbol{q}_k^{\min}$ at every iteration k. Then, the link-reversal algorithm will find a DAG whose maximum flow supports f^{max} in a finite number of iterations.*

4. DISTRIBUTED DYNAMIC ALGORITHM

In the previous sections we developed a link reversal algorithm based on the assumption that we had a routing policy that lexicographically minimized the overload vector $\mathbf{q}_k^{\min}$. The algorithm reversed all the links that went from the set of all the non-overloaded nodes A_k^c to the set of overloaded nodes A_k. We showed that repeating this process for some iterations results in a DAG that supports the arrival rate λ.

The goal of this paper is to develop a link reversal algorithm based on backpressure. To achieve this goal, we develop a threshold based algorithm that identifies the cut (A_k, A_k^c) using the queue backlog information of backpressure. We can use this cut to perform the link reversals without computing the lexicographically minimum overload vector. Because this algorithm generates the same sequence of DAGs as the link reversal algorithm described in Section 3, all the previous theorems hold, and it will obtain the DAG that supports the arrival rate λ (when possible). We will call this algorithm the loop free backpressure (LFBP) algorithm.

We begin by creating an initial DAG D_0 using the method presented in Section 3.1. Then, we use the backpressure algorithm to route the packets from the source to the destination over D_0. Let $Q_n(t)$ be the queue length at node n in slot t. The backpressure algorithm can be written as in Algorithm 2. It simply sends packets on a link (i, j) if node i has more packets than j.

Algorithm 2 Backpressure algorithm (BP)

```
for all (i,j) ∈ E_k do
    if Q_i(t) > Q_j(t) then
        Transmit min{c_ij, Q_i(t)} packets from i to j
    end if
end for
```

Since backpressure is throughput optimal [1], if the arrival rate is less than $f_0^{\max}$, then all queues are stable. If the arrival rate is larger than $f_0^{\max}$, the system is unstable and the queue length grows at some nodes. In this case, the next lemma shows that if we were using a routing policy that produced the optimal overload vector $\mathbf{q}_k^{\min}$, the set of all the overloaded nodes A_k and the non-overloaded nodes A_k^c form the smallest min-cut of the DAG D_k.

DEFINITION 1. *We define the smallest min-cut (X^*, X^{*c}) in the DAG D_k as the min-cut with the smallest number of nodes in the source side of the cut, i.e., (X^*, X^{*c}) solves*

$$\begin{aligned} &\textit{minimize: } |X| \\ &\textit{subject to: } (X, X^c) \textit{ is a min-cut of } D_k. \end{aligned}$$

LEMMA 6. *Let A_k the set of overloaded nodes under a flow allocation (f_{ij}) that induces the lexicographically minimum overload vector in the DAG D_k. If $|A_k| > 0$, then (A_k, A_k^c) is the unique smallest min-cut in D_k.*

PROOF OF LEMMA 6. The proof is in Appendix B. □

Essentially, at every iteration, the link reversal algorithm of Section 3 discovers the smallest min-cut (A_k, A_k^c) of the DAG D_k and reverses the links that go from A_k^c to A_k. Now the following theorem shows that the backpressure algorithm can be augmented with some thresholds to identify the smallest min-cut.

THEOREM 3. *Assume that (A_k, A_k^c) is the smallest min-cut for DAG D_k with a cut capacity of $f_k^{\max} = cap(A_k, A_k^c) < \lambda$. If packets are routed using the backpressure routing algorithm, then there exist finite constants T and R such that the following happens:*

1. *For some $t < T$, $Q_n(t) > R$ for all $n \in A_k$, and*
2. *For all t, $Q_n(t) < R$ for $n \in A_k^c$.*

PROOF. We will prove the two claims separately. To prove the first claim we will use the fact that the network is overloaded and bottlenecked at the cut (A_k, A_k^c). We will prove the second claim using the fact that the number of packets that arrive into A_k^c in each time-slot is upper-bounded by $f_k^{\max}$, and any cut in the network has a capacity larger than or equal to f_k^{max}. The detailed proofs for both claims are given in the Appendix C. □

In LFBP, each node n implements a threshold-based smallest min-cut detection mechanism. When we start using a particular DAG D_k, in each time-slot, we check whether the queue crosses a prespecified threshold R_k. Any queue that crosses the threshold gets marked as overloaded. After using the DAG D_k for T_k timeslots, all the nodes that have their queue marked overloaded form the set A_k. When the time T_k and threshold R_k are large enough, the cut (A_k, A_k^c) is the smallest min-cut as proven in Theorem 3.

After determining the smallest min-cut, an individual node can perform a link reversal by comparing its queue's overload status with its neighbor's. All the links that go from a non-overloaded node to an overloaded node are reversed to obtain D_1. By Lemma 3 we know that D_1 is also a DAG, and by Lemma 4 D_1 lexicographically improves the overload vector. By iterating over the above steps, Theorem 1 guarantees that this algorithm will eventually result in a DAG that supports λ. The complete loop free backpressure algorithm iterations are given by Algorithm 3. This algorithm requires only local coordination between neighbors, and hence LFBP is distributed.

Algorithm 3 LFBP (Executed by node n)

```
Input: sequences {T_k}, {R_k}, unique ID n
Generate initial DAG D_0 by directing each link {n,j}
    to (n,j) if n < j, to (j,n) if j > n.
Mark the queue Q_n as not overloaded
Initialize t ← 0, k ← 0
while true do
    Use BP to send/recive packets on all links of node n
    if (Q_n(t) > R_k) then
        Mark Q_n as overloaded.
    end if
    t ← t + 1

    T_k ← T_k − 1
    if T_k = 0 then
        Reverse all links (j,n) such that Q_j is not over-
    loaded and Q_n is overloaded.
        k ← k + 1
        Mark Q_n as not overloaded
    end if
end while
```

Because this algorithm is based on thresholds, in practice, there is a possibility that the identified cut might not be the

smallest-min cut. However, Lemma 3 can be generalized to show that for any partitioning of a DAG (A, A^c), reversing the links from A^c to A keeps the graph acyclic. That is, any graph resulting from a reversal based on a false smallest min-cut is also a DAG. So, if the subsequent iterations use the correct smallest min-cuts, the algorithm will eventually obtain a DAG that supports the arrival rate λ.

4.1 Algorithm modification for topology changes

In this section we consider networks with time-varying topologies, where several links of graph G may appear or disappear over time. Although the DAG that supports λ depends on the topology of G, our proposed policy LFBP can adapt to the topology changes and efficiently track the optimal solution.Additionally, the loop free structure of a DAG is preserved under link removals. Thus, if some of the links in the network disappear, we may continue using LFBP on the new network.

To handle the appearance of new links in the network smoothly, we will slightly extend LFBP to guarantee the loop free structure. For a DAG D_k, every node n stores a unique state $x_n(k)$ representing its position in the topological ordering of the DAG D_k. The states are maintained such that they are unique and all the links go from a node with the lower state to a node with the higher state. When a new link $\{i, j\}$ appears we can set its direction to go from i to j if $x_i(k) < x_j(k)$ and from j to i otherwise. Since this assignment of direction to the new link is in alignment with the existing links in the DAG, the loop-free property is preserved.

The state for each node n can be initialized using the unique node ID during the initial DAG creation, i.e. $x_n(0) = n$. Then whenever a reversal is performed the state of node n can be updated as follows:

$$x_n(k) = \begin{cases} x_n(k-1) - 2^k\Delta, & \text{if } n \text{ is overloaded,} \\ x_n(k-1), & \text{otherwise.} \end{cases}$$

Here, Δ is some constant chosen such that $\Delta > \max_{i,j\in N} x_i(0) - x_j(0)$. Note that this assignment of state is consistent with the way the link directions are assigned by the link reversal algorithm. The states for the non-overloaded nodes are unchanged, so the links between these nodes are unaffected. Also, the states for all the overloaded nodes are decreased by the same amount $2^k\Delta$, so the direction of the links between the overloaded nodes is also preserved. Furthermore, the quantity $-2^k\Delta$ is less than the lowest possible state before the kth iteration, so the overloaded nodes have a lower state than the non-overloaded nodes. Hence, the links between the overloaded and non-overloaded nodes go from the overloaded nodes to the non-overloaded nodes.

In this scheme, the states x_n decrease unboundedly as more reversals are preformed. In order to prevent this, after a certain number of reversals, we can rescale the states by dividing them by a large positive number. This decreases the value of the state while maintaining the topological ordering of the DAG. The number of reversals k can be reset to 0, and a new Δ can be chosen such that it is greater than the largest difference between the rescaled states.

5. COMPLEXITY ANALYSIS

To understand the number of iteration the link-reversal algorithm takes to obtain the optimal DAG, we analyze the time complexity of the algorithm.

THEOREM 4. *Let C be a vector of the capacities of all the links in E, and let I be the set of indices $1, 2, ..., |E|$. Define $\delta > 0$ to be the smallest positive difference between the capacity of any two cuts. Specifically, δ is the solution of the following optimization problem*

$$\min_{A,B\subseteq I} \sum_{a\in A} c_a - \sum_{b\in B} c_b$$
$$\text{subject to: } \sum_{a\in A} c_a > \sum_{b\in B} c_b.$$

The number of iterations taken by the link reversal algorithm before it stops is upper bounded by $\lceil |N| \frac{f^{\max}}{\delta} \rceil$, where $f^{\max}$ is the max-flow of the undirected network.

PROOF. From Lemma 8, after each iteration either the max-flow of the DAG increases, or the max-flow stays the same and the number of nodes in the source side of the smallest min-cut increases. We can bound the number of consecutive iterations such that there is no improvement in the max-flow. In particular, every such iteration will add at least one node to the source set. So, it is impossible to have more than $|N| - 2$ such iteration. Hence, every $|N|$ iterations we are guaranteed to have at least one increase in the max-flow.

Max-flow is equal to the min-cut capacity, and min-cut capacity is defined as the sum of link capacities. Say, the max-flow of DAG D_{k+1} is greater than that of D_k. Let A be the set of indices (in the capacity vector C) of the links in the min-cut of D_{k+1} , and B be the set of indices of the links in the min-cut of D_k. This choice of A and B forms a feasible solution to the optimization problem given in the theorem statement. Since the optimal solution δ lower bounds all the feasible solutions in the minimization problem, the increase in the max-flow must be greater than or equal to δ.

Every $|N|$ iteration the max-flow increases at least by δ. Hence, the DAG supporting the max-flow $f^{\max}$ is formed within $\lceil |N| f^{\max}/\delta \rceil$ iterations. □

COROLLARY 1. *In a network where all the link capacities are rational with the least common denominator $\mathcal{D} \in \mathbb{N}$, the number of iterations is upper bounded by $(|N|\mathcal{D}f^{\max})$.*

PROOF. Since the capacities are rational we can write the capacity of the i^{th} link as $c_i = \frac{\mathcal{N}_i}{\mathcal{D}}$, where $\mathcal{N}_i$ is a natural number. From the definition of δ in Theorem 4, we get δ to be the value of the following optimization problem:

$$\min_{A,B\subseteq I} \frac{1}{\mathcal{D}} \left(\sum_{a\in A} \mathcal{N}_a - \sum_{b\in B} \mathcal{N}_b \right)$$
$$\text{subject to: } \sum_{a\in A} \mathcal{N}_a > \sum_{b\in B} \mathcal{N}_b.$$

All the $\mathcal{N}_{(.)}$ are integers, so to satisfy the constraint we must have the difference $\sum_{a\in A} \mathcal{N}_a - \sum_{b\in B} \mathcal{N}_b \geq 1$. Hence $\delta \geq \frac{1}{\mathcal{D}}$. Using this value of δ in Theorem 4, we can see that the number of iterations is upper bounded by $(|N|\mathcal{D}f^{\max})$. □

COROLLARY 2. *In a network with unit capacity links, the number of iterations the link-reversal algorithm takes to obtain the optimal DAG is upper bounded by $|N||E|$.*

PROOF. Using the definition of δ in Theorem 4, we get $\delta = 1$. The max-flow $f^{\max} \leq |E|$. So, by Theorem 4, the number of iterations is upper bounded by $|N||E|$. □

We conjecture that these upper bounds are not tight, and finding a tighter bound will be pursued in the future research. We simulated the link reversal algorithm in 50,000 different Erdos-Renyi networks ($p = 0.5$) of sizes 10 to 50 with randomly assigned link capacities. The link reversal algorithm started with a random initial DAG. We found that it took less than 2 iterations on average to find the optimal DAG.

A worst case lower bound for the number of iteration is $|N|$. This lower bound can be achieved in a line network where the initial DAG has all of its links in the wrong direction.

6. SIMULATION RESULTS

We compare the delay performance of the LFBP algorithm and the BP algorithm via simulations. We will see that the network with the LFBP routing has a smaller backlog on average under the same load. This shows that the LFBP algorithm has a better delay performance. We consider two types of networks for the simulations: a simple network with fixed topology, and a network with grid topology where the links appear and disappear randomly.

6.1 Fixed topology

We consider a network with the topology shown in Figure 4(a). The edge labels represent the link capacities. The undirected network has the maximum throughput of 15 packets per time slot. Figure 4(b) shows the initial DAG D_0. Instead of running the initial DAG algorithm of Section 3.1, here we choose a zero throughput DAG to test the worst-case performance of LFBP. The arrivals to the network are Poisson with rate $\lambda = 15\rho$, where we vary $\rho = .5, .55, ..., .95$. For the LFBP algorithm, we set the overload detection threshold to $R_k = 60$ for all n, k. To choose this parameter, we observed that the backlog buildup in normal operation rarely raises above 60. We also choose the detection period $T_1 = 150$ and $T_k = 50$ for all $k > 1$. This provides enough time for buildup, which improve the accuracy of the overload detection mechanism.

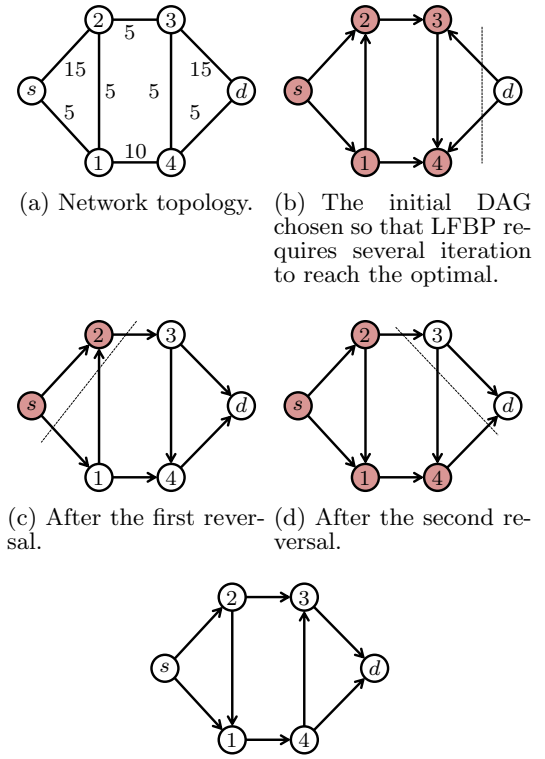
We simulate both algorithms for one million slots, using the same arrival process sample path. Figures 4(c) - 4(e) show the various DAGs that are formed by the LFBP algorithm at iterations $k = 1, 2, 3$. We can see that the nodes in the smallest min-cut get overloaded and the link reversals gradually improve the DAG until the throughput optimal DAG is reached.

Figure 5 compares the total average backlog in the network for BP and LFBP, which is indicative of the average delay. A significant delay improvement is achieved by LFBP, for example at load 0.5 the average delay is reduced by 66% We observe that the gain in the delay performance is more pronounced when the load is low. In low load situations, the network doesn't have enough "pressure" to drive the packets to the destination and so under BP the packets go in loops.

6.2 Randomly changing topology

To understand the delay performance of the LFBP algorithm on networks with randomly changing topology, we consider a network where 16 nodes are arranged in a 4×4 grid. All the links are taken to be of capacity six. For the LFBP algorithm, we choose a random initial DAG with zero throughput shown in Figure 6. The source is on the upper left corner (node 1) and the destination is on the bottom right (node 16).

In the beginning of the simulations all 24 network links are activated. At each time slot an active link fails with a probability 10^{-4} and an inactive link is activated with a probability 10^{-3}. The maximum throughput of the undirected network without any link failures is 12. Clearly on average, each link is "on" a fraction $\frac{10}{11}$ of the time, and



(a) Network topology.

(b) The initial DAG chosen so that LFBP requires several iteration to reach the optimal.

(c) After the first reversal.

(d) After the second reversal.

(e) The throughput optimal DAG.

Figure 4: Figure (a) depicts the original network. Figures (b)-(e) are the various stages of the DAG. The red nodes represent the overloaded nodes, and the dashed line shows the boundary of the overloaded and the non-overloaded nodes.

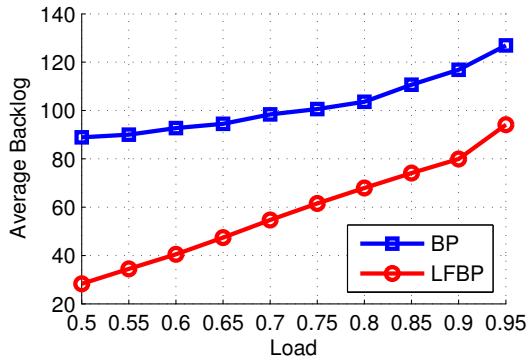


Figure 5: Average backlog in the network (Fig. 4(a)) with fixed topology for the Loop Free Backpressure (LFBP) and the Backpressure (BP) algorithms.

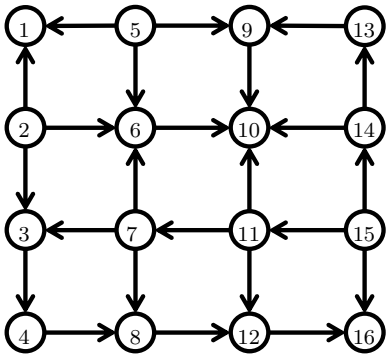


Figure 6: Initial DAG for the LFBP algorithm chosen so that the LFBP needs several iterations to reach the optimal DAG. All the links have capacity six.

thus the average maximum throughput of the undirected network with these link failure rates is $\frac{10}{11} \times 12 = 10.9$. The arrivals to the networks are Poisson with rate $\lambda = 10.9\rho$, where $\rho = .1, .2, ..., .6$. For the LFBP algorithm, the detection threshold is set to $R_k = 100$ and the detection period is $T_k = 30$ for all n, k. These parameters were chosen so that there are several reversals before a topology change occurs in the undirected network. The simulation was carried out for a million slots.

Figure 7 compares the average backlog of LFBP and BP. In the low load scenarios LFBP reduces delay significantly (by 85% for load = 0.1) even though the topology changes challenge the convergence of the link-reversal algorithm. As the load increases, both the algorithms begin to obtain a similar delay performance.

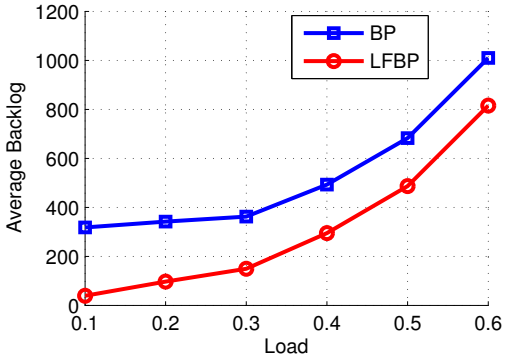


Figure 7: Average backlog in the network with random link failures (Fig. 6) for the Loop Free Backpressure algorithm and the Backpressure algorithm.

7. MULTICOMMODITY SIMULATION

We extend of the link reversal algorithm to the networks with multiple commodities. The multi-commodity algorithm is identical to the single commodity algorithm, with the exception that we now use the multicommodity backpressure of [1]. Each node n maintains a queue $Q_n^y(t)$ for each commodity y. Each commodity is assigned its own initial DAG. A pseudocode for the multicommodity LFBP that we used is given in Algorithm 4.

Algorithm 4 Multicommodity LFBP (Executed by node n)

Input: sequences $\{T_k\}, \{R_k\}$, unique ID n
For each commodity y, generate initial DAG D_0^y by directing $\{n, j\}$ to (n, j) if $n < j$, to (j, n) if $j > n$.
Mark all queues Q_n^y as not overloaded
Initialize $t \leftarrow 0$, $k \leftarrow 0$
while true **do**
    Use Multicommodity BP to send/recive packets on all links of node n
    for all y **do**
        if $(Q_n^y(t) > R_k)$ **then**
            Mark this Q_n^y as overloaded.
        end if
    end for
    $t \leftarrow t + 1$

    $T_k \leftarrow T_k - 1$
    if $T_k = 0$ **then**
        for all y **do**
            Reverse links (j, n) if Q_j^y is not overloaded and Q_n^y is overloaded.
        end for
        $k \leftarrow k + 1$
        Mark all queues as not overloaded
    end if
end while

For the simulation, we consider a network arranged in a 4×4 grid as shown in Figure 6. Each link has a capacity of 6 packets per time-slot. There are three commodities in the network defined by the source destination pairs (1,16), (4,13) and (5,8). For the LFBP algorithm, each commodity starts with the same initial DAG given in Figure 6.

We use the arrival rate vector $\lambda^{\max} = [7.18, 6.96, 9.86]$, which is a max-flow vector for this network computed by solving a linear program. We scale this vector by various load factors ρ ranging from 0.1 to 0.9. The arrivals for each commodity i is Poisson with rate $\rho\lambda_i^{\max}$. In the beginning of the LFBP simulation, $\lfloor 500/\rho \rfloor$ dummy packets are added to the source of each commodity. This is helpful in low load cases because it forces the algorithm to find a DAG with high throughput, and avoids stopping at a DAG that only supports the given (low) load. R_k was chosen to be 50 and $T_k = 50$ for all $k > 0$. The simulation was executed for 500,000 time-steps.

Figure 8 shows the average backlog in the network for different loads under backpressure and multicommodity LFBP. We can see that the LFBP algorithm has a significantly improved delay performance compared to backpressure.

8. CONCLUSION

Backpressure routing and link reversal algorithms have been separately proposed for mobile wireless networks applications. In this paper we show that these two distributed schemes can be successfully combined to yield good throughput and delay performance.We develop the Loop-Free Backpressure Algorithm which jointly routes packets in a constrained DAG and reverses the links of the DAG to improve its throughput. We show that the algorithm ultimately re-

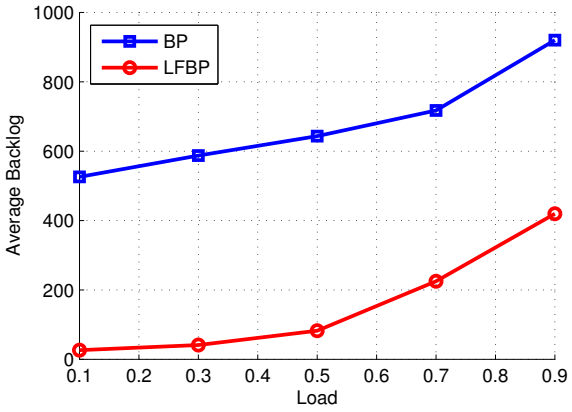


Figure 8: Average backlog in a multicommodity network with fixed topology for LFBP and BP algorithms.

sults in a DAG that yields the maximum throughput. Additionally, by restricting the routing to this DAG we eliminate loops, thus reducing the average delay. Future investigations involve optimization of the overload detection parameters and studying the performance of the scheme on the networks with multiple commodities.

9. REFERENCES

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936-1949, December 1992.

[2] L. X. Bui, R. Srikant and A. Stolyar, "A novel architecture for reduction of delay and queueing structure complexity in the back-pressure algorithm," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1597-1609, December 2011.

[3] M. J. Neely, E. Modiano and C. E. Rohrs, "Dynamic power allocation and routing for time varying wireless networks," *IEEE Journal on Selected Areas in Communications*, Special Issue on Wireless Ad-hoc Networks, vol. 23, no. 1, pp. 89-103, January 2005.

[4] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on Communications*, vol. 29, no. 1, pp. 11-18, January 1981.

[5] V.D. Park and M.S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," INFOCOM, 1997.

[6] L. Georgiadis and L. Tassiulas, "Optimal overload response in sensor networks." *IEEE Transactions on Information Theory*, vol.52, no. 6, pp. 2684-2696, June 2006.

[7] H. Xiong, R. Li, A. Eryilmaz and E. Ekici, "Delay-aware cross-layer design for network utility maximization in multi-hop networks." *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 5, pp. 951-959, May 2011.

[8] L. Ying, S. Shakkottai, A. Reddy and S. Liu, "On combining shortest-path and backpressure routing over multihop wireless networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 841-854, June 2011.

[9] P.-K. Huang, X. Lin, and C.-C. Wang, "A low-complexity congestion control and scheduling algorithm for multihop wireless networks with order-optimal per-flow delay," *IEEE/ACM Trans. on Networking*, vol. 21, no. 2, pp. 2588-2596, April 2013.

[10] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," Morgan & Claypool, 2010.

[11] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, 8: 399, 1956.

[12] L. Georgiadis, P. Georgatsos, K. Floros, and S. Sartzetakis, "Lexicographically optimal balanced networks," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 818-829, December 2002.

[13] L. Huang and M. J. Neely, "Delay reduction via Lagrange multipliers in stochastic network optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 4, pp. 842-857, April 2011.

[14] L. Georgiadis, M. J. Neely and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," Foundations and trends in networking, Now Publishers Inc, 2006.

APPENDIX

A. LEMMA 7

LEMMA 7. *Consider a DAG D_k with source node s, destination node d, and arrival rate λ. Let A_k be the set of overloaded nodes under the flow allocation (f_{ij}) that yields the lexicographically minimum overload vector. If $|A_k| > 0$, then (A_k, A_k^c) is a min-cut of the DAG D_k.*

PROOF PROOF OF LEMMA 7. First we show that (A_k, A_k^c) is a cut, i.e., the source node $s \in A_k$ and the destination node $d \in A_k^c$. The destination node d has zero queue overload rate $q_d = 0$ because it does not buffer packets; hence $d \in A_k^c$. We show $s \in A_k$ by contradiction. Assume $s \notin A_k$. The property (8) shows that there is no flow going from A_k^c to A_k, i.e.,

$$\sum_{(i,j)\in E_k:\, i\in A_k^c,\, j\in A_k} f_{ij} = 0.$$

The flow conservation equation applied to the collection A_k of nodes yields

$$\begin{aligned}\sum_{n\in A_k} q_n &= \sum_{(i,n)\in E_k:\, i\in A_k^c,\, n\in A_k} f_{in} - \sum_{(n,j)\in E_k:\, n\in A_k,\, j\in A_k^c} f_{nj} \\ &= -\sum_{(n,j)\in E_k:\, n\in A_k, j\in A_k^c} f_{nj} \le 0,\end{aligned}$$

which contradicts the assumption that the network is overloaded (i.e., $|A_k| > 0$). Note that in the above equation λ does not appear because of the premise $s \notin A_k$.

By the max-flow min-cut theorem, it remains to show that the capacity of the cut (A_k, A_k^c) is equal to the maximum flow $f_k^{\max}$ of the DAG D_k. Under the flow allocation (f_{ij}) that induces the lexicographically minimal overload vector, the throughput of the destination node d is the maximum

flow $f_k^{\max}$ (see Lemma 1). It follows that

$$f_k^{\max} = \lambda - \sum_{i \in N} q_i = \lambda - \sum_{i \in A_k} q_i \tag{10}$$

$$= \sum_{(i,j) \in E_k:\, i \in A_k,\, j \in A_k^c} f_{ij} \tag{11}$$

$$= \sum_{(i,j) \in E_k:\, i \in A_k, j \in A_k^c} c_{ij} = \mathrm{cap}_k(A_k, A_k^c). \tag{12}$$

where (10) uses (7) and $q_i = 0$ for all nodes $i \notin A_k$, (11) follows the flow conservation law over the node set A_k, and (12) uses the property (9) in Lemma 1. □

B. PROOF OF LEMMA 6

Proof of Lemma 6. Lemma 7 shows that (A_k, A_k^c) is a min cut of the DAG D_k. It suffices to prove that if there exists another min-cut (B, B^c), i.e., $A_k \neq B$ and $\mathrm{cap}_k(A_k, A_k^c) = \mathrm{cap}_k(B, B^c)$, then $A_k \subset B$. The proof is by contradiction. Let us assume that there exists another min-cut (B, B^c) such that $A_k \not\subset B$. We have the source node $s \in A_k \cap B$ and the destination node $d \in A_k^c \cap B^c$. Consider the partition $\{C, D, E, F\}$ of the network nodes such that $C = A_k \cap B$, $D = A_k \backslash B$, $E = B \backslash A_k$ and $F = N \backslash (A_k \cup B)$ (see Fig. 9). Since $A_k \not\subset B$ and $A_k \neq B$, we have $|D| > 0$. Also, we have $s \in C$ and $d \in F$. Let (f_{ij}) be a flow allocation that yields the lexicographically minimum overload vector in D_k. Properties (8) and (9) show that

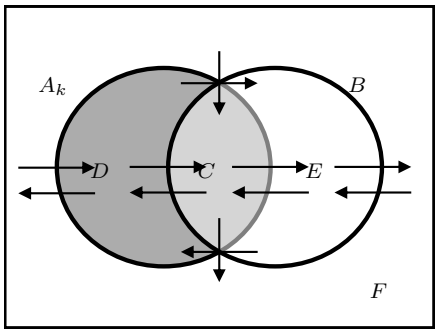


Figure 9: A partition of the node set N where $A_k = C \cup D$ and $B = C \cup E$.

$$f_{ij} = c_{ij},\ \forall\, i \in A_k,\, j \in A_k^c, \tag{13}$$

$$f_{ij} = 0,\ \forall\, i \in A_k^c,\, j \in A_k. \tag{14}$$

The capacity of the cut (B, B^c) in the DAG D_k, defined in (1), satisfies

$$\mathrm{cap}_k(B, B^c) = \mathrm{cap}_k(B, D) + \mathrm{cap}_k(B, F), \tag{15}$$

where $B^c = D \cup F$. Under the flow allocation (f_{ij}), we have

$$\mathrm{cap}_k(B, D) = \sum_{(i,j) \in E_k:\, i \in B, j \in D} c_{ij} \geq \sum_{(i,j) \in E_k:\, i \in B, j \in D} f_{ij}. \tag{16}$$

Applying the flow conservation equation to the collection of nodes in D yields

$$\sum_{(i,j) \in E_k:\, i \in B, j \in D} f_{ij} \geq \sum_{i \in D} q_i + \sum_{(i,j) \in E_k:\, i \in D, j \in F} f_{ij}. \tag{17}$$

In (17), the first term is the sum of incoming flows into the set D; notice that there is no incoming flow from F to D because of the flow property (14). The second term is the sum of queue overload rates in D. The last term is a partial sum of outgoing flows leaving the set D, not counting flows from D to B; hence the inequality (17). From the flow property (13), the outgoing flows from the set D to F satisfy

$$\sum_{(i,j) \in E_k:\, i \in D, j \in F} f_{ij} = \sum_{(i,j) \in E_k:\, i \in D, j \in F} c_{ij}. \tag{18}$$

Combining (15)-(18) yields

$$\begin{aligned} \mathrm{cap}_k(B, B^c) &= \mathrm{cap}_k(B, D) + \mathrm{cap}_k(B, F) \\ &\geq \sum_{i \in D} q_i + \sum_{(i,j) \in E_k:\, i \in D, j \in F} c_{ij} + \mathrm{cap}_k(B, F) \\ &> \sum_{(i,j) \in E_k:\, i \in D, j \in F} c_{ij} + \mathrm{cap}_k(B, F) \\ &= \mathrm{cap}_k(A_k \cup B, F), \end{aligned} \tag{19}$$

where the second inequality follows that all nodes in D are overloaded and $q_n > 0$ for all $n \in D$. Inequality (19) shows that there exists a cut $(A_k \cup B, F)$ that has a smaller capacity, contradicting that (B, B^c) is a min-cut in the DAG D_k. Finally, we note that the partition (A_k, A_k^c) is unique because the lexicographically minimal overload vector is unique by Lemma 1. □

C. PROOF OF THEOREM 3

Proof of the first claim. First we will show that the queue at the source $Q_s(t)$ crosses any arbitrary threshold R_1. We know that for some node $n \in A_k$, $Q_n(t) \to \infty$ as $t \to \infty$ because the external arrival rate to the source $s \in A_k$ is larger than the rate of departure from set A_k, i.e. $\lambda > cap(A_k, A_k^c)$. The backpressure algorithm sends packets on a link (i,j) only if $Q_i(t) > Q_j(t)$. Hence, at any time-slot if a node $b \neq s$ has a large backlog, then one of its parents p must also have a large backlog. Q_p can be slightly smaller than Q_b because Q_b might also receive packets from other nodes at the same time-slot. Specifically, $Q_p(t) > Q_b(t+1) - \sum_i c_{ib}$. Performing the induction on the parent of p we can see that the source node must have a high backlog when any node in A_k develops a high backlog. Note that the network is a DAG and the node n received packets form the source to develop its backlog, so the induction much reach the source node. Hence, when $Q_b(T_1) \gg R_1$, $Q_s(t) > R_1$ for some $t < T_1$.

Now we will show that every node in A_k crosses the threshold R. Let $B_1 \subseteq A_k$ be the set of nodes such that $Q_n(t) > R_1$ for some time $t < T_1$. We showed that $s \in B_1$. We will show that when $B_1 \neq A_k$, there exists some set B_2, such that (i) $B_1 \subset B_2$, and (ii) for every node $n \in B_2$, $Q_n(t) > R_2$ for some $t < T_2$. Here, R_2 and T_2 are large thresholds.

Assume $B_1 \neq A_k$. Let $C_1 = A_k \backslash B_1$, i.e all nodes in C_1 haven't crossed the threshold R_1 until time T_1. Let $c_{B_1 C_1}$ be the total capacity of the links going from B_1 to C_1, and $c_{C_1 A_k^c}$ be the total capacity of the links going from C_1 to A_k^c. We have $c_{B_1 C_1} > c_{C_1 A_k^c}$ because (A_k, A_k^c) is the smallest min-cut (see Figure 10). When the backlogs of the nodes of B_1 are much larger than the nodes of C_1, the nodes in C_1 receive packets from B_1 at the rate of $c_{B_1 C_1}$ packets per time-slot, and no packets are sent in the reversed direction. The rate of packets leaving the nodes in C is upper bounded

by $c_{B_1 A_k^c}$ which is smaller than the incoming rate. Hence, at least one node $n' \in C$ must collect a large backlog, say larger than $R_2 < R_1$. So, each node in the set $B_2 = B_1 \cup \{n'\}$ have a backlog larger than R_2 at some finite time T_2.

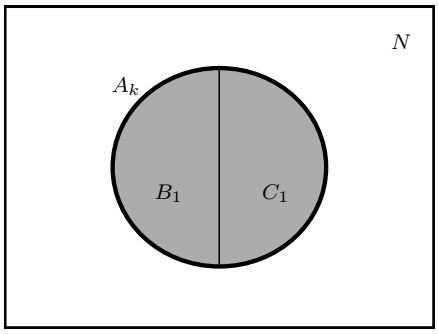


Figure 10: Let (A_k, A_k^c) be the smallest min-cut. We showed that $s \in B_1$. Say, $c_{C_1 A^c} \geq c_{B_1 C_1}$ then the cut (B_1, B_1^c) has the capacity of $c_{B_1 A^c} + c_{B_1 C_1} \leq cap(A_k, A_k^c)$. This contradicts the assumption that (A_k, A_k^c) is the smallest min-cut. So, $c_{C_1 A_k^c} < c_{B_1 C_1}$.

Now using induction we can see that for B_m where $m < |A_k|$, $B_m = A_k$ and all the nodes in B_m cross a threshold $R = \min\{R_1, ..., R_m\}$ by time $T = \max\{T_1, ..., T_m\}$. □

PROOF OF THE SECOND CLAIM. We will use the following fact to prove this claim: for any subset of nodes S, if the number of packets entering S is lower than or equal to the number of packets leaving S on every time-slot, then the total backlog in S doesn't grow. So, the backlog in each node of S is bounded.

Assume a node b develops a backlog $Q_b(t) > R_1$. Here R_1 is a chosen such that

$$R_1 = |A_k^c| \sum_{i,j \in A_k^c} c_{ij} + \max_{n \in A_k^c} Q_n(0).$$

Consider a subset B of A_k^c such that for every node $i \in B$ and $j \in C = A_k^c \backslash B$, $(Q_i(t) - Q_j(t)) > c_{ij}$. The sets B and C must be nonempty because $Q_b(t)$ is large and $Q_d(t)$ is zero, that is $b \in B$ and $d \in C$. Note that backpressure doesn't send any data from C to B.

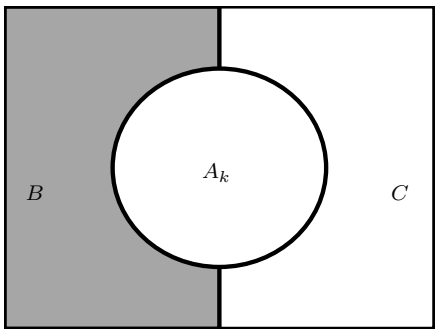


Figure 11: Let (A_k, A_k^c) be the smallest min-cut. We showed that $d \in C$. Say, $c_{AB} > c_{BC}$ then the cut $(B \cup A_k, (B \cup A_k)^c)$ has the capacity of $c_{BC} + c_{A_k C} < c_{AB} + c_{A_k C} = cap(A_k, A_k^c)$. This contradicts the assumption that (A_k, A_k^c) is the smallest min-cut. So, $c_{AB} < c_{BC}$.

Let c_{AB} be the capacity of the links going from A to B, and let c_{BC} be the capacity of the links going from B to C. So, the number of packets entering B at timeslot t is upper bounded by c_{AB}. The number of packets leaving B is equal to c_{BC}. Since (A, A^c) is the smallest min-cut, $c_{AB} \leq c_{BC}$ (see Figure 11). Hence, the number of packets entering B is less than or equal to the number of packets leaving it at time t.

Therefore as soon as one of the nodes crosses threshold R_1, the sum backlog becomes bounded. We can choose a threshold $R \gg R_1$ such that this threshold is never crossed by any nodes in A_k^c.

□

D. LEMMA 8

LEMMA 8. *Consider the case when $\lambda > f_k^{\max}$. The link reversal algorithm is applied on DAG D_k to obtain D_{k+1}. Let (A_k, A_k^c) and (A_{k+1}, A_{k+1}^c) be the smallest min-cuts of D_k and D_{k+1} respectively. Then, either $cap_k(A_k, A_k^c) > cap_{k+1}(A_{k+1}, A_{k+1}^c)$, or $cap_k(A_k, A_k^c) = cap_{k+1}(A_{k+1}, A_{k+1}^c)$ and $|A_{k+1}| > |A_k|$*

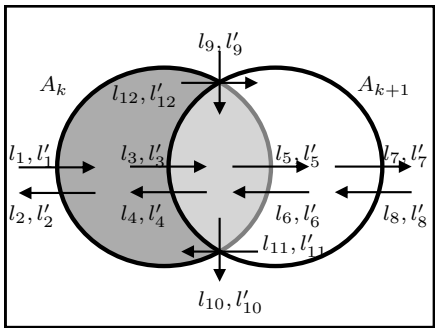


Figure 12: Here l_i represents the sum of the capacities of the links going from one partition to the next in the DAG D_k, and l'_i represents the sum of the link capacities in the DAG D_{k+1}. For example, $l9$ and $l9'$ represent the links that go from $(A_k \cup A_{k+1})^c$ to $(A_k \cap A_{k+1})$ in DAGs D_k and D_{k+1} respectively.

PROOF. Consider the partitioning of the nodes as shown in Figure 12. For $i = 1, ..., 12$, l_i represents the sum of the capacities of the links going from one partition to the next in the DAG D_k, and l'_i represents the sum of the link capacities in the DAG D_{k+1}. The capacities of the smallest min-cut, before and after the reversal are given by

$$cap_k(A_k, A_k^c) = l_2 + l_5 + l_{10} + l_{12} \text{ and}$$

$$cap_{k+1}(A_{k+1}, A_{k+1}^c) = l'_4 + l'_7 + l'_{10} + l'_{11}$$

respectively. Note that only the links that are coming into A_k are different in D_k and D_{k+1}. So

$$l_i = l'_i \text{ for } i = 3, 4, 7, 8, 10, 12. \tag{20}$$

Because of the reversal there are no links coming into A_k in the DAG D_{k+1}:

$$l'_1, l'_6, l'_9, l'_{11} = 0. \tag{21}$$

After the reversal, the incoming links to A_k become outgoing from A_k,

$$l'_{10} = l_{10} + l_9. \tag{22}$$

(Corresponding equations for l'_2, l'_5 and l'_{12} are omitted because they are not necessary for the proof). Since (A_k, A_k^c) is a min-cut,

$$l_5 \leq l_7. \tag{23}$$

This is true because otherwise the cut $(A_k \cup A_{k+1}, (A_k \cup A_{k+1})^c)$ in the DAG D_k has a smaller capacity then the min cut $(A_k, A_k)^c$. Specifically, let us assume $l_5 > l_7$. Then, we get the contradiction:

$$\begin{aligned} cap_k(A_k \cup A_{k+1}, (A_k \cup A_{k+1})^c) &= l_2 + l_7 + l_{10} \\ &< l_2 + l_5 + l_{10} + l_{12} \\ &= cap_k(A_k, A_k)^c \end{aligned}$$

First we will show that if $A_k \backslash A_{k+1} \neq \phi$, then the capacity of the DAG must have increased. The proof is by contradiction.

Let us assume that the throughput didn't increase. So,

$$\begin{aligned} cap_k(A_k, A_k^c) &\geq cap_{k+1}(A_{k+1}, A_{k+1}^c) \\ &= l'_4 + l'_7 + l'_{10} + l'_{11} \\ &= l_4 + l_7 + l_{10} + 0 && (24) \\ &\geq l_4 + l_5 + l_{10} && (25) \\ &= cap_k(A_k \cap A_{k+1}, (A_k \cap A_{k+1})^c). && (26) \end{aligned}$$

(24) is follows from (20) and (21), and (25) follows from (23). Since $A_k \backslash A_{k+1} \neq \phi$ by assumption, $|A_k| > |A_k \cap A_{k+1}|$. This leads to a contradiction, because in DAG D_k the cut $(A_k \cap A_{k+1}, (A_k \cap A_{k+1})^c)$ is smaller than the smallest min-cut (A_k, A_k^c). Hence, $cap_k(A_k, A_k^c) < cap_{k+1}(A_{k+1}, A_{k+1}^c)$.

Next, we will consider the case $A_k \backslash A_{k+1} = \phi$. Using (23),

$$cap_k(A_k, A_k^c) = l_5 + l_{10} \leq l_7 + l_{10}.$$

In this situation, we again have two cases. First, if $A_k = A_{k+1}$ we know that $l'_{10} > l_{10}$ and $l_7 = 0$. Hence, $cap_k(A_k, A_k^c) < l'_{10} = cap_{k+1}(A_{k+1}, A_{k+1}^c)$.

Second, if $A_k \subset A_{k+1}$, then $|A_k| > |A_{k+1}|$ and

$$l'_{10} \geq l_{10}. \tag{27}$$

Using (20) and (27) $cap_k(A_k, A_k^c) \leq cap_{k+1}(A_{k+1}, A_{k+1}^c)$. $\square$