RESEARCH ARTICLE OPEN ACCESS

Backpressure-Based Packet-By-Packet Adaptive Routing For Traffic Management in Communication Networks

P. Swetha, Mrs. O. Rajitha
M. Tech (S.E), Sir Visweshwaraih Institute of Science & Tech, Madanapalli.
Assistant professor, Sir Visweshwaraih Institute of Science & Tech, Madanapalli

Abstract
Back pressure-based adaptive routing algorithms where each packet is routed along a possibly different pathhave been extensively studied in the literature. However, suchalgorithms typically result in poor delay performance and involvehigh implementation complexity. In this paper, we develop anew adaptive routing algorithm built upon the widely-studiedback-pressure algorithm. We decouple the routing and schedulingcomponents of the algorithm by designing a probabilistic routingtable which is used to route packets to per-destination queues.The scheduling decisions in the case of wireless networks aremade using counters called shadow queues. The results arealso extended to the case of networks which employ simpleforms of network coding. In that case, our algorithm provides alow-complexity solution to optimally exploit the routing-codingtrade-off.

I. INTRODUCTION

The back-pressure algorithm introduced in [25] has beenwidely studied in the literature. While the ideas behindscheduling using the weights suggested in that paper have beensuccessful in practice in base stations and routers, the adaptiverouting algorithm is rarely used. The main reason for this isthat the routing algorithm can lead to poor delay performancedue to routing loops. Additionally, the implementation of theback-pressure algorithm requires each node to maintain predestination queues which can be burdensome for a wirelineor wireless router. Motivated by these considerations, we re-examine the back-pressure routing algorithm in the paper anddesign a new algorithm which has much superior performanceand low implementation complexity.

Prior work in this area [22] has recognized the importanceof doing shortest-path routing to improve delay performanceand modified the back-pressure algorithm to bias it towardstaking shortest-hop routes. A part of our algorithm has similarmotivating ideas, but we do much more. In addition to provablythroughput-optimal routing which minimizes the numberof hops taken by packets in the network, we decouple routingand scheduling in the network through the use of probabilisticrouting tables and the so-called shadow queues. The min-hoprouting idea was studied first in a conference paper [7] andshadow queues were introduced in [6], but the key step ofdecoupling the routing and scheduling which leads to bothdramatic delay reduction and the use of per-next-hop queueingis original here. The min-hop routing idea is also studied in[26] but their solution requires even more queues than theoriginal back-pressure algorithm.

We also consider networks where simple forms of networkcoding is allowed [17]. In such networks, a relay between twoother nodes XORs packets and broadcast them to decrease thenumber of transmissions. There is a trade-off between choosinglong routes to possibly increase network coding opportunities(see the notion of reverse carpooling in [10]) and choosingshort routes to reduce resource usage. Our adaptive routingalgorithm can be modified to automatically realize this trade-offwith good delay performance. In addition, network codingrequires each node to maintain more queues [15] and ourrouting solution at least reduces the number of queues to bemaintained for routing purposes, thus partially mitigating theproblem. An offline algorithm for optimally computing therouting-coding trade-off was proposed in [23]. Our optimizationformulation bears similarities to this work but our mainfocus is on designing low-delay on-line algorithms. Backpressuresolutions to network coding problems have also beenstudied in [14], [11], [8], but the adaptive routing-codingtrade-off solution that we propose here has not been studiedpreviously.
We summarize our main results below.
_ Using the concept of shadow queues, we decouple routingand scheduling. A shadow network is used to update aprobabilistic routing table which packets use upon arrivalat a node. The back-pressure-based scheduling algorithmis used to serve FIFO queues over each link.
_ The routing algorithm is designed to minimize the averagenumber of hops used by packets in the network.This idea, along with the scheduling/routing

decoupling,leads to delay reduction compared with the traditionalback-pressure algorithm.

_ Each node has to maintain counters, called shadowqueues, per destination. This is very similar to the idea ofmaintaining a routing table per destination. But the realqueues at each node are per-next-hop queues in the caseof networks which do not employ network coding. Whennetwork coding is employed, per-previous-hop queuesmay also be necessary but this is a requirement imposedby network coding, not by our algorithm.

_ The algorithm can be applied to wireline and wirelessnetworks. Extensive simulations show dramatic improvementin delay performance compared to the back-pressurealgorithm.

The rest of the paper is organized as follows. We presentthe network model in Section II. In Section III and IV, the traditionalback-pressure algorithm and its modified version areintroduced. We develop our adaptive routing and scheduling algorithm for wireline and wireless networks with and without network coding in Section V, VI and VII. In Section VIII, thesimulation results are presented. We conclude our paper inSection IX.

II. THE NETWORK MODEL

We consider a multi-hop wire line or wireless networkrepresented by a directed graph G = (N;L); where N is theset of nodes and L is the set of directed links. A directed linkthat can transmit packets from node n to node j is denotedby (nj) 2 L: We assume that time is slotted and define thelink capacity cnj to be the maximum number of packets thatlink (nj) can transmit in one time slot.

Let F be the set of flows that share the network. Eachflow is associated with a source node and a destination node,but no route is specified between these nodes. This meansthat the route can be quite different for packets of the sameflow. Let b(f) and e(f) be source and destination nodes,respectively, of flow f: Let xf be the rate (packets/slot) atwhich packets are generated by flow f: If the demand onthe network, i.e., the set of flow rates, can be satisfied bythe available capacity, there must exist a routing algorithmand a scheduling algorithm such that the link rates lie inthe capacity region. To precisely state this condition, wedefine _dnj to be the rate allocated on link (nj) to packets destined for node d: Thus, the total rate allocated to all flowsat link (nj) is given by

$$\mu_{nj} := \sum_{d \in \mathcal{N}} \mu_{nj}^{d}.$$

Clearly, for thenetwork to be able to meet the traffic demand, we should have:

$$\{\mu_{nj}\}_{(nj)\in\mathcal{L}} \in \Lambda,$$

whereA_ is the capacity region of the network for 1-hop traffic.The capacity region of the network for 1-hop traffic containsall sets of rates that are stabilizable by some kind of schedulingpolicy assuming all traffics are 1-hop traffic. As a special case,in the wire line network, the constraints are:

$$\mu_{nj} \le c_{nj}, \qquad \forall (nj).$$

As opposed to _; let _ denote the capacity region of the multihopnetwork, i.e., for any set of flows fxf gf2F 2 _; thereexists some routing and scheduling algorithms that stabilizethe network.

In addition, a flow conservation constraint must be satisfiedat each node, i.e., the total rate at which traffic can possiblyarrive at each node destined to d must be less than or equal tothe total rate at which traffic can depart from the node destinedto d :

$$\sum_{f\in\mathcal{F}} x_f I_{\{b(f)=n,e(f)=d\}} + \sum_{l:(ln)\in\mathcal{L}} \mu_{ln}^{d} \le \sum_{j:(nj)\in\mathcal{L}} \mu_{nj}^{d}, \tag{1}$$

where I denotes the indicator function. Given a set of arrivalrates x = fxf gf2F that can be accommodated by the network,one version of the multi-commodity flow problem is to findthe traffic splits _dnj such that (1) is satisfied. However, findingthe appropriate traffic split is computationally prohibitive andrequires knowledge of the arrival rates. The back-pressurealgorithm to be described next is an adaptive solution to themulti-commodity flow problem.

III. THROUGHPUT-OPTIMAL BACK-PRESSUREALGORITHM AND ITS LIMITATIONS

The back-pressure algorithm was first described in [25]in the context of wireless networks and independently discoveredlater in [2] as a low-complexity solution to certainmulti-commodity flow problems. This algorithm combinesthe scheduling and routing functions together. While manyvariations of this basic algorithm have been studied, theyprimarily focus on maximizing throughput and do not considerQoS performance. Our algorithm uses some of these ideasas building blocks and therefore, we first describe the basicalgorithm, its drawbacks and some prior solutions.

The algorithm maintains a queue for each destination at eachnode. Since the number of destinations can be as large as thenumber of nodes, this per-destination queueing requirement can be quite large for practical implementation in a network.At each link, the algorithm assigns a weight to each possibledestination which is called back-pressure. Define the back pressureat link (nj) for destination d at slot t to be

$$w_{nj}^{d}[t] = Q_{nd}[t] - Q_{jd}[t],$$

where Qnd[t] denotes the number of packets at node n destinedfor node d at the beginning of time slot t:

Under this notation,Qnn[t] = 0; 8t: Assign a weight wnj to each link (nj); wherewnj is defined to be the maximum back-pressure over allpossible destinations, i.e.,

$$w_{nj}[t] = \max_d w_{nj}^d[t].$$

Let d_nj be the destination which has the maximum weight onlink (nj);

$$d_{nj}^*[t] = \arg\max_d \{w_{nj}^d[t]\}. \tag{2}$$

If there are ties in the weights, they can be broken arbitrarily.Packets belonging to destination d_nj[t] are scheduled fortransmission over the activated link (nj): A schedule is a set oflinks that can be activated simultaneously without interferingwith each other. Let □ denote the set of all schedules. Theback-pressure algorithm finds an optimal schedule __[t] whichis derived from the optimization problem:

$$\pi^*[t] = \arg\max_{\pi\in\Gamma} \sum_{(nj)\in\pi} c_{nj} w_{nj}[t]. \tag{3}$$

Specially, if the capacity of every link has the same value,the chosen schedule maximizes the sum of weights in anyschedule.At time t; for each activated link (nj) 2 __[t] we removecnj packets from Qnd_nj[t] if possible, and transmit thosepackets to Qjd_nj[t]: We assume that the departures occur firstin a time slot, and external arrivals and packets transmittedover a link (nj) in a particular time slot are available to node j at the next time slot. Thus the evolution of the queue Qnd[t]is as follows:

$$\begin{aligned} Q_{nd}[t+1] \; = \; & Q_{nd}[t] - \sum_{j:(nj)\in\mathcal{L}} I_{\{d_{nj}^*[t]=d\}}\,\hat{\mu}_{nj}[t] \\ & + \sum_{l:(ln)\in\mathcal{L}} I_{\{d_{ln}^*[t]=d\}}\,\hat{\mu}_{ln}[t] \\ & + \sum_{f\in\mathcal{F}} I_{\{b(f)=n,e(f)=d\}}\,a_f[t], \end{aligned} \tag{4}$$

where ^_nj[t] is the number of packets transmitted over link(nj) in time slot t and af [t] is the number of packets generatedby flow f at time t: It has been shown in [25] that the backpressurealgorithm maximizes the throughput of the network.

A key feature of the back-pressure algorithm is that packetsmay not be transferred over a link unless the back-pressureover a link is non-negative and the link is included in thepicked schedule. This feature prevents further congestingnodes that are already congested, thus providing the adaptivelyof the algorithm. Notice that because all links can be activatedwithout interfering with each other in the wire line network, □is the set of all links. Thus the back-pressure algorithm can belocalized at each node and operated in a distributed mannerin the wire line network.The back-pressure algorithm has several disadvantages thatprohibit practical implementation:

_ The back-pressure algorithm requires maintaining queuesfor each potential destination at each node. This queuemanagement requirement could be a prohibitive overheadfor a large network.

_ The back-pressure algorithm is an adaptive routing algorithmwhich explores the network resources and adaptsto different levels of traffic intensity. However it mightalso lead to high delays because it may choose long pathsunnecessarily. High delays are also a result of maintaininga large number of queues at each node. Only one queuecan be scheduled at a time, and the unused service couldfurther contribute to high latency.

In this paper, we address the high delay and queueingcomplexity issues. The computational complexity issue forwireless networks is not addressed here. We simply use therecently studied greedy maximal scheduling (GMS) algorithm.Here we call it the largest-weight-first algorithm, in short,LWF algorithm. LWF algorithm requires the same queue structurethat the back-pressure algorithm uses. It also calculates theback-pressure at each link using the same way. The differencebetween these two algorithms only lies in the methods to pick aschedule. Let S denote the set of all links initially. Let Nb(l) bethe set of links within the interference range of link l includingl itself. At each time slot, the LWF algorithm picks a link lwith the maximum weight first, and removes links within theinterference range of link l from S; i.e., S = SnNb(l); then itpicks the link with the maximum weight in the updated set S;and so forth. It should be noticed that LWF algorithm reducesthe computational complexity with a price of the reductionof the network capacity region. The LWF algorithm wherethe weights are queue lengths (not back-pressures) has beenextensively studied in [9], [16], [4], [18], [19]. While these studies indicate that there may be reduction in throughputdue to LWF in certain special network topologies, it seemsto perform well in simulations and so we adopt it here.

In the rest of the paper, we present our main results whicheliminate many of the problems associated with the backpressurealgorithm.

IV. MIN-RESOURCE ROUTING USING BACK-PRESSUREALGORITHM

As mentioned in Section III, the back-pressure algorithmexplores all paths in the network and as a result may choosepaths which are unnecessarily long which may even containloops, thus leading to poor performance. We address thisproblem by introducing a cost function which measures thetotal amount of resources used by all flows in the network.Specially, we add up traffic loads on all links in the networkand use this as our cost function. The goal then is to minimizethis cost subject to network capacity constraints.Given a set of packet arrival rates that lie within the capacityregion, our goal is to find the routes for flows so that we use asfew resources as possible in the network. Thus, we formulatethe following optimization problem:

$$\begin{aligned} \min \quad & \sum_{(nj)\in\mathcal{L}} \mu_{nj} \qquad (5)\\ s.t. \quad & \sum_{f\in\mathcal{F}} x_f I_{\{b(f)=n,e(f)=d\}} + \sum_{(ln)\in\mathcal{L}} \mu_{ln}^d \le \sum_{(nj)\in\mathcal{L}} \mu_{nj}^d,\\ & \qquad \forall d\in\mathcal{N}, n\in\mathcal{N},\\ & \{\mu_{nj}\}_{(nj)\in\mathcal{L}} \in \Lambda. \end{aligned}$$

We now show how a modification of the back-pressurealgorithm can be used to solve this min-esource routingproblem. (Note that similar approaches have been used in[20], [21], [24], [12], [13] to solve related resource allocationproblems.)

Let fqndg be the Lagrange multipliers corresponding to theflow conservation constraints in problem (5). Appending theseconstraints to the objective, we get

$$\begin{aligned} &\min_{\mu\in\Lambda} \sum_{(nj)\in\mathcal{L}} \mu_{nj} + \sum_{n,d} q_{nd}\Big(\sum_{f\in\mathcal{F}} x_f I_{\{n=b(f),e(f)=d\}}\\ &\qquad + \sum_{(ln)\in\mathcal{L}} \mu_{ln}^d - \sum_{(nj)\in\mathcal{L}} \mu_{nj}^d\Big) \qquad (6)\\ &= \min_{\mu\in\Lambda}\Big(-\sum_{(nj)\in\mathcal{L}} \sum_{d} \mu_{nj}^d (q_{nd} - q_{jd} - 1)\\ &\qquad -\sum_{n,d} q_{nd} \sum_{f\in\mathcal{F}} x_f I_{\{n=b(f),e(f)=d\}}\Big). \end{aligned}$$

If the Lagrange multipliers are known, then the optimal _ canbe found by solving

$$\max_{\mu\in\Lambda} \sum_{(nj)\in\mathcal{L}} \mu_{nj} w_{nj}$$

wherewnj = maxd(qnd□qjd□1): The form of the constraintsin (5) suggests the following update algorithm to compute

q_{nd} :

$$\begin{aligned} q_{nd}[t+1] = \Big[q_{nd}[t] + \frac{1}{M}\Big(\sum_{f\in\mathcal{F}} x_f I_{\{n=b(f),e(f)=d\}}\\ + \sum_{(ln)\in\mathcal{L}} \mu_{ln}^d - \sum_{(nj)\in\mathcal{L}} \mu_{nj}^d\Big)\Big]^+ \qquad (7) \end{aligned}$$

where 1M is a step-size parameter. Notice that Mqnd[t] looksvery much like a queue update equation, except for the factthat arrivals into Qnd from other links may be smaller than_dln when Qld does not have enough packets. This suggeststhe following algorithm.

Min-resource routing by back-pressure: At time slot t;

_ Each node n maintains a separate queue of packets foreach destination d; its length is denoted Qnd[t]. Each linkis assigned a weight

$$w_{nj}[t] = \max_d \left(\frac{1}{M} Q_{nd}[t] - \frac{1}{M} Q_{jd}[t] - 1\right), \qquad (8)$$

where M > 0 is a parameter.

_ Scheduling/routing rule:

$$\pi^*[t] \in \arg\max_{\pi\in\Gamma} \sum_{(nj)\in\pi} c_{nj} w_{nj}[t]. \qquad (9)$$

_ For each activated link (nj) 2 __[t] we remove cnjpackets from Qnd_nj[t] if possible, and transmit thosepackets to Qjd_nj[t]; where d_nj[t] achieves the maximumin (8).

Note that the above algorithm does not change if we replacethe weights in (8) by the following, re-scaled ones:

$$w_{nj}[t] = \max_d \left(Q_{nd}[t] - Q_{jd}[t] - M\right), \qquad (10)$$

and therefore, compared with the traditional back-pressurescheduling/routing, the only difference is that each link weightis equal to the maximum differential backlog minus parameterM. (M = 0 reverts the algorithm to the traditional one.) Forsimplicity, we call this algorithm M-back-pressure algorithm.The performance of the stationary process which is "produced"by the algorithm with fixed parameter M is withino(1) of the optimal as M goes to 1 (analogous to the proofsin [21], [24]; see also the related proof in [12], [13]):

$$\left| \mathbb{E}\left[\sum_{(nj)\in\mathcal{L}} \mu_{nj}[\infty]\right] - \sum_{(nj)\in\mathcal{L}} \mu_{nj}^* \right| = o(1),$$

where __ is an optimal solution to (5).

Although M-back-pressure algorithm could reduce the delay by forcing flows to go through shorter routes, simulations indicate a significant problem with the basic algorithm presented above. A link can be scheduled only if the backpressure of at least one destination is greater than or equal to M: Thus, at light to moderate traffic loads, the delays could be high since the back-pressure may not build up sufficiently fast. In order to overcome all these adverse issues, we develop a new routing algorithm in the following section. The solution also simplifies the queuing data structure to be maintained at each node.

V. PARN: PACKET-BY-PACKET ADAPTIVE ROUTING AND SCHEDULING ALGORITHM FOR NETWORKS

In this section, we present our adaptive routing and scheduling algorithm. We will call it PARN (Packet-by-Packet Adaptive Routing for Networks) for ease for repeated reference later. First, we introduce the queue structure that is used in PARN.

In the traditional back-pressure algorithm, each node n has to maintain a queue qnd for each destination d: Let jN j and jDj denote the number of nodes and the number of destinations in the network, respectively. Each node maintains jDj queues. Generally, each pair of nodes can communicate along a path connecting them. Thus, the number of queues maintained at each node can be as high as one less than the number of nodes in the network, i.e., jDj=jN j 1:

Instead of keeping a queue for every destination, each node n maintains a queue qnj for every neighbour j; which is called a real queue. Notice that real queues are per-neighbour queues. Let Jn denote the number of neighbours of node n; and let Jmax = maxnJn: The number of queues at each node is no greater than Jmax: Generally, Jmax is much smaller than jN j: Thus, the number of queues at each node is much smaller compared with the case using the traditional back-pressure algorithm.

In additional to real queues, each node n also maintains a counter, which is called shadow queue, pnd for each destination d: Unlike the real queues, counters are much easier to maintain even if the number of counters at each node grows linearly with the size of the network. A back-pressure algorithm run on the shadow queues is used to decide which links to activate. The statistics of the link activation are further used to route packets to the per-next-hop neighbour queues mentioned earlier. The details are explained next.

A. Shadow Queue Algorithm – M-back-pressure Algorithm.

The shadow queues are updated based on the movement of fictitious entities called shadow packets in the network. The movement of the fictitious packets can be thought of as an exchange of control messages for the purposes of routing and schedule. Just like real packets, shadow packets arrive from outside the network and eventually exit the network. The external shadow packet arrivals are general as follows: when an exogenous packet arrives at node n to the destination d; the shadow queue pnd is incremented by 1; and is further incremented by 1 with probability " in addition. Thus, if the arrival rate of a flow f is xf ; then the flow generates “shadow traffic” at a rate xf (1 + "): In words, the incoming shadow traffic in the network is (1 + ") times of the incoming real traffic.

The back-pressure for destination d on link (nj) is taken to Be

$$w^d_{nj}[t] = p_{nd}[t] - p_{jd}[t] - M,$$

where M is a properly chosen parameter. The choice of M will be discussed in the simulations section.

The evolution of the shadow queue pnd[t] is

$$\begin{aligned} p_{nd}[t+1] \quad = \quad & p_{nd}[t] - \sum_{j:(nj)\in\mathcal{L}} I_{\{d^*_{nj}[t]=d\}}\,\hat{\mu}_{nj}[t] \\ & + \sum_{l:(ln)\in\mathcal{L}} I_{\{d^*_{ln}[t]=d\}}\,\hat{\mu}_{ln}[t] \\ & + \sum_{f\in\mathcal{F}} I_{\{b(f)=n,e(f)=d\}}\,\hat{a}_f[t], \end{aligned} \qquad (11)$$

where ^_nj[t] is the number of shadow packets transmitted over link (nj) in time slot t, d_ nj[t] is the destination that has the maximum weight on link (nj); and ^af [t] is the number of shadow packets generated by flow f at time t: The number of shadow packets scheduled over the links at each time instant is determined by the back-pressure algorithm in equation (9).

From the above description, it should be clear that the shadow algorithm is the same as the traditional back-pressure algorithm, except that it operates on the shadow queueing system with an arrival rate slightly larger than the real external arrival rate of packets. Note the shadow queues do not involve any queueing data structure at each node; there are no packets to maintain in a FIFO order in each queue. The shadow queue is simply a counter which is incremented by 1 upon a shadow packet arrival and decremented by 1 upon a departure.

The back-pressure algorithm run on the shadow queues is used to activate the links. In other words, if __ nj = 1 in (9), then link (nj) is activated and packets are served from the real queue at the link in a first-in, first-out fashion. This is, of course, very different from the traditional back-pressure algorithm where a link is activated to serve packets to a particular destination. Thus, we have to develop a routing scheme that assigns packets arriving to a node to a particular next-hop neighbor so that the system remains stable. We design such an algorithm next.

B. Adaptive Routing Algorithms

Now we discuss how a packet is routed once it arrives at a node. Let us define a variable _d nj[t] to be the number of shadow packets “transferred” from node n to node j for destination d during time slot t by the shadow queue algorithm. Let us denote by __d nj the expected value of _d nj[t], when the shadow queueingprocess is in a stationary regime; let ^_d nj[t] denote an estimate of __d nj, calculated at time t. (In the simulations we use the exponential averaging, as specified in the next section.)

At each time slot, the following sequence of operations occurs at each node n: A packet arriving at node n for destination d is inserted in the real queue qnj for next-hop neighbor j with probability

$$P^d_{nj}[t] = \frac{\hat{\sigma}^d_{nj}[t]}{\sum_{k:(nk)\in\mathcal{L}} \hat{\sigma}^d_{nk}[t]}. \qquad (12)$$

Thus, the estimates ^_d nj[t] are used to perform routing operations: in today’s routers, based on the destination of a packet, a packet is routed to its next hop based on routing table entries. Instead, here, the __’s are used to probabilistically choose the next hop for a packet. Packets waiting at link (nj) are transmitted over the link when that link is scheduled (See Figure 1).

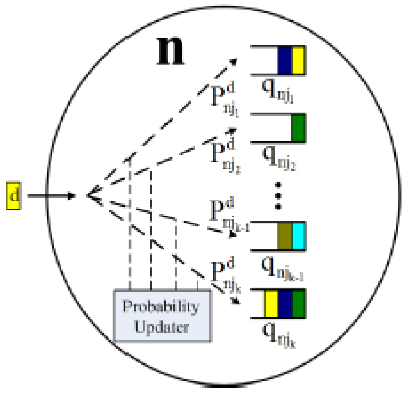

Fig. 1. Probabilistic splitting algorithm in Node n

The first question that one must ask about the above algorithm is whether it is stable if the packet arrival rates from flows are within the capacity region of the multi-hop network. This is a difficult question, in general. Since the shadow queues are positive recurrent, "good" estimates ^_d nj[t] can be maintained by simple averaging (e.g. as specified in the next section), and therefore the probabilities in (12) will stay close to their "ideal" values

$$\bar{P}^d_{nj} = \frac{\bar{\sigma}^d_{nj}}{\sum_{k:(nk)\in\mathcal{L}} \bar{\sigma}^d_{nk}}.$$

The following theorem asserts that the real queues are stable if Pdnj are fixed at _ Pdnj:

Theorem 1: Suppose, Pdnj[t] _ _ Pdnj. Assume that there exists a delta such that fxf (1 + _ + _)g lies in T. Let af [t] be the number of packets arriving from flow f at time slot t; with E(af [t]) = xf and E(af [t]) < 1: Assume that the arrival process is independent across time slots and flows (this assumption can be considerably relaxed). Then, the Markov chain, jointly describing the evolution of shadow queues and real FIFO queues (whose state include the destination of the real packet in each position of each FIFO queue), is positive recurrent.

Proof: The key ideas behind the proof are outlined. The details are similar to the proof in [5] and are omitted.
_ The average rate at which packets arrive to link (nj) is strictly smaller than the capacity allocated to the link by the shadow process if " > 0. (This fact is verified in Appendix A.)
_ It follows that the fluid limit of the real-queue process is same as that of the networks in [3]. Such fluid limit is stable [3], which implies the stability of our process as well.

VI. IMPLEMENTATION DETAILS

The algorithm presented in the previous section ensures that the queue lengths are stable. In this section, we discuss a number of enhancements to the basic algorithm to improve performance.

A. Exponential Averaging

To compute ^_d nj[t] we use the following iterative exponential averaging algorithm:

$$\hat{\sigma}^d_{nj}[t] = (1-\beta)\,\hat{\sigma}^d_{nj}[t-1] + \beta\,\sigma^d_{nj}[t], \tag{13}$$

where 0 < _ < 1:

B. Token Bucket Algorithm

Computing the average shadow rate ^_d nj[t] and generating random numbers for routing packets may impose a computational overhead of routers which should be avoided if possible. Thus, as an alternative, we suggest the following simple algorithm. At each node n; for each next-hop neighbor j and each destination d; maintain a token bucket rdnj: Consider the shadow traffic as a guidance of the real traffic, with tokens removed as shadow packets traverse the link. In detail, the token bucket is decremented by _d nj[t] in each time slot, but cannot go below the lower bound 0:

$$r^d_{nj}[t] = \max\{r^d_{nj}[t-1] - \sigma^d_{nj}[t], 0\}.$$

When rd $r^d_{nj}[t-1] - \sigma^d_{nj}[t] < 0,$ we say that $\sigma^d_{nj}[t] - r^d_{nj}[t-1]$ tokens (associated with bucket rdnj) are "wasted" in slot t. Upon a packet arrival at node n for destination d; find the token bucket rdnj_ which has the smallest number of tokens (the minimization is over next-hop neighbors j), breaking ties arbitrarily, add the packet to the corresponding real queue qnj_ and add one token to the corresponding bucket:

$$r^d_{nj^*}[t] = r^d_{nj^*}[t-1] + 1. \tag{14}$$

To explain how this algorithm works, denote by __d nj the average value of _d nj[t] (in stationary regime), and by _dn the average rate at which real packets for destination d arrive at node n. Due to the fact that real traffic is injected by each source at the rate strictly less than the shadow traffic, we have

$$\eta^d_n < \sum_j \bar{\sigma}^d_{nj}. \tag{15}$$

For a single-node network, (15) just means that arrival rate is less than available capacity. More generally, it is an assumption that needs to be proved. However, here our goal is to provide an intuition behind the token bucket algorithm, so we simply assume (15). Condition (15) guarantees that the token processes are stable (that is, roughly, they cannot runaway to infinity) since the total arrival rate to the token buckets at a node is less than the total service rate and the arrivals employ a join-the-shortest-queue discipline. Moreover, since rdnj[t] are random processes, the token buckets will "hit 0" in a non-zero fraction of time slots, except in some degenerate cases; this in turn means that the arrival rate of packets at the token bucket must be less than the token generation rate:

$$\eta_{nj}^{d} < \bar{\sigma}_{nj}^{d}, \qquad (16)$$

where _d nj is the actual rate at which packets arriving at n and destined for d are routed along link (nj). Inequality (16) thus describes the idea of the algorithm.

Ideally, in addition to (16), we would like to have the ratios _d nj=__d nj to be equal across all j, i.e., the real packet arrival rates at the outgoing links of a node should be proportional to the shadow service rates. It is not difficult to see that if " is very small, the proportion will be close to ideal. In general, the token-based algorithm does not guarantee that, that is why it is an approximation.

Also, to ensure implementation correctness, instead of (14), we use

$$r_{nj_*}^{d}[t] = \min\{r_{nj_*}^{d}[t-1] + 1, B\}, \qquad (17)$$

i.e., the value of rdnj_ [t] is not allowed to go above some relatively large value B, which is a parameter of the order of O(1=_). Under "normal circumstances", rdnj_ [t] "hitting" ceiling B is a rare event, occurring due to the process randomness. The main purpose of having the upper bound B is to detect serious anomalies when, for whatever reason, the condition (15) "breaks" for prolonged periods of time – such situation is detected when any rdnj_ [t] hits the upper bound B frequently.

C. Extra Link Activation

Under the shadow back-pressure algorithm, only links with back-pressure greater than or equal to M can be activated. The stability theory ensures that this is sufficient to render the real queues. On the other hand, the delay performance can still be unacceptable. Recall that the parameter M was introduced to discourage the use of unnecessarily long paths. However, under light and moderate traffic loads, the shadow back-pressure at a link may be frequently less than M, and thus, packets at such links may have to wait a long time before they are processed. One way to remedy the situation is to activate additional links beyond those activated by the shadow back-pressure algorithm.

The basic idea is as follows: in each time slot, first run the shadow back-pressure algorithm. Then, add additional links to make the schedule maximal. If the extra activation procedure depends only on the state of shadow queues (but beyond that, can be random and/or arbitrarily complex), then the stability result of Theorem 1 still holds (with essentially same proof). Informally, the stability prevails, because the shadow algorithm alone provides sufficient average throughput on each link, and adding extra capacity "does not hurt"; thus, with such extra activation, a certain degree of "decoupling" between routing (totally controlled by shadow queues) and scheduling (also controlled by shadow queues, but not completely) is achieved.

For example, in the case of wireline networks, by the above arguments, all links can be activated all the time. The shadow routing algorithm ensures that the arrival rate at each link is less than its capacity. In this case the complete decoupling of routing and scheduling occurs.

In practice, activating extra links which have large queue backlogs leads to better performance than activating an arbitrary set of extra links. However, in this case, the extra activation procedure depends on the state of real queues which makes the issue of validity of an analog of Theorem 1 much more subtle. We believe that the argument in this subsection provides a good motivation for our algorithm, which is confirmed by simulations.

D. The Choice of the Parameter "

From basic queueing theory, we expect the delay at each link to be inversely proportional to the mean capacity minus the arrival rate at the link. In a wireless network, the capacity at a link is determined by the shadow scheduling algorithm. This capacity is guaranteed to be at least equal to the shadow arrival rate. The arrival rate of real packets is of course smaller. Thus, the difference between the link capacity and arrival rate could be proportional to epsilon. Thus, epsilon should be sufficiently large to ensure small delays while it should be sufficiently small to ensure that the capacity region is not diminished significantly. In our simulations, we found that choosing " = 0:1 provides a good tradeoff between delay and network throughput.

In the case of wireline networks, recall from the previous subsection that all links are activated. Therefore, the parameter epsilon plays no role here.

VII. EXTENSION TO THE NETWORK CODING CASE

In this section, we extend our approach to consider networks where network coding is used to improve throughput. We consider a simple form of network coding illustrated in Figure 2. When i and j each have a packet to send to the other through an intermediate relay n, traditional transmission requires the following set of transmissions: send a packet a from ito n, then n to j, followed by j to n and n to i. Instead, using network coding, one can first send from ito n, then j to n, XOR the two packets and broadcast the XORed packet from n to both i and j. This form of network coding reduces the number of transmissions from four to three. However, the network coding can only improve throughput only if such coding opportunities are available in the network. Routing plays an important role in determining whether such opportunities exist. In this section, we design an algorithm to automatically find the right tradeoff between using possibly long routes to provide network coding opportunities and the delay incurred by using long routes.

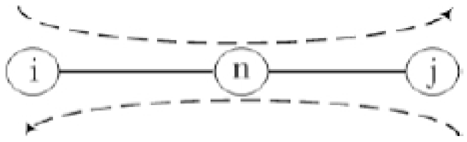

Fig. 2. Network coding opportunity

A. System Model

We still consider the wireless network represented by the graph G = (N;L): Let xf be the rate (packets/slot) at which packets are generated by flow f: To facilitate network coding, each node must not only keep track of the destination of the packet, but also remember the node from which a packet was received. Let _dl nj be the rate at which packets received from either node l or flow l, destined for node d, are scheduled over link (nj). Note that, for compactness of notation, we allow l in the definition of _dl nj to denote either a flow or a node. We assume _dl nj is zero when such a transmission is not feasible, i.e., when n is not the source node or d is not the destination node of flow l, or if (ln) or (nj) is not in L. At node n; the network coding scheme may generate a coded packet by "XORing" two packets received from previous-hop nodes l and j destined for the destination nodes d and d0 respectively, and broadcast the coded packet to nodes j and l: Let _d;d0njjl denote the rate at which coded packets can be transferred from node n to nodes j and l destined for nodes d and d0; respectively. Notice that, due to symmetry, the following equality holds _d;d0 njjl = _d0;d njlj : Assume _d;d0 njjl to be zero if at least one of (nl); (ln); (nj) and (jn) doesn't belong to L: Note that _dl nj = 0 when d = l or d = n; and _d;d0 njjl = 0 when d = n or d0 = n:

There are two kinds of transmissions in our network model: point-to-point transmissions and broadcast transmissions. The total point-to-point rate at which packets received externally or from a previous-hop node are scheduled on link (nj) and destined to d is denoted by

$$\mu^d_{nj,pp} = \sum_{l:l\in\mathcal{F}} \mu^d_{lnj} + \sum_{l:l\in\mathcal{N}} \mu^d_{lnj},$$

and the total broadcast rate at which packets scheduled on link (nj) destined to d is denoted by

$$\mu^d_{nj,broad} = \sum_{d'} \sum_{l:l\neq j} \mu^{d,d'}_{n|jl}.$$

The total point-to-point rate on link (nj) is denoted by

$$\mu_{nj,pp} = \sum_d \mu^d_{nj,pp}$$

and the total broadcast rate at which packets are broadcast from node n to nodes j and l is denoted by

$$\mu_{n|jl} = \sum_{d'} \sum_d \mu^{d,d'}_{n|jl}.$$

Let _ be the set of rates including all point-to-point transmissions and broadcast transmissions, i.e.,

$$\mu \;=\; \{\{\mu_{nj,pp}\}_{(nj)}, \{\mu_{n|jl}\}_{(n|jl)}\}.$$

The multi-hop traffic should also satisfy the flow conservation constraints. Flow conservation constraints: For each node n; each neighbour j; and each destination d; we have

$$\mu^d_{nj,pp} + \mu^d_{nj,broad} \le \sum_k \mu^d_{njk} + \sum_{d'} \sum_{k:k\neq n} \mu^{d,d'}_{j|kn}, \tag{18}$$

where the left-hand side denotes the total incoming traffic rate at link nj destined to d; and the right-hand side denotes the total outgoing traffic rate from link nj destined to d: For each node n and each destination d; we have

$$\sum_{f\in\mathcal{F}} x_f I_{\{b(f)=n,e(f)=d\}} \le \sum_{f\in\mathcal{F}} \sum_{j\in\mathcal{N}} \mu^d_{fnj}, \tag{19}$$

where I denotes the indicator function.

B. Links and Schedules

We allow broadcast transmission in our network model. In order to define a schedule, we first define two kinds of "links:" the point-to-point link and the broadcast link. A point-to-point link (nj) is a link that supports point-to-point transmission, where (nj) 2 L; A broadcast link (njlj) is a "link" which contains links (nl) and (nj) and supports broadcast transmission. Let B denote the set of all broadcast links, thus (njlj) 2 B: Let _ L be the union of the set of the point-to-point links L and the set of the broadcast links B; i.e., _ L = L [B: We let 0 denote the set of links that can be activated simultaneously. By abusing notation, 0 can be thought of as a set of vectors where each vector is a list of 1's or 0's where a 1 corresponds to an active link and a 0 corresponds to an inactive link. Then, the capacity region of the network for 1- hop traffic is the convex hull of all schedules, i.e., _0 = co(0): Thus, , $\mu \in \Lambda'$.

C. Queue Structure and Shadow Queue Algorithm

Each node n maintains a set of counters, which are called shadow queues, plnd for each previous hop l and each destination d; and p0nd for external flows destined for d at node n: Each node n also maintains a real queue, denoted by qlnj; for each previous hop l and each next-hop neighbor j; and q0nj for external flows with their next hop j: By solving the optimization problem with flow conservation constraints, we can work out the back-pressure algorithm for network coding case (see the brief description in Appendix B). More specifically, for each link (nj) 2 L in the network and for each destination d; define the back-pressure at every slot to be

$$\begin{aligned} &w^d_{nj}[t] = \max_{l:(ln)\in\mathcal{L} \text{ or } l=0} w^d_{lnj}[t] \\ &\text{where } w^d_{lnj}[t] = p_{lnd}[t] - p_{njd}[t] - M, \\ &\text{and } l^*_{nj}[t] = \arg\max_{l:(ln)\in\mathcal{L} \text{ or } l=0} w^d_{lnj}[t]. \end{aligned} \quad (20)$$

For each broadcast at node n to nodes j and l destined for d and d0; respectively, define the back-pressure at every slot to be

$$w^{d,d'}_{n|jl}[t] = w^d_{lnj}[t] + w^{d'}_{jnl}[t]. \quad (21)$$

The weights associated with each point-to-point link (nj) 2 L and each broadcast link (njjl) are defined as follows

$$\begin{aligned} &w_{nj}[t] = \max_d \{w^d_{nj}[t]\}, \\ &w_{n|jl}[t] = \max_{d,d'} \{w^{d,d'}_{n|jl}[t]\}, \\ &\text{with } d^*_{nj}[t] = \arg\max_d \{w^d_{nj}[t]\}, \\ &\{d,d'\}^*_{n|jl}[t] = \arg\max_{d,d'} \{w^{d,d'}_{n|jl}[t]\}. \end{aligned} \quad (22)$$

The rate vector ~__[t] at each time slot is chosen to satisfy

$$\tilde{\mu}^*[t] \in \arg\max_{\tilde{\mu}\in\Gamma'} \left\{ \sum_{(nj)\in\mathcal{L}} \tilde{\mu}_{nj,pp} w_{nj}[t] + \sum_{(n|jl)\in\mathcal{B}} \tilde{\mu}_{n|jl} w_{n|jl}[t] \right\}.$$

By running the shadow queue algorithm in network coding case, we get a set of activated links in _ L at each slot. Next we describe the evolution of the shadow queue lengths in the network. Notice that the shadow queues at each node n are distinguished by their previous hop l and their destination d; so plnd only accepts the packets from previous hop l with destination d: The similar rule should be followed when packets are drained from the shadow queue plnd: We assume the departures occur before arrivals at each slot, and the evolution of queues is given by

$$\begin{aligned} p_{lnd}[t+1] = {} & \Big[p_{lnd}[t] - \sum_{j\in\mathcal{N}} \tilde{\mu}^*_{nj,pp}[t] I_{\{l=l^*_{nj}, d=d^*_{nj}\}} \\ & - \sum_{d'\in\mathcal{N}} \sum_{j\in\mathcal{N}} \tilde{\mu}^*_{n|jl}[t] I_{\{\{d,d'\}=\{d,d'\}^*_{n|jl}\}} \Big]^+ \\ & + \sum_{k\in\mathcal{N}} \hat{\mu}^d_{kln}[t] I_{\{k=l^*_{ln}, d=d^*_{ln}\}} \\ & + \sum_{k\in\mathcal{N}} \sum_{d'\in\mathcal{N}} \hat{\mu}^{d,d'}_{l|nk}[t] I_{\{\{d,d'\}=\{d,d'\}^*_{l|nk}\}} \\ & + \sum_{f\in\mathcal{F}} \hat{a}_f[t] I_{\{b(f)=n, e(f)=d, l=0\}}, \end{aligned} \quad (23)$$

where ^_d kln[t] is the actual number of shadow packets scheduled over link (ln) and destined for d from the shadow queue pkld at slot t; ^_d;d0ljnk[t] is the actual number of coded shadow packets transfered from node l to nodes n and k destined for nodes d and d0 at slot t; and ^af denotes the actual number of shadow packets from external flow f received at node n destined for d:

D. Implementation Details

The implementation details of the joint adaptive routing and coding algorithm are similar to the case with adaptive routing only, but the notation is more cumbersome. We briefly describe it here.

1) Probabilistic Splitting Algorithm: The probabilistic splitting algorithm chooses the next hop of the packet based on the probabilistic routing table. Let Pdlnj[t] be the probability of choosing node j as the next hop once a packet destined for d receives at node n from previous hop l or from external flows, i.e., l = 0 at slot t: Assume that Pdlnj[t] = 0 if (nj) 62 L: Obviously, P j2N Pdlnj[t] = 1: Let _d lnj[t] denote the number of potential shadow packets "transferred" from node n to node j destined for d whose previous hop is l during time slot t: Notice that the packet comes from an external flow if l = 0: Also notice that _d lnj[t] is contributed by shadow traffic point-to-point transmission as well as shadow traffic broadcast transmission, i.e.,

$$\begin{aligned} \sigma^d_{lnj}[t] = {} & \mu^*_{nj,pp}[t] I_{\{l=l^*_{nj}[t], d=d^*_{nj}[t]\}} \\ & + \sum_{d'\in\mathcal{N}} \mu^*_{n|jl}[t] I_{\{\{d,d'\}=\{d,d'\}^*_{n|jl}[t]\}}. \end{aligned}$$

We keep track of the the average value of _d lnj[t] across time by using the following updating process:

$$\hat{\sigma}^d_{lnj}[t] = (1-\beta)\hat{\sigma}^d_{lnj}[t-1] + \beta\sigma^d_{lnj}[t], \quad (24)$$

where 0 _ _ _ 1: The splitting probability Pdlnj[t] is expressed as follows:

$$P^d_{lnj}[t] = \frac{\hat{\sigma}^d_{lnj}[t]}{\sum_{k\in\mathcal{N}} \hat{\sigma}^d_{lnk}[t]}. \quad (25)$$

2) Token Bucket Algorithm: At each node n; for each previous-hop neighbor l; next-hop neighbor j and each estination d; we maintain a token bucket rdlnj: At each time slot t; the token bucket is decremented by _d lnj[t]; but cannot go below the lower bound 0 :

$$r^d_{lnj}[t] = \max\{r^d_{lnj}[t-1] - \sigma^d_{lnj}[t], 0\}.$$

When rd

$r^d_{lnj}[t-1] - \sigma^d_{lnj}[t] < 0$, we say $\sigma^d_{lnj}[t] - r^d_{lnj}[t-1]$ tokens (associated with bucket d lnj) are "wasted" in slot t: Upon a packet arrival from previous hop l at node n for destination d at slot t; we find the token bucket rdlnj_ which has the smallest number of tokens (the minimization is over next-hop neighbors j), breaking ties arbitrarily, add the packet to the corresponding real queue qlnj_ ; and add one token from the corresponding bucket:

$$r^d_{lnj^*}[t] = r^d_{lnj^*}[t] + 1.$$

E. Extra link Activation

Like the case without network coding, extra link activation can reduce delays significantly. As in the case without network coding, we add additional links to the schedule based on the queue lengths at each link. For extra link activation purposes, we only consider point-to-point links and not broadcast. Thus, we schedule additional point-to-point links by giving priority to those links with larger queue backlogs.

VIII. SIMULATIONS

We consider two types of networks in our simulations: wireline and wireless. Next, we describe the topologies and simulation parameters used in our simulations, and then present our simulation results.

A. Simulation Settings

1) Wireline Setting: The network shown in Figure 3 has 31 nodes and represents the GMPLS network topology of North America [1]. Each link is assume to be able to transmit 1 packets in each slot. We assume that the arrival process is a Poisson process with parameter _; and we consider the arrivals come within a slot are considered for service at the beginning of the next slot. Once a packet arrives from an external flow at a node n, the destination is decided by probability mass function ^ Pnd; d = 1; 2; :::N; where ^ Pnd is the probability that a packet is received externally at node n destined for d: Obviously, P d:d6=n ^ Pnd = 1; and ^ Pnn = 0: The probability ^ Pnd is calculated by

$$\hat{P}_{nd} = \frac{J_d + J_n}{\sum_{k:k \neq n} (J_k + J_n)},$$

whereJn denotes the number of neighbors of node n: Thus, we use ^ Pnd to split the incoming traffic to each estination based on the degrees of the source and the destination.

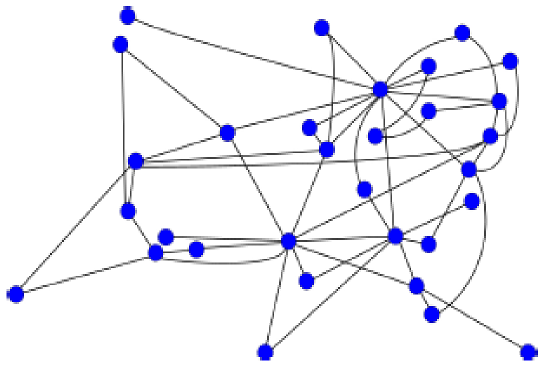

Fig. 3. Sprint GMPLS network topology of North America with 31 nodes.[1]

2) Wireless Setting: We generated a random network with 30 nodes which resulted in the topology in Figure 4. We used the following procedure to generate the random network: 30 nodes are placed uniformly at random in a unit square; then starting with a zero transmission range, the transmission range was increased till the network was connected. We assume that each link can transmit one packet per time slot. We assume a 2-hop interference model in our simulations. By a k-hop interference model, we mean a wireless network where a link activation silences all other links which are k hops from the activated link. The packet arrival processes are generated using the same method as in the wireline case. We simulate two cases given the network topology: the no coding case and the network coding case. In both wireline and wireless simulations, we chose _ in (13) to be 0:02.

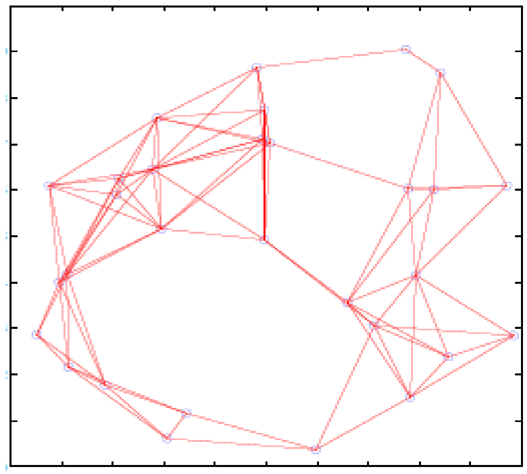

Fig. 4. Wireless network topology with 30 nodes.

B. Simulation Results

1) Wireline Networks: First, we compare the performance of three algorithms: the traditional back-pressure lgorithm, the basic shadow queue routing/scheduling algorithm without the extra link activation enhancement and PARN. Without extra link activation, to ensure that the real arrival rate at each link is less than the link capacity provided by the shadow algorithm, we choose " = 0:02: Figure 5 shows delay as a function of the arrival rate lambda for the three algorithms. As can be seen from the figure, simply using a value of M > 0 does not help to reduce delays without extra link activation. The reason is that, while M > 0 encourages the use of shortest paths, links with back-pressure less than M will not be scheduled and thus can contribute to additional delays.

Next, we study the impact of M on the performance on PARN.

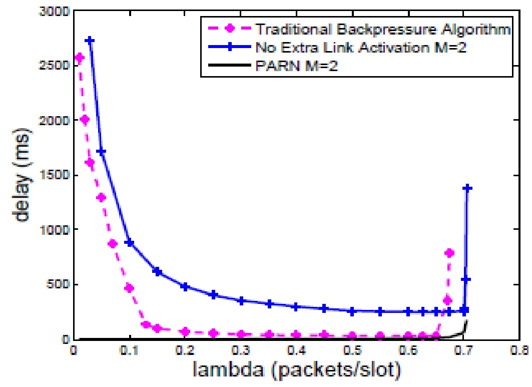

Fig. 5. The impact of the parameter M in Sprint GMPLS network topology

Figure 6 shows the delay performance for various M with extra link activation in the wireline network. The delays or different values of M (except M = 0) are almost the same in the light traffic region. Once M is sufficiently larger than zero, extra link activation seems to play a bigger role, than the choice of the value of M; in reducing the average delays. The wireline simulations show the usefulness of the PARN algorithm for adaptive routing. However, a wireline network does not capture the scheduling aspects inherent to wireless networks, which is studied next.

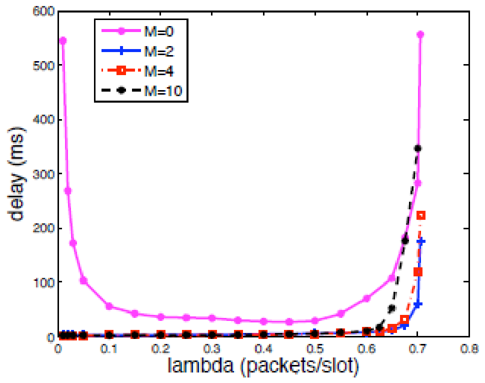

Fig. 6. Packet delay as a function of _ under PARN in Sprint GMPLS network topology

2) Wireless Networks: In the case of wireless networks, even with extra link activation, to ensure stability even when the arrival rates are within the capacity region, we need " > 0: We chose " = 0:1 in our simulations due to reasons mentioned in Section VI.

In Figure 7, we study wireless networks without network coding. From the figure, we see that the delay performance is relatively insensitive to the choice of M as long as it is sufficiently greater than zero. The use of M ensures that unnecessary resource wastage does not occur, and thus, extra link activation can be used to decrease delays significantly.

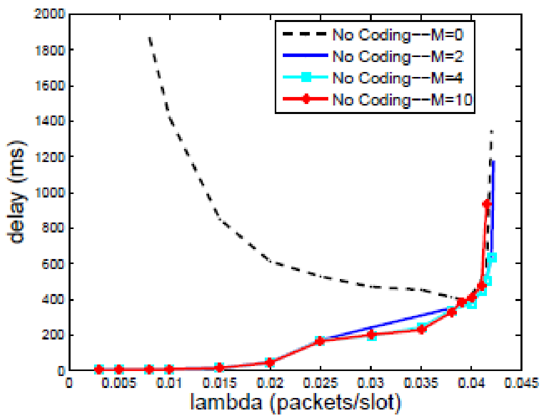

Fig. 7. Packet delay as a function of _ under PARN in the wireless network under 2-hop interference model without network coding

In Figures 8 and 9, we show the corresponding results for the case where both adaptive routing and network coding are used. Comparing Figures 7 and 8, we see that, when used in conjunction with adaptive routing, network coding can increase the capacity region. We make the following observation regarding the case M = 0 in Figure 9: in this case, no attempt is made to optimize routing in the network. As a result, the delay performance is very bad compared to the cases with M > 0 (Figure 8). In other words, network coding alone does not increase capacity sufficiently to overcome the effects of back-pressure routing. On the other hand, PARN with M > 0 harnesses the power of network coding by selecting routes appropriately.

Next, we make the following observation about network coding. Comparing Figures 8 and 9, we noticed that at moderate to high loads (but when the load is within the capacity region of the no coding case), network coding increases delays slightly. We believe that this is due to fact that packets are stored in multiple queues under network coding at each node: for each next-hop neighbour, a queue for each previous-hop neighbour must be maintained. This seems to result in slower convergence of the routing table. Finally, we study the performance of the probabilistic splitting algorithm versus the token bucket algorithm. In our simulations, the token bucket algorithm runs significantly faster, by a factor of 2: The reason is that many more calculations are needed for the probabilistic splitting algorithm as compared to the token bucket algorithm. This may have some implications for practice. So, in Figure 10, we compare the delay performance of the two algorithms. As can be seen from the figure, the token bucket and probabilistic splitting algorithms result in similar performance. Therefore, in practice, the token bucket algorithm may be preferable.

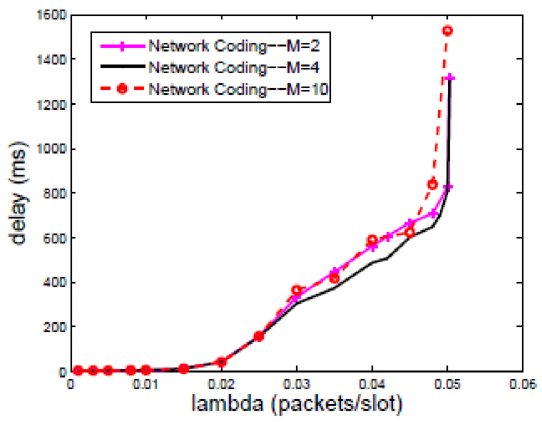

Fig. 8. Packet delay as a function of _ under PARN for M > 0 in the wireless network under 2-hop interference model with network coding

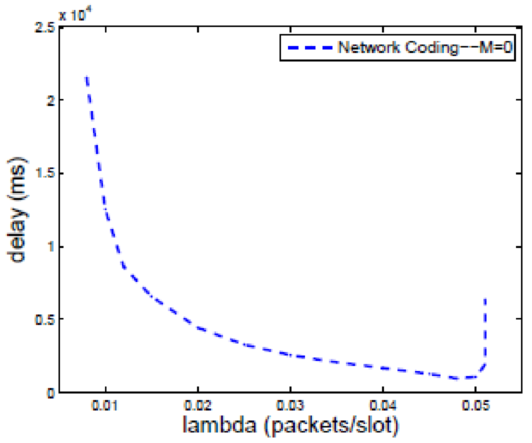

Fig. 9. Packet delay as a function of _ under PARN for M = 0 in the wireless network under 2-hop interference model with network coding

IX. CONCLUSION

The back-pressure algorithm, while being throughputoptimal, is not useful in practice for adaptive routing since the delay performance can be really bad. In this paper, we have presented an algorithm that routes packets on shortest hops when possible, and decouples routing and scheduling using a probabilistic splitting algorithm built on the concept of shadow queues introduced in [6], [7]. By maintaining a probabilistic routing table that changes slowly over time, real packets do not have to explore long paths to improve throughput, this functionality is performed by the shadow "packets." Our algorithm also allows extra link activation to reduce delays. The algorithm has also been shown to reduce the queuing complexity at each node and can be extended to optimally trade off between routing and network coding.

REFERENCES

[1] Sprint IP network performance. Available at https://www.sprint.net/performance/.

[2] B. Awerbuch and T. Leighton. *A simple local-control approximation algorithm for multicommodity flow*. In Proc. 34th Annual Symposium on the Foundations of Computer Science, 1993.

[3] M. Bramson. *Convergence to equilbria for fluid models of FIFO queueing networks*. QueueingSystems:Theory and Applications, 22:5– 45, 1996.

[4] A. Brzezinski, G. Zussman, and E. Modiano. *Enabling distributed throughput maximization in wireless mesh networks - a partitioning approach*. In Proc. ACM Mobicom, Sep. 2006.

[5] L. Bui, R. Srikant, and A. L. Stolyar. *A novel architecture for delay reduction in the back-pressure scheduling algorithm*. IEEE/ACM Trans. Networking.Submitted, 2009.

[6] L. Bui, R. Srikant, and A. L. Stolyar. *Optimal resource allocation for multicast flows in multihop wireless networks*. Philosophical Transactions of the Royal Society, Ser.A, 2008.To appear.

[7] L. Bui, R. Srikant, and A. L. Stolyar. *Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing*. In Proceedings of IEEE INFOCOM Mini-Conference, April 2009.

[8] L. Chen, T. Ho, S. H. Low, M. Chiang, and J. C. Doyle. *Optimization based rate control for multicast with network coding*. In Proc. IEEE INFOCOM, Anchorage, Alaska, May 2007.

[9] A. Dimakis and J. Walrand. *Sufficient conditions for stability of longestqueue- first scheduling: Second-order properties using fluid limits*. Advances in Applied Probability, June 2006.

[10] M. Effros, T. Ho, and S. Kim. *A tiling approach to network code design for wireless networks*. In Information Theory Workshop, 2006.

BIOGRAPHY

P.SWETHA has received her B.Tech degree from Sri VenkatesaPerumal College of Engg& Tech, Puttur. She is post graduated in M. Tech (S.E) stream from Department of Computer Science Engineering from Sir Visweshwaraih Institute of Science & Tech, Madanapalli.