Accelerated Backpressure Algorithm

Michael Zargham, Alejandro Ribeiro, Ali Jadbabaie

***Abstract*—An Accelerated Backpressure (ABP) algorithm is developed using the Accelerated Dual Descent (ADD) method, a distributed approximate Newton-like algorithm that only uses local information. The construction is based on writing the Backpressure algorithm as the solution to a network feasibility problem solved via stochastic dual subgradient descent and applying stochastic ADD in place of the stochastic gradient descent algorithm. The ABP algorithm is proven to guarantee queue stability throughout the network. Numerical experiments demonstrate a significant reduction in total packets queued at steady state.**

I. Introduction

This paper considers the problem of joint routing and scheduling in packet networks. Packets are accepted from upper layers as they are generated and marked for delivery to intended destinations. To accomplish delivery of information nodes need to determine routes and schedules capable of accommodating the generated traffic. From a node-centric point of view, individual nodes handle packets that are generated locally as well as packets received from neighboring nodes. The goal of each node is to determine suitable next hops for each flow conducive to successful packet delivery.

A joint solution to this routing and scheduling problem is offered by the Backpressure algorithm [1]. In Backpressure, nodes keep track of the number of packets in their local queues for each flow and share this information with neighboring agents. Nodes compute the differences between the number of packets in their queues and the number of packets in neighboring queues for all flows and assign the transmission capacity of the link to the flow with the largest queue differential. The term backpressure is used because the algorithm emulates the physical behavior of fluids under pressure. Regardless of interpretation and despite its simplicity, Backpressure can be proved to be an optimal policy in the following sense: if given arrival rates can be supported by some routing-scheduling policy, they can be supported by Backpressure. Notice that Backpressure relies on queue lengths only and does not need knowledge or estimation of packet arrival rates. It is also important to recognize that Backpressure is an iterative algorithm. Packets are initially routed more or less at random but as queues build throughout the network suitable routes and schedules are eventually learned.

The main drawback of Backpressure is the slow convergence rate of this iterative process for route discovery. This is better understood through an alternative interpretation of Backpressure as a dual stochastic subgradient descent algorithm [2], [3]. Joint routing and scheduling can be formulated as the determination of per-flow routing variables that satisfy link capacity and flow conservation constraints. In this model the packet transmission rates are abstracted as continuous variables. To solve the resulting feasibility problem we associate Lagrange multipliers with the flow conservation constraints and proceed to implement subgradient descent in the dual domain. This solution methodology leads to distributed implementations and for that reason is commonly utilized to find optimal operating points of wired [4]–[6] and wireless communication networks [7]–[9]. More important to the discussion here, the resulting algorithm turns out equivalent to Backpressure with queue lengths taking the place of noisy versions of the corresponding dual variables. The slow convergence rate of Backpressure is thus expected because the convergence rate of subgradient descent is of order $O(1/\sqrt{k})$ when measured with respect to the expected objective suboptimality of kth iterate, [10]. Simple modifications can speed up the convergence rate of Backpressure by rendering it equivalent to stochastic gradient descent [11] which has a convergence rate of order $O(1/k)$ for the expected suboptimality of kth iterate.

To reduce the convergences times for Backpressure, we need to incorporate information on the curvature of the dual function. This could be achieved by using Newton's method, but the determination of dual Newton steps requires coordination among all nodes in the network. To overcome this limitation, methods to determine approximate Newton steps in a distributed manner have been proposed. Early contributions on this regard are found in [12], [13]. Both of these methods, however, are not fully distributed because they require some level of global coordination. Efforts to overcome this shortcoming include approximating the Hessian inverse with the inverse of its diagonals [14], the use of consensus iterations to approximate the Newton step [15], [16], and the accelerated dual descent (ADD) family of algorithms that uses a Taylor expansion of the Hessian inverse to compute approximations to the Newton step [17]. Members of the ADD family are indexed by a parameter N indicating that local Newton step approximations are constructed at each node with information gleaned from agents no more than N hops away. ADD is proven to achieve an exponential convergence rate and demonstrated to reduce the time to find optimal operating points – as measured by the number of communication instances – by two orders of magnitude with respect to regular subgradient descent.

This paper adapts the ADD family of algorithms to develop variations of Backpressure that will always eventually clear the queues. This extends our previous work which used a Lyapunov drift based argument to show the queues do not become unbounded, [18]. We begin by introducing the problem of stabilizing queues in a communication network and review the Backpressure algorithm of [1] used to solve this problem – Section II. We proceed to demonstrate that the Backpressure algorithm is equivalent to stochastic subgradient descent – Section II-A. Restating the problem in an optimization framework, we review the generalization to Soft Backpressure – Section II-C. We construct an approximate stochastic dual Newton step using local information and define the Accelerated Backpressure algorithm

This research is supported by Army Research Lab MAST Collaborative Technology Alliance, AFOSR complex networks program, ARO P-57920-NS, NSF CAREER CCF-0952867, and NSF CCF-1017454, ONR BRC N00014-12-1-0997 and NSF-ECS-0347285. The authors are with the Department of Electrical and Systems Engineering, University of Pennsylvania. Address: 200 South 33^{rd} st. Philadelphia, PA, 19104. Phone: 215-898-9241. Fax: 215-573-2068. Email: {zargham, aribeiro, jadbabai}@seas.upenn.edu.

Algorithm 1: Backpressure at node i

for $t = 0, 1, 2, \cdots$ **do**
  Observe local queue lengths $\{q_i^k(t)\}_k$ for all flows k
  for *all neighbors* $j \in n_i$ **do**
    Send queue lengths $\{q_i^k(t)\}_k$ – Receive $\{q_j^k(t)\}_k$
    Determine flow with largest pressure:

$$k_{ij}^* = \operatorname*{argmax}_k \left[q_i^k(t) - q_j^k(t)\right]^+$$

    Set routing variables to $r_{ij}^k(t) = 0$ for all $k \neq k_{ij}^*$ and

$$r_{ij}^{k_{ij}^*}(t) = C_{ij}\mathbb{I}\left\{q_i^{k_{ij}^*}(t) - q_j^{k_{ij}^*}(t) > 0\right\}$$

    Transmit $r_{ij}^{k_{ij}^*}(t)$ packets for flow k_{ij}^*
  end
end

(ABP) which routes packets using the dual variables as queue priorities – Section III. Implementing the dual update with a decaying step size, we prove the queues are always eventually empty using a supermartingale argument – Section IV. Numerical experiments demonstrate that the ABP algorithm stabilizes with 60% shorter queues than the Backpressure algorithm of [1] and 25% shorter queues than the Soft Backpressure algorithm of [11] – Section V.

II. Preliminaries

Consider a given network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of links between nodes. Denote as C_{ij} the capacity of link $(i,j) \in \mathcal{E}$ and define the neighborhood of i as the set $n_i = \{j \in \mathcal{V} | (i,j) \in \mathcal{E}\}$ of nodes j that can communicate directly with i. There is also a set of information flows $\mathcal{K}$ with the destination of flow $k \in \mathcal{K}$ being the node $o_k \in \mathcal{V}$.

At time index t terminal $i \neq o_k$ generates a random number $a_i^k(t)$ packets to be delivered to o_k. The random variables $a_i^k(t) \geq 0$ are assumed independent and identically distributed across time with expected value $\mathbb{E}\left[a_i^k(t)\right] = a_i^k$. In the same time slot node i routes $r_{ij}^k(t) \geq 0$ units of information through neighboring node $j \in n_i$ and receives $r_{ji}^k(t) \geq 0$ packets from neighbor j.

The difference between the total number of received packets $a_i^k(t) + \sum_{j\in n_i} r_{ji}^k(t)$ and the sum of transmitted packets $\sum_{j\in n_i} r_{ij}^k(t)$ is added to the local queue – or subtracted if this quantity is negative. Therefore, the number $q_i^k(t)$ of k-flow packets queued at node i evolves according to

$$q_i^k(t+1) = \left[q_i^k(t) + a_i^k(t) + \sum_{j\in n_i} r_{ji}^k(t) - r_{ij}^k(t)\right]^+, \quad (1)$$

where the projection $[\cdot]^+$ into the nonnegative reals is because the number of packets in queue cannot become negative. We remark that (1) is stated for all nodes $i \neq o_k$ because packets routed to their destinations are removed from the system.

To ensure packet delivery it is sufficient to guarantee that all queues $q_i^k(t+1)$ remain stable. In turn, this can be guaranteed if the average rate at which packets exit queues is less than the rate at which packets are loaded into them. To state this formally observe that the time average limit of arrivals satisfies $\lim_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^t a_i^k(\tau) = \mathbb{E}\left[a_i^k(t)\right] := a_i^k$ and define the ergodic limit $r_{ij}^k := \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^t r_{ij}^k(\tau)$. If the processes $r_{ij}^k(t)$ controlling the movement of information through the network are asymptotically stationary, queue stability follows if

$$\sum_{j\in n_i} r_{ij}^k - r_{ji}^k \geq a_i^k + \xi \quad \forall\ k, i \neq o_k \quad (2)$$

where $\xi > 0$ is a small constant. For future reference, define the vector $r := \{r_{ij}^k\}_{k,i\neq o_k,j}$, built by stacking all the routing variables r_{ij}^k. Since at most C_{ij} packets can be transmitted by the link (i,j) the routing variables $r_{ij}^k(t)$ always satisfy

$$\sum_k r_{ij}^k(t) \leq C_{ij} \quad (3)$$

We emphasize that (3) holds for all times whereas (2) holds in terms of time averages. The joint routing and scheduling problem can be formally stated as the determination of nonnegative variables $r_{ij}^k(t) \geq 0$ that satisfy (3) for all times t and whose time average limits r_{ij}^k satisfy (2). The Backpressure algorithm solves this problem by assigning all the capacity of the link (i,j) to the flow with the largest queue differential $q_i^k(t) - q_j^k(t)$. Specifically, for each link we determine the flow pressure

$$k_{ij}^* = \operatorname*{argmax}_k \left[q_i^k(t) - q_j^k(t)\right]^+. \quad (4)$$

If the maximum pressure $\max_k \left[q_i^k(t) - q_j^k(t)\right]^+ > 0$ is strictly positive we set $r_{ij}^k(t) = C_{ij}$ for $k = k_{ij}^*$. Otherwise the link remains idle during the time frame. The resulting algorithm is summarized in Algorithm 1. The Backpressure algorithm works by observing the queue differentials on each link and then assigning the capacity for each link to the data type with the largest positive queue differential, thus driving the time average of the queue differentials to zero and stabilizing the queues as a consequence. For the generalizations developed in this paper it is necessary to reinterpret Backpressure as a dual stochastic subgradient descent as we do in the following section.

A. Dual stochastic subgradient descent

Since the parameters that are important for queue stability are the time averages r_{ij}^k of the routing variables $r_{ij}^k(t)$ an alternative view of the joint routing and scheduling problem is the determination of variables r_{ij}^k satisfying (2) and (3). This can be formulated as the solution of an optimization problem. Let $f_{ij}^k(r_{ij}^k)$ be arbitrary concave functions on $\mathbb{R}_+$ and consider the optimization problem

$$\begin{aligned} r^* := \operatorname{argmax} & \sum_{k,i\neq o_k,j} f_{ij}^k(r_{ij}^k) \\ \text{s.t.} & \sum_{j\in n_i} r_{ij}^k - r_{ji}^k \geq a_i^k + \xi, \quad \forall\ k, i \neq o_k, \\ & \sum_{k\in\mathcal{K}} r_{ij}^k \leq C_{ij}, \qquad\qquad \forall\ (i,j) \in \mathcal{E}. \end{aligned} \quad (5)$$

where the optimization is over nonnegative variables $r_{ij}^k \geq 0$. Since only feasibility is important for queue stability, solutions to (5) ensure stable queues irrespectively of the objective functions $f_{ij}^k(r_{ij}^k)$.

Since the problem in (5) is concave, it has null duality gap and can be solved in dual domain using a descent algorithm. We associate the multipliers λ_i^k with the constraints $\sum_{j\in n_i} r_{ij}^k - r_{ji}^k \geq a_i^k + \xi$ and keep the constraints $\sum_k r_{ij}^k \leq C_{ij}$ implicit.

The Lagrangian associated with the optimization problem in (5) is then defined as

$$\mathcal{L}(r,\lambda) = \sum_{k,i\neq o_k,j} f_{ij}^k(r_{ij}^k) + \sum_{k,i\neq o_k} \lambda_i^k \Big(\sum_{j\in n_i} r_{ij}^k - r_{ji}^k - a_i^k - \xi \Big), \tag{6}$$

where we have introduced the vector $\lambda := \{\lambda_i\}_{i=1}^n$ stacking the n local multipliers $\lambda_i := \{\lambda_i^k\}_{k:i\neq o_k}$ which in turn stack all the multipliers at node i that are associated with different flows. The Lagrange dual function is defined as

$$h(\lambda) := \max_{\sum_k r_{ij}^k \leq C_{ij}} \mathcal{L}(r,\lambda). \tag{7}$$

To compute a descent direction for $h(\lambda)$ define the primal Lagrangian maximizers as a function of λ,

$$R_{ij}^k(\lambda) := \operatorname*{argmax}_{\sum_k r_{ij}^k \leq C_{ij}} \mathcal{L}(r,\lambda). \tag{8}$$

A descent direction for the dual function is available in the form of the dual subgradient whose components $\bar{g}_i^k(\lambda)$ are obtained by evaluating the constraint slack associated with the Lagrangian maximizers

$$\bar{g}_i^k(\lambda) := \sum_{j\in n_i} R_{ij}^k(\lambda) - R_{ji}^k(\lambda) - a_i^k - \xi. \tag{9}$$

Since the Lagrangian $\mathcal{L}(r,\lambda)$ in (6) is separable in the routing functions $R_{ij}^k(\lambda)$ the determination of the maximizers $R_{ij}^k(\lambda) := \operatorname{argmax}_{\sum_k r_{ij}^k \leq C_{ij}} \mathcal{L}(r,\lambda)$ can be decomposed into the maximization of separate summands. Considering the coupling constraints $\sum_k r_{ij}^k \leq C_{ij}$ it suffices to consider variables $\{r_{ij}^k\}_k$ for all flows across a given link. After reordering terms it follows that we can compute the routing on each edge (i,j),

$$\begin{aligned} R_{ij}^k(\lambda) = \operatorname{argmax} & \sum_k f_{ij}^k(r_{ij}^k) + r_{ij}^k\big(\lambda_i^k - \lambda_j^k\big) \\ \text{s.t.} & \sum_{k\in\mathcal{K}} r_{ij}^k \leq C_{ij}. \end{aligned} \tag{10}$$

Introducing a time index t, subgradients $\bar{g}_i^k(\lambda(t))$ could be computed using (9) with Lagrangian maximizers $R_{ij}^k(\lambda(t))$ given by (10). A subgradient descent iteration could then be defined to find the variables r^* that solve (5); see e.g., [19].

The problem in computing $\bar{g}_i^k(\lambda)$ is that we don't know the average arrival rates a_i^k. We do observe, however, the instantaneous rates $a_i^k(t)$ that are known to satisfy $\mathbb{E}\left[a_i^k(t)\right] = a_i^k$. Therefore,

$$g_i^k(\lambda,t) := \sum_{j\in n_i} R_{ij}^k(\lambda) - R_{ji}^k(\lambda) - a_i^k(t) - \xi, \tag{11}$$

is a stochastic subgradient of the dual function in the sense that its expected value $\mathbb{E}\left[g_i^k(\lambda,t)\right] = \bar{g}_i^k(\lambda)$ is the subgradient defined in (9). We can then minimize the dual function using a stochastic subgradient descent algorithm. At time t we have multipliers $\lambda(t)$ and determine Lagrangian maximizers $r_{ij}^k(t) = R_{ij}^k(\lambda(t))$ as per (10). We then proceed to determine the stochastic subgradient $g_i^k(t) = g_i^k(\lambda(t),t)$ using (11) and update multipliers along the stochastic subgradient direction,

$$\lambda_i^k(t+1) = \Big[\lambda_i^k(t) - \alpha_t g_i^k(\lambda(t),t)\Big]^+ = \Big[\lambda_i^k(t) - \alpha_t g_i^k(t)\Big]^+, \tag{12}$$

where the projection $[\cdot]^+$ into the nonnegatives is to ensure the dual variables stay feasible and α_t is a stepsize sequence appropriately chosen so as to ensure convergence; see e.g., [11]. For future reference define the local vector $g_i(t) := \{g_i^k(t)\}_{k:i\neq o_k}$ that stacks all the stochastic subgradients at node i that are associated with different flows and the global vector $g(t) := \{g_i(t)\}_{i=1}^n$ that stacks the n local stochastic subgradient vectors $g_i(t)$.

Algorithm 2: Backpressure as stochastic subgradient descent

Observe $q_i^k(0)$. Initialize $\lambda_i^k(0) = q_i^k(0)$ for all k and $i \neq o_k$
for $t = 0, 1, 2, \cdots$ **do**
  for *all neighbors* $j \in n(i)$ **do**
    Send duals $\{\lambda_i^k(t)\}_k$ – Receive duals $\{\lambda_j^k(t)\}_k$
    Determine flow with largest dual variable differential:

$$k_{ij}^* = \operatorname*{argmax}_k \Big[\lambda_i^k(t) - \lambda_j^k(t)\Big]^+$$

    Set routing variables to $r_{ij}^k(t) = 0$ for all $k \neq k_{ij}^*$ and

$$r_{ij}^{k_{ij}^*}(t) = C_{ij}\mathbb{I}\Big\{\lambda_i^{k_{ij}^*}(t) - \lambda_j^{k_{ij}^*}(t) > 0\Big\}$$

    Transmit $r_{ij}^{k_{ij}^*}(t)$ packets for flow k_{ij}^*
  end
  Send variables $\{r_{ij}^k(t)\}_{kj}$ – Receive variables $\{r_{ji}^k(t)\}_{kj}$
  Update multipliers $\{\lambda_i^k(t)\}_k$ along stochastic subgradient

$$\lambda_i^k(t+1) = \Big[\lambda_i^k(t) - \alpha_t\Big(\sum_{j\in n(i)} r_{ij}^k(t) + r_{ji}^k(t) - a_i^k(t) - \xi\Big)\Big]^+$$

end

Substituting $g_i^k(t)$ for its explicit expression in (11) with $r_{ij}^k(t) = R_{ij}^k(\lambda(t))$ yields the dual variable update

$$\lambda_i^k(t+1) = \Big[\lambda_i^k(t) - \alpha_t\Big(\sum_{j\in n_i} r_{ij}^k(t) - r_{ji}^k(t) - a_i^k(t) - \xi\Big)\Big]^+, \tag{13}$$

which, except for the presence of the step size α_t and the constant ξ, is equivalent to the queue update in (1). Properties of the descent algorithm in (13) vary with the selection of the functions $f_{ij}^k(r_{ij}^k)$. Two cases of interest are when $f_{ij}^k(r_{ij}^k) = 0$ and when $f_{ij}^k(r_{ij}^k)$ are continuously differentiable, strongly concave, and monotone decreasing on $\mathbb{R}_+$ but otherwise arbitrary. The former allows an analogy with the backpressure as given by Algorithm 1 while the latter leads to a variation termed Soft Backpressure.

B. Backpressure as stochastic subgradient descent

The classical Backpressure algorithm, [1] can be recovered by setting $f_{ij}^k(r_{ij}^k) = 0$ for all links flows k and links (i,j). With this selection the objective to be maximized in (10) becomes $\sum_k r_{ij}^k \left(\lambda_i^k(t) - \lambda_j^k(t)\right)$. To solve this maximization it suffices to find the flow with the largest dual variable differential

$$k_{ij}^* = \operatorname*{argmax}_k \left[\lambda_i^k(t) - \lambda_j^k(t)\right]^+. \tag{14}$$

If the value of the corresponding maximum is nonpositive, i.e., $\max_k \left[\lambda_i^k(t) - \lambda_j^k(t)\right]^+ \leq 0$, all summands in (10) are nonpositive and the largest objective in (10) is attained by making $r_{ij}^k(t) = 0$ for all flows k. Otherwise, since the sum of routing variables $r_{ij}^k(t)$ must satisfy $\sum_{k\in\mathcal{K}} r_{ij}^k \leq C_{ij}$ the maximum objective is attained by making $r_{ij}^k(t) = C_{ij}$ for $k = k_{ij}^*$ and $r_{ij}^k(t) = 0$ for all other k.

The algorithm that follows from the solution of this maximization is summarized in Algorithm 2. The dual stochastic subgradient descent is implemented using node level protocols.

Algorithm 3: Soft Backpressure at node i

1 Observe $q_i^k(0)$. Initialize $\lambda_i^k(0) = q_i^k(0)$ for all k and $i \neq o_k$
2 **for** $t = 0, 1, 2, \cdots$ **do**
3 **for** *all neighbors* $j \in n(i)$ **do**
4 Send duals $\{\lambda_i^k(t)\}_k$ – Receive duals $\{\lambda_j^k(t)\}_k$
5 Compute μ_{ij} such that

$$\sum_k F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda) \right]^+ \right) = C_{ij}$$

6 Transmit packets at rate

$$r_{ij}^k(t) = F_{ij}^k \left(-[\lambda_i^k(t) - \lambda_j^k(t) - \mu_{ij}]^+ \right)$$

7 **end**
8 Send variables $\{r_{ij}^k(t)\}_{kj}$ – Receive variables $\{r_{ji}^k(t)\}_{kj}$
9 Update multipliers $\{\lambda_i^k(t)\}_k$ along stochastic subgradient

$$\lambda_i^k(t+1) = \left[\lambda_i^k(t) - \alpha_t \left(\sum_{j \in n_i} r_{ij}^k(t) - r_{ji}^k(t) - a_i^k(t) - \xi \right) \right]^+$$

10 **end**

At each time instance nodes send their multipliers $\lambda_i^k(t)$ to their neighbors. After receiving multiplier information from its neighbors, each node can compute the multiplier differentials $\lambda_i^k(t) - \lambda_j^k(t)$ for each edge. The node then allocates the full capacity of each outgoing link to whichever commodity has the largest differential. Once the transmission rates $r_{ij}^k(t)$ are set they are used for communication. Nodes also share these variables as shown in Step 7 so that they can be used to compute the stochastic gradient in (11). This stochastic gradients are used to update the multipliers as shown in Step 8 [cf. (13)].

Comparing (4) with (14) we see that the assignments of flows to links are the same if we identify multipliers $\lambda_i^k(t)$ with queue lengths. Furthermore, if we consider the Lagrangian update in (13) with $\alpha_t = 1$ for all times t and $\xi = 0$ we see that they too coincide. Thus, we can think of Backpressure as a particular case of stochastic subgradient descent. From that point of view algorithms 1 and 2 are identical. The only difference is that since queues take the place of multipliers the update in Step 8 of Algorithm 2 is not necessary in Algorithm 1. The exchange of routing variables $\{r_{ij}^k(t)\}_{kj}$ in Step 7 that is necessary to implement Step 8 is not needed as well.

C. Soft Backpressure

Assume that the functions $f_{ij}^k(r_{ij}^k)$ are continuously differentiable, strongly convcave, and monotone decreasing on $\mathbb{R}_+$. In this case the derivatives $y = \partial f_{ij}^k(x)/\partial x \leq 0$ for all $x \geq 0$ and thus have inverse functions that we denote as

$$F_{ij}^k(y) := \left[\partial f_{ij}^k(x)/\partial x \right]^{-1}(y) \geq 0, \, \forall y \leq 0. \tag{15}$$

The Lagrangian maximizers in (10) can be explicitly written in terms of the derivative inverses $F_{ij}^k(y)$. Furthermore, the maximizers are unique for all λ implying that the dual function is differentiable. We detail these two statements in the following proposition.

Proposition 1. *If the functions $f_{ij}^k(r_{ij}^k)$ in (5) are continuously differentiable, strongly convcave, and monotone decreasing on $\mathbb{R}_+$, the dual function $h(\lambda) := \max_{\sum_k r_{ij}^k \leq C_{ij}} \mathcal{L}(r, \lambda)$ is differentiable for all λ. Furthermore, the gradient component along the λ_i^k direction is $\bar{g}_i^k(\lambda)$ as defined in (11) with*

$$R_{ij}^k(\lambda) = F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda) \right]^+ \right). \tag{16}$$

where $\mu_{ij}(\lambda)$ is either 0 if $\sum_k F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k \right]^+ \right) \leq C_{ij}$ or chosen as the solution to the equation

$$\sum_k F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda) \right]^+ \right) = C_{ij}. \tag{17}$$

Proof. The dual problem (7) is convex because it is the Lagrange dual of a concave maximization problem. Since $f(r)$ is strongly concave, the maximizers $R(\lambda)$ generated by (10) are unique. Thus $\bar{g}(\lambda)$ is unique for any λ, ensuring that $h(\lambda)$ is differentiable, appendix A.4 [20].

To compute the maximizers $R(\lambda)$, we consider the primal optimization (10). Taking the Lagrange dual of the domain constraints yields the extended Lagrangian,

$$\bar{\mathcal{L}}(r, \mu) = \sum_k f_{ij}^k(r_{ij}^k) + r_{ij}^k(\lambda_i^k - \lambda_j^k) + \mu_{ij}(C_{ij} - \sum_k r_{ij}^k). \tag{18}$$

Considering the Karush-Kuhn-Tucker (KKT) optimality conditions as defined in [20][Section 5.5] for (18) yields the equations

$$f'_{ij,k}(r_{ij}^k) = - \left[\lambda_i^k - \lambda_j^k - \mu_{ij} \right]^+ \tag{19}$$

$$\sum_k r_{ij}^k \leq C_{ij} \tag{20}$$

for all $(i, j) \in \mathcal{E}$ where $f'_{ij,k}(x) = \partial f_{ij}^k(x)/\partial x$. Applying the definition of $F_{ij}^k(\cdot)$ from equation (15) to (19) we get the desired relation in (16). It remains to enforce (20) by selection of μ_{ij} which gives us condition (17). The assertion that

$$\mu_{ij} = 0 \text{ when } \sum_k F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k \right]^+ \right) \leq C_{ij} \tag{21}$$

holds by complementary slackness, [20, Section 5.5]. ■

While (17) does not have a closed form solution it can be computed quickly numerically using a binary search because it is a simple single variable root finding problem. We operate under the assumption that computation time cost is small compared to communication time cost.

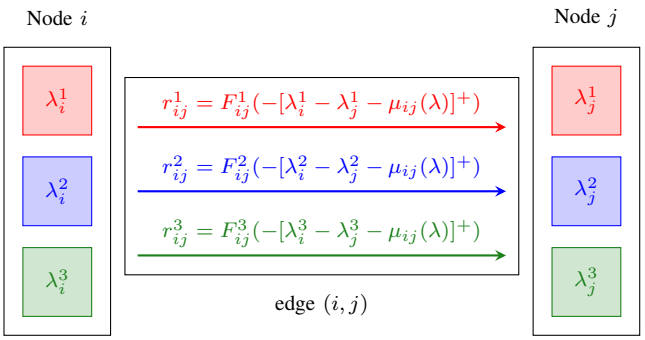

Fig. 1. Flow variables r_{ij}^k encoding packet transmission rates are computed from dual variables at nodes i and j. For Backpressure and Soft Backpressure the variables λ_i^k are equivalent to the queue lengths q_i^k. For Accelerated Backpressure, λ_i^k is a priority rating for q_i^k. The constant $\mu_{ij}(\lambda)$ is chosen so that the total rate $\sum_k r_{ij}^k$ stays below the edge's capacity C_{ij}.

The result in Proposition 1 combined with the stochastic subgradient descent iteration in (13) yields the Soft Backpressure algorithm summarized in Algorithm 3. Soft Backpressure is implemented using node level protocols. At each time instance nodes send their multipliers $\lambda_i^k(t)$ to their neighbors. After receiving multiplier information from its neighbors, each node can compute the multiplier differentials $\lambda_i^k(t) - \lambda_j^k(t)$ for each edge. The nodes then solve for μ_{ij} on each of its outgoing edges to satisfy (17). The capacity of each edge is then allocated to the different commodities using the expression in (16). With the transmission rates $\{r_{ij}^k(t)\}_{kj}$ set, we proceed as in regular Backpressure. The communication rates are used for communication and also shared with neighbors as shown in Step 7. These variables determine the stochastic gradient in (11) which is used to update the multipliers as shown in Step 8 [cf. (13)].

Algorithm 3 is called Soft Backpressure because rather than allocating all of an edge's capacity to the data type with the largest pressure $\lambda_i^k - \lambda_j^k$, it divides the capacity among the different data types based on their respective pressures via (16). This expression has a reverse water filling interpretation; see [11]. We further observe that Soft Backpressure is solved using the same information required of Backpressure, the lengths of the queues for all data types on either end of the link whose flow you are computing. This means that assuming a step size $\alpha_t = 1$ and constant $\xi = 0$, Algorithm 3 can be implemented without computing the multipliers but rather by simply observing the queue lengths $q_i^k(t)$ which are equal to the dual variables $\lambda_i^k(t)$.

An important difference between Backpressure and Soft Backpressure is that the former is equivalent to stochastic *subgradient* descent whereas the latter is tantamount to stochastic *gradient* descent because the dual function is differentiable. This improves the average convergence rate from $O(1/\sqrt{k})$ to $O(1/k)$. In practice, Soft Backpressure is still slow to converge, motivating the ABP algorithm that we introduce in the following section.

III. Accelerated Backpressure

The backpressure-type algorithms of Section II exhibit slow convergence because they use rate of change information, but have no knowledge of curvature. In twice differentiable centralized deterministic problems, second order methods such as Newton method are used to take advantage of curvature information. Direct application of Newton-type methods is not possible here because: (i) the dual function is not twice differentiable; (ii) only a stochastic gradient can be observed in accordance with (11); (iii) the Newton direction requires data gathering and processing at a central node. We solve (i) by introducing a generalized Hessian; (ii) is resolved by the fact that the generalized Hessian is deterministic in this case and (iii) is resolved by using a Taylor's expansion of the Hessian inverse to approximate the Newton direction.

A. The Generalized Hessian

The dual function is not twice differentiable because the Lagrange dual defined in (7) has domain constraints which cause lack of smoothness in its gradient – see propositions 2 and 3. However, the second order behavior can be characterized by a *Generalized Hessian*, which is analagous to a subgradient because it may be taken from a set of values defined by the convex hull of the left and right limits approaching the points of discontinuity.

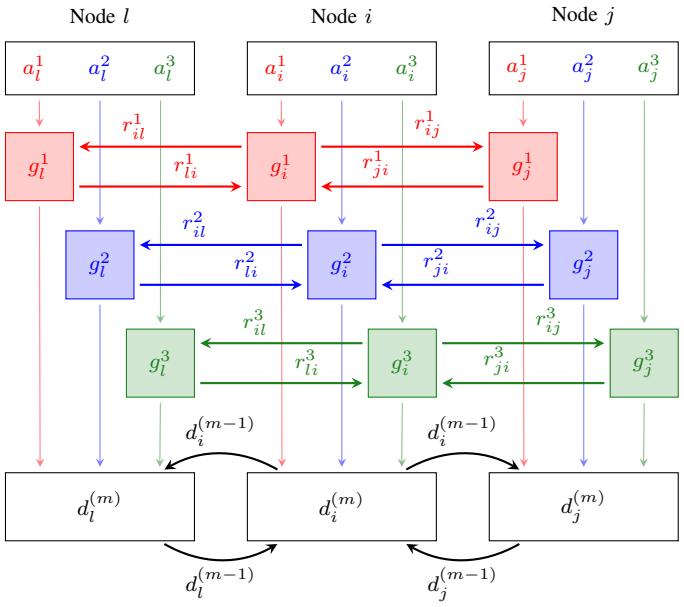

Fig. 2. The Update of the queue priorities λ_i^k is determined by the value of the gradient component $g_i^k = \sum_{j\in n_i} r_{ji}^k - r_{ij}^k - a_i^k - \xi$ which is determined by the values of local and neighboring routing variables as well as the arrival rates observed from upper layers. In Backpressure and Soft Backpressure the gradient is just subtracted from λ_i^k. In Accelerated Backpressure the gradient is used along with local curvature information to determine the approximate Newton direction $d_i^{(N)}$. This is accomplished by the embedded communication loop that we summarize in Algorithm 5.

Definition 1 (Generalized Hessian [21]). *We say that $H(\lambda)$ is a generalized Hessian of the dual function $h(\lambda) = \mathcal{L}(R(\lambda),\lambda)$ if and only if*

$$H(\lambda) \in \partial^2 \mathcal{L}(R(\lambda),\lambda) = co\left\{ \lim_{\bar{\lambda}\to\lambda, \bar{\lambda}\in D_g} \nabla^2 \mathcal{L}(R(\lambda),\lambda) \right\} \quad (22)$$

where $co\{\cdot\}$ denotes convex hull and D_g is the set of points on which $\mathcal{L}(R(\lambda),\lambda)$ is twice differentiable.

Writing $H(\lambda)$ in block form we have,

$$H(\lambda) = \begin{bmatrix} H_{11}(\lambda) & H_{12}(\lambda) & \cdots & H_{1n}(\lambda) \\ H_{21}(\lambda) & H_{22}(\lambda) & \dots & H_{2n}(\lambda) \\ \vdots & \vdots & \ddots & \vdots \\ H_{n1}(\lambda) & H_{n2}(\lambda) & \cdots & H_{nn}(\lambda) \end{bmatrix}. \quad (23)$$

The matrix (23) has $n \times n$ blocks where each of the blocks is a square matrix with dimension K equal to the number of commodity types. The elements of the $(i,j)^{th}$ block are explicitly given by

$$\Big[H_{ij}(\lambda)\Big]_{kl} := \frac{\partial^2 h(\lambda)}{\partial \lambda_i^k \partial \lambda_j^l} = \frac{\partial^2 \mathcal{L}(R(\lambda),\lambda)}{\partial \lambda_i^k \partial \lambda_j^l}, \quad (24)$$

where we expressed the dual function $h(\lambda) = \mathcal{L}(R(\lambda),\lambda)$ in terms of the Lagrangian maximizers $R(\lambda)$ in (8) to write the second equality. As per (24), the $(i,j)^{th}$ block H_{ij} is associated with second derivatives with respect to dual variables of nodes i and j. The $(k,s)^{th}$ element of each block is associated with second derivatives with respect to commodities k and s.

Implicit in (24) is the differentiation of $R(\lambda)$ with respect to λ. It can be seen from (16) that $R_{ij}^k(\lambda)$ is not differentiable at the points where $\mu_{ij}(\lambda) = \lambda_i^k - \lambda_j^k$. These points arise precisely when the capacity constraint on edge (i,j) becomes active.

We show that D_g is a dense set by considering differentiability of $R_{ij}^k(\lambda)$ in two cases. First, consider the case with λ such that $\sum_k R_{ij}^k(\lambda) = C_{ij}$, then according to Proposition 1, $R_{ij}^k(\lambda)$ is may not be differentiable when $\lambda_i^k - \lambda_j^k = \mu_{ij}(\lambda)$, due to the $[\cdot]^+$ function. According to (15), when $\lambda_i^k - \lambda_j^k = \mu_{ij}(\lambda)$, we have $R_{ij}^k(\lambda) = F_{ij}^k(0) = \max_x f_{ij}^k(x)$. By monotonicity (decreasing) of $f_{ij}^k(x)$ on $\mathbb{R}^+$, the optimal routing is the function $R_{ij}^k(\lambda) = 0$ which is differentiable with $\nabla R_{ij}^k(\lambda) = 0$ for any λ such that $\lambda_i^k - \lambda_j^k < \mu_{ij}(\lambda)$. Therefore, given $\lambda_i^k - \lambda_j^k = \mu_{ij}(\lambda)$ when can choose $\bar{\lambda}$ with $\bar{\lambda}_i^k = \lambda_i^k - \epsilon$ for sufficiently small $\epsilon > 0$ and $\bar{\lambda}_j^k = \lambda_j^k$ for all $j \neq i$ ensuring $R_{ij}^k(\bar{\lambda})$ is differentiable.

Second, assume we have λ such that $\sum_k R_{ij}^k(\lambda) < C_{ij}$. Then according to Proposition 1, $\mu_{ij} = 0$ and $R_{ij}^k(\lambda)$ is differentiable as long as $\lambda_i^k \neq \lambda_j^k$. If $\lambda_i = \lambda_j$, there is a $\bar{\lambda}$ with $\bar{\lambda}_i^k = \lambda_i^k + \epsilon$ and $\bar{\lambda}_j = \lambda_j$ for all $i \neq j$ which is differentiable for ϵ small enough to ensure $\sum_k R_{ij}^k(\bar{\lambda}) < C_{ij}$; by continuity of $F_{ij}^k(x)$ a sufficiently small ϵ exists. Furthermore, when $\mu_{ij} = 0$ and $\lambda_i^k < \lambda_j^k$, we have $\lambda_i^k - \lambda_j^k < \mu_{ij}(\lambda)$ and $R_{ij}^k(\lambda) = 0$ which yields the property that the discontinuities all occur between non-trivial functions of λ and the zero function.

From Proposition 1, if the capacity constraint is not active, the gradient $\nabla_\lambda R_{ij}^k(\lambda)$ is a non-trivial function of λ, but when the capacity constraints becomes active $\nabla_\lambda R_{ij}^k(\lambda)$ is zero. Since the points of discontinuity are transitions between the nonzero gradients and zero gradients $\nabla_\lambda R_{ij}^k(\lambda) = 0$ the definition of the set of Generalized Hessians in (22) allows us to select the null derivative at these points to generate a specific Generalized Hessian.

We proceed to compute this specific Generalized Hessian. The set of commodities that are active across a particular edge are important. Thus, for given λ and edge $(i,j) \in \mathcal{E}$ define

$$\mathcal{K}_{ij}(\lambda) = \{k \in \mathcal{K} : R_{ij}^k(\lambda) > 0\}, \tag{25}$$

as the set of commodities that are being transported across given edge (i,j) for multipliers λ. Further define the capacity sharing coefficients

$$s_{ij}(\lambda) = \begin{cases} 1/|\mathcal{K}_{ij}(\lambda)| & \text{if } \mu_{ij}(\lambda) > 0, \\ 0 & \text{if } \mu_{ij}(\lambda) = 0, \end{cases} \tag{26}$$

which take the value zero when the capacity constraint on edge (i,j) is inactive, i.e., when $\sum_k r_{ij}^k(\lambda) < C_{ij}$, and equals the inverse cardinality $1/|\mathcal{K}_{ij}(\lambda)|$ of the set $\mathcal{K}_{ij}(\lambda)$ when the link capacity is saturated. Using these definitions computing the generalized Hessian $H(\lambda)$ is a straightforward but cumbersome derivation that we relegate to Appendix A. The results are stated in the following two propositions that concern the off-diagonal blocks $H_{ii}(\lambda)$ and diagonal blocks $H_{ii}(\lambda)$, respectively.

Proposition 2. *Let $H_{ij}(\lambda)$ denote an off diagonal block of the generalized Hessian $H(\lambda)$ in* (23) *associated with the dual function $h(\lambda)$ defined in* (7) *and denote the derivative of the inverse derivative function $F_{ij}^k(\lambda)$ of* (15) *as $F'_{ij,k} = \partial F_{ij}^k(\lambda)/\partial\lambda$. The kth diagonal element of the Hessian block $H_{ij}(\lambda)$ is given by*

$$\begin{aligned}\Big[H_{ij}(\lambda)\Big]_{kk} \;=\; & F'_{ij,k}\Big[R_{ij}^k(\lambda)\Big]\mathbf{1}\Big[k \in \mathcal{K}_{ij}(\lambda)\Big]\Big[1 - s_{ij}(\lambda)\Big] \\ & + F'_{ji,k}\Big[R_{ji}^k(\lambda)\Big]\mathbf{1}\Big[k \in \mathcal{K}_{ji}(\lambda)\Big]\Big[1 - s_{ji}(\lambda)\Big].\end{aligned} \tag{27}$$

where $\mathcal{K}_{ij}(\lambda)$ is the set of active commodities in the link $i \to j$ defined in (25) *and $s_{ij}(\lambda)$ is the capacity sharing coefficient in* (26)*. The (k,l)th off diagonal element of $H_{ij}(\lambda)$ can be written as*

$$\begin{aligned}\Big[H_{ij}(\lambda)\Big]_{kl} \;=\; & -F'_{ij,k}\Big[R_{ij}^k(\lambda)\Big]\mathbf{1}\Big[k, l \in \mathcal{K}_{ij}(\lambda)\Big]s_{ij}(\lambda) \\ & - F'_{ji,k}\Big[R_{ji}^k(\lambda)\Big]\mathbf{1}\Big[k, l \in \mathcal{K}_{ji}(\lambda)\Big]s_{ji}(\lambda).\end{aligned} \tag{28}$$

Proof. See Appendix A. ■

Proposition 3. *The diagonal blocks $H_{ii}(\lambda)$ of the generalized Hessian $H(\lambda)$ in* (23) *associated with the dual function $h(\lambda)$ defined in* (7) *are given by the sums of the off diagonal blocks $H_{ij}(\lambda)$, i.e.,*

$$H_{ii}(\lambda) \;=\; \sum_j H_{ij}(\lambda) \;=\; \sum_{j \in n_i} H_{ij}(\lambda). \tag{29}$$

Proof. See Appendix A. ■

The expression in (29) reduces computation of the diagonal blocks of $H(\lambda)$ to the computation of the off diagonal blocks. Their validity endows the Hessian of the dual function with an interpretation as a weighted block Laplacian. The expressions in (27) and (28) look complicated but are actually simple. The terms of the form $F'_{ij,k}(R_{ij}^k(\lambda))$ are just the derivatives of the inverse derivative function in (15). If, e.g., we use quadratic functions $f_{ij}^k(R_{ij}^k(\lambda)) = -(R_{ij}^k(\lambda))^2/2$ we just have $F'_{ij,k}(R_{ij}^k(\lambda)) = -1$.

The indicator function $\mathbf{1}(k \in \mathcal{K}_{ij}(\lambda))$ in (27) simply distinguishes between the values of λ for which commodity k is transported from i to j from those for which it is not. The term $F'_{ij,k}(R_{ij}^k(\lambda))(1 - s_{ij}(\lambda))$ is added to the Hessian component $[H_{ij}]_{kk}$ if and only if the commodity k is active in the link $i \to j$. Likewise the term $F'_{ji,k}(R_{ji}^k(\lambda))(1 - s_{ji}(\lambda))$ is added to Hessian component $[H_{ij}]_{kk}$ if and only if the commodity k is active in the link $j \to i$. The variables s_{ij} simply distinguish the case when $\mu_{ij}(\lambda)$ is null from when it is not, or, equivalently, the case when some of the link capacity is left unused from the case when all of the link capacity is allocated to some node. If some capacity is unused in the link $i \to j$ we have $\mu_{ij}(\lambda) = 0$ and the derivative $F'_{ij,k}(R_{ij}^k)$ is not scaled. If all of the capacity in the link $i \to j$ is used, the derivative $F'_{ij,k}(R_{ij}^k)$ is scaled by $(1 - 1/|\mathcal{K}_{ij}(\lambda)|)$.

Likewise, the indicator functions $\mathbf{1}(k, l \in \mathcal{K}_{ij}(\lambda))$ and $\mathbf{1}(k, l \in \mathcal{K}_{ji}(\lambda))$ in (28) imply that the corresponding terms are added to the Hessian component $[H_{ij}]_{kl}$ only if both commodities, k and l are active in the link $i \to j$ or the link $j \to i$. If some capacity is left unused in the link $i \to j$ we have $s_{ij}(\lambda) = 0$ and nothing is added to the Hessian component $[H_{ij}]_{kl}$. If all of the capacity is used we scale the derivative $F'_{ij,k}(R_{ij}^k(\lambda))$ by the inverse of the number of active commodities on the link, $1/|\mathcal{K}_{ij}(\lambda)|$. Likewise, we add nothing to $[H_{ij}]_{kl}$ if some capacity is left unused in the link $j \to i$ and scale $F'_{ji,k}(R_{ji}^k(\lambda))$ by the inverse of the number of active commodities on the link $j \to i$ otherwise.

Propositions 2 and 3 define a generalized Hessian for the dual function $h(\lambda)$ defined in (7). The components of the Hessian depend on the primal Lagrangian maximizers $R_{ij}^k(\lambda)$ as well as the auxiliary variable $\mu_{ij}(\lambda)$. As it follows from Proposition 1, both of these quantities depend on the dual variable λ but are independent of the arrival rates a_i^k. This is because the gradient dependence on the arrival rates is an additive constant [cf. (9)]. This simple fact is of fundamental importance here because it allows exact computation of the generalized Hessian $H(\lambda)$ and the

consequent implementation of a stochastic Newton algorithm in which generalized Hessian inverses $H^{-1}(\lambda)$ premultiply stochastic gradients $g_i^k(\lambda)$ instead of (regular) gradients $\bar{g}_i^k(\lambda)$. Indeed, consider time index t and current iterate $\lambda(t)$ and recall that $g(t)$ stands for the stochastic subgradient with components given by (11). The stochastic Newton method replaces the dual update in (12) by the recursion

$$\lambda(t+1) = \left[\lambda(t) - \alpha_t H^{-1}(t) g(t)\right]^+, \tag{30}$$

where we have used $H^{-1}(t) = H^{-1}(\lambda(t))$ to denote the inverse of the generalized Hessian evaluated at $\lambda(t)$. The deterministic nature of the generalized Hessian $H^{-1}(t)$ when $\lambda(t)$ is given permits the definition of the dual stochastic Newton algorithm in (30). This possibility notwithstanding, implementation of the stochastic Newton method in (30) requires centralized computation of the generalized Hessian inverses $H^{-1}(t)$. The remaining challenge addressed by ABP is to approximate the inversion of $H(\lambda)$ through local operations. We discuss this in the following section.

B. Distributed Approximation of the stochastic Newton Step

We begin by pointing out that the generalized Hessian in (23) with blocks as explicitly given in propositions 2 and 3 is block sparse with a sparsity pattern that matches the adjacency matrix of the network. We state this in the following lemma along with the fact that the diagonal blocks $H_{ii}(\lambda)$ are positive semidefinite.

Lemma 1. *The Dual Hessian, $H(\lambda)$ in* (23) *associated with the dual function $h(\lambda)$ defined in* (7) *is block sparse with respect to the graph $\mathcal{G}$, i.e.,*

$$H_{ij}(\lambda) = \mathbf{0} \qquad \text{for all } i \neq j, \text{ s.t. } (i,j) \notin \mathcal{E}. \tag{31}$$

Furthermore, the diagonal blocks of $H_{ii}(\lambda)$ are positive semidefinite.

Proof. From (26), $\sum_k \mathbf{1}(k \in \mathcal{K}_{ij}) s_{ij} = 1$, combined with (29), we have $[H_{ii}]_{kk} \geq \sum_l [H_{ii}]_{kl}$. Recall, $F'_{ij,k}(r_{ij}^k) \leq 0$ from (15) and the assumption that $f_{ij}^k(\cdot)$ is monotone decreasing. Therefore the diagonal elements are positive and according to [22, Section 6.2] H_{ii} is positive semidefinite by diagonal dominance. Block sparsity arises trivially from the fact that $\partial/\partial \lambda_{i'}^{k'} \left(- \left[\lambda_i^k - \lambda_j^k - \mu_{ij}\right]^+ \right) = 0$ for any k' when i' is not i or j. ■

Lemma 1 guarantees that all elements of the Hessian can be computed using local information. Elements of the Hessian require knowledge of the local action sets $\mathcal{K}_{ij}(\lambda)$ from (25), which are computed using the local flow values $\{r_{ij}^k(\lambda)\}_k$. Furthermore, Lemma 1 states that the diagonal blocks $H_{ii}(\lambda)$ are positive semidefinite, which in turn indicates that the nodes depend positively on their own queues. This fits our intuition because we expect penalties on a specific queue to become larger when those queues grow.

In order to accelerate Backpressure and retain its distributed nature we compute a dual update direction based on the ADD family of algorithms defined in [23]. ADD relies on splitting the dual Hessian and leverages its block sparsity pattern to approximate its inverse. To be explicit, consider the dual iterate $\lambda(t)$ at time t and denote as $H(t) = H(\lambda(t))$ the corresponding Hessian. Let I denote an identity matrix of proper dimension and define the block diagonal matrix $D(t)$ with diagonal blocks $D_{ii}(t)$ defined in terms of the Hessian diagonal blocks $H_{ii}(t)$ as

$$D_{ii}(t) = 2H_{ii}(t) + I. \tag{32}$$

The Hessian splitting separates the block diagonal matrix $D(t)$ for which we define a matrix $B(t) := D(t) - H(t)$ so that we can write

$$\begin{aligned} H(t) &= D(t) - B(t) \\ &= D(t)\left[I - D^{-1}(t)B(t)\right]. \end{aligned} \tag{33}$$

For future reference observe that $B(t)$ has the same sparsity pattern of $H(t)$. Denote the diagonal blocs of $B(t)$ as $B_{ii}(t) = D_{ii}(t) - H_{ii}(t) = I + H_{ii}(t)$ and the off diagonal blocks of $B(t)$ as $B_{ij}(t) = -H_{ij}(t)$

The ADD-N algorithm approximates the Hessian inverse $H^{-1}(t)$ by truncating the Taylor's expansion of the term $\left[I - D^{-\frac{1}{2}}(t)B(t)D^{-\frac{1}{2}}(t)\right]^{-1}$ at its Nth term. Using $\bar{H}^{(N)}$ to denote such approximation we have

$$\bar{H}^{(N)}(t) = \sum_{m=0}^{N} \left[D^{-1}(t)B(t)\right]^m D^{-1}(t). \tag{34}$$

Relying on the Hessian inverse approximation in (34) to substitute $H^{-1}(t)$ in the stochastic Newton method in (30) yields the ABP-N algorithm. Specifically, define the ABP-N direction as

$$\begin{aligned} d^{(N)}(t) \;&:=\; -\bar{H}^{(N)}(t) g(t) \\ &=\; -\sum_{m=0}^{N} \left[D^{-1}(t)B(t)\right]^m D^{-1}(t) g(t), \end{aligned} \tag{35}$$

and the ABP-N algorithm by recursive application of the iteration

$$\lambda(t+1) = \left[\lambda(t) - \alpha_t \bar{H}^{(N)}(t) g(t)\right]^+ = \left[\lambda(t) + \alpha_t d^{(N)}(t)\right]^+. \tag{36}$$

Equivalently, we can define the local components $d_i^{(N)}(t)$ of the ABP direction $d_i^{(N)}(t) = \{d_i^{(N)}(t)\}_{i=1}^n$ so that it is possible to rewrite (36) for each of the multiplier components,

$$\lambda_i(t+1) = \left[\lambda_i(t) + \alpha_t d_i^{(N)}(t)\right]^+. \tag{37}$$

The important observation here is that the ABP-N direction $d_i^{(N)}(t)$ in (37) can be computed by node i by aggregating information from, at most, N hops away. Indeed, observe that $B(t)$ has a sparsity pattern that matches the sparsity pattern of the network, because this is true of $H(t)$. Thus, when we consider the second summand $\left[D^{-1}(t)B(t)\right]^1$ in (35) we end up with a matrix whose sparsity pattern is that of the underlying network. When we consider the third term $\left[D^{-1}(t)B(t)\right]^2$ the resulting sparsity pattern is that of the network of two-hop neighbors. Since determination of the ABP direction $d^{(N)}(t)$ necessitates communication with the nodes that match this sparsity pattern, no information from nodes farther than N hops away is necessary.

More to the point, observe that the ABP-0 direction can be written as $d^{(0)} = -D^{-1} g_i$ and that for all N other than 0 we can write the recursion

$$d^{(N)}(t) = -D^{-1}(t) g(t) + D^{-1}(t) B(t) d^{(N-1)}(t). \tag{38}$$

Exploiting the fact that the sparsity pattern of $B(t)$ is that of the network and using the definitions of the local pieces $d_i^{(N)}$ of the ABP direction and $g_i(t)$ of the gradient as well as the definition

Algorithm 4: Local computation of ABP direction

1 **function** d_i^N = ABP-N dir $\left(\{r_{ij}^k, r_{ji}^k\}_{jk}, \{a_i^k\}_k, \{\mu_{ij}\}_j\right)$
2 Active sets $\mathcal{K}_{ij}$ as per (25). Sharing coefficients s_{ij} as per (26)
3 Hessian blocks H_{ij} as per (27) and (28). Blocks H_{ii} as per (29)
4 Set $D_{ii} = 2H_{ii} + I$, $B_{ii} = I + H_{ii}$ and $B_{ij} = -H_{ij}$
5 Stochastic gradients: $g_i^k = \sum_{j \in n_i} r_{ji}^k - r_{ij}^k - a_i^k - \xi$
6 Initialize ABP direction: $d_i^{(0)} = -D_{ii}^{-1} g_i$,
7 **for** *m=1,..., N* **do**
8 Send $d_i^{(m-1)}$ to $j \in n_i$. Receive directions $d_j^{(m-1)}$ from $j \in n_i$
9 Update ABP direction:
$d_i^{(m)} = -D_{ii}^{-1} g_i(t) + \sum_{j \in n_i} D_{ii}^{-1} B_{ij} d_j^{(m-1)}$
10 **end**
11 **return** $d_i^{(N)} = d_i^{(m)}$

Algorithm 5: Accelerated Backpressure for node i

1 Observe $q_i^k(0)$. Initialize $\lambda_i^k(0) = q_i^k(0)$ for all k and $i \neq o_k$
2 **for** $t = 0, 1, 2, \cdots$ **do**
3 **for** *all neighbors* $j \in n(i)$ **do**
4 Send duals $\{\lambda_i^k(t)\}_k$ – Receive duals $\{\lambda_j^k(t)\}_k$
5 Compute μ_{ij} such that
$$\sum_k F_{ij}^k \left(- \left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+ \right) = C_{ij}$$
6 Transmit packets at rate
$$r_{ij}^k(t) = F_{ij}^k \left(-[\lambda_i^k(t) - \lambda_j^k(t) - \mu_{ij}]^+ \right)$$
7 **end**
8 Send variables $\{r_{ij}^k(t)\}_{kj}$ – Receive variables $\{r_{ji}^k(t)\}_{kj}$
9 $d_i^{(N)}(t)$ = ABP-N dir $\left(\{r_{ij}^k(t), r_{ji}^k(t)\}_{jk}, \{a_i^k(t)\}_k, \{\mu_{ij}\}_j\right)$
10 Update dual variables: $\lambda_i(t+1) = \left[\lambda_i(t) + \alpha_t d_i^{(N)}(t)\right]^+$
11 **end**

of the blocks D_{ii} and B_{ij} the recursion in (38) is equivalent to the componentwise recursions

$$d_i^{(N)}(t) = D_{ii}^{-1}(t) g_i(t) + \sum_{j \in n_i, j=i} D_{ii}^{-1}(t) B_{ij}(t) d_j^{(N-1)}(t), \quad (39)$$

where the ABP-0 direction can also be computed locally as $d_i^{(0)}(t) = -D_{ii}^{-1}(t) g_i(t)$.

The expression in (39) results in a local function for the computation of the local ABP direction $d_i(t)$ that we shown in Algorithm 4. The inputs to this function are routing rates $r_{ij}^k = r_{ij}^k(t)$ and $r_{ji}^k = r_{ji}^k(t)$ for all neighbors $j \in n_i$ and flows k; arrival rates $a_i^k = a_i^k(t)$ for all flows k; and water levels μ_{ij} for all neighbors $j \in n_i$. All of these variables are available locally except for the incoming rates r_{ji}^k that are communicated from neighbors. These inputs are used to determine the active sets $\mathcal{K}_{ij}$ in (25) and the capacity sharing coefficients s_{ij} in (26) as stated in Step 2. Observe that in (25) and (26) these sets and coefficients are defined as functions of λ, which they are, but that is suffices to know r_{ij}^k, r_{ji}^k, and μ_{ij} to compute them. These preliminary computations are then used in Step 3 to determine the local Hessian blocks H_{ij} using the expressions (27) and (28) of Proposition 2 and the Hessian blocks H_{ii} using (29) in Proposition 3. The values of the Hessian block are then used to determine the corresponding blocks of the splitting matrices in (33). These blocks are $D_{ii} = 2H_{ii} + I$, $B_{ii} = I + H_{ii}$ and $B_{ij} = -H_{ij}$, which are computed in Step 4. The local components of the stochastic gradient g_i^k are then computed as dictated by (11). These preliminary computations lead to the core of the algorithm in steps 6-10 that implement the recursion in (39). Step 6 initializes the recursion by determining the value of the ABP-0 direction $d_i^{(0)} = -D_{ii}^{-1} g_i$. The update in Step 9 utilizes local and neighboring components of the ABP-$(m-1)$ direction to compute the local component of the ABP-m direction. To have the neighboring components available for this computation the local components of the ABP-$(m-1)$ direction have to be shared between neighbors as stated in Step 8. The outcome of N iterations of this loop is the ABP-N which is returned in Step 11. Observe that to implement this loop we require a total of N communication exchanges with neighboring nodes.

ABP is summarized in Algorithm 5. The algorithm is identical to Soft Backpressure up until Step 8 and is repeated for clarity. The difference is that instead of updating the dual variables by descending on the local stochastic gradient we descend along the local ABP-N direction $d_i^{(N)}(t)$ (Step 10). The ABP-N direction is computed with a call to Algorithm 4 to which the routing rates $r_{ij}^k(t)$ and $r_{ji}^k(t)$, the arrival rates $a_i^k = a_i^k(t)$, and the water levels μ_{ij} are passed as parameters (Step 8).

TABLE I
MAXIMUM COMMUNICATION COST PER NODE AND ITERATION, RELATIVE TO THE MAXIMUM DEGREE Δ

	Communication	Computation
Backpressure	$O(\Delta)$	$O(\Delta)$
Soft Backpressure	$O(2\Delta)$	$O(3\Delta)$
Accelerated Backpressure	$O((N+2)\Delta)$	$O(N\Delta^2 + 3\Delta)$

Implementation of each ABP-N iteration requires $N+2$ communications with each neighbor. These include the exchange of duals and primals in steps 4 and 8 of Algorithm 5 and the N exchanges of ABP directions in Step 9 of Algorithm 4. This is an extra N communications per neighbor relative to the cost of Soft Backpressure – which requires only exchanges of primals and duals – and an extra $N+1$ communications relative to Backpressure – which requires exchange of primals only. Denoting the maximum node degree as Δ this gives communication costs of order $O((N+2)\Delta)$, $O(2\Delta)$, and $O(\Delta)$ as summarized in Table I. The computational cost of each Backpressure iteration grows proportional to number of neighbors as we need to perform primal computations for each link. For Soft Backpressure we have primal and dual computations in steps 6 and 9 of Algorithm 3 each of which grows with the number of neighbors. We also need to compute the water level μ_{ij} as per Step 5 of Algorithm 3, the complexity of which also scales linearly with the number of neighbors. This yields a total complexity that scales in the order of 3Δ. For ABP-N we also have to implement N matrix products at a cost of not more than Δ^2 operations each. The resulting computational costs are also summarized in Table I. We emphasize that all these costs grows with the maximum degree Δ rather than with the size of the network n. The communications and computation costs of ABP grow with the parameter N. Larger N generates a more accurate approximation of the Newton Step

at the cost of additional communication and computation. In practice, implementations with $N = 1$ perform best in practice; see Section V.

IV. Stability Analysis

In order to claim the ABP algorithm is an alternative to the Backpressure and Soft Backpressure algorithms, a guarantee that it achieves queue stability is provided. We do so by combining a Lyapunov drift analysis [2] with duality theory, in particular as applied to stochastic subgradient descent, [10]. For compactness we make use of matrix vector notation and annotate time as subindices instead of arguments. The queue update equation in (1) becomes

$$q_{t+1} = q_t - g(\lambda_t) \tag{40}$$

where $q_t = \{q_i^k(t)\}$, and $g(\lambda_t) = \{g_i^k(\lambda_t)\}$ both of which are organized as $(n-1)$ stacks of $|\mathcal{K}|$-dimensional vectors. For the dual update in (36) we also remove the projection operator and write

$$\lambda_{t+1} = \lambda_t + \alpha_t d_t \tag{41}$$

where $\lambda_t = \lambda_(t)$ and $d_t = d^{(N)}(t)$. We can remove the projection in (36) because in problem (5) the capacity constraint is binding at the optimal point, making the problem equivalent to the equality constrained case. We proceed with our analysis using (41) and start by proving important properties of stochastic gradients and Hessians.

Lemma 2. *The stochastic gradient is bounded*

$$\|g_t(\lambda)\| \leq \gamma, \qquad \forall t, \forall \lambda, \tag{42}$$

every Generalized Hessian satisfies

$$\|H(\lambda)\| \leq \Gamma, \qquad \forall \lambda \tag{43}$$

and the approximate inverse Hessian is bounded

$$0 < \delta \leq \|\bar{H}(\lambda)\| \leq N+1, \qquad \forall \lambda \tag{44}$$

where N is the communication parameter introduced in (34)*.*

Proof. For $g_t(\lambda)$, consider (11) and recall our assumption that the random portion of the arrivals has finite support. Then observing from (8) that $R(\lambda)$ lies in a compact polyhedral set

$$R(\lambda) \in \mathcal{C} = \{r \geq 0 : \sum_k r_{ij}^k \leq C_{ij}, \forall (i,j) \in \mathcal{E}\}, \tag{45}$$

implies that $g_t(\lambda)$ has a finite upper bound γ that holds for all λ and all t.

In the case of $H(\lambda)$, we follow the same logic as for $g_t(\lambda)$. Consider any Generalized Hessian definition satisfying (22) and the element-wise definition in (24). $H(\lambda)$ is a function of $R(\lambda)$ because $\nabla h(\lambda) = \bar{g}(\lambda)$ defined in (9). From (25) and (26), $\mathcal{K}_{ij}(\lambda)$ and $s_{ij}(\lambda)$ are computed from $R(\lambda)$, so we can express

$$\|H(\lambda)\| = \|\tilde{H}(R(\lambda))\| \tag{46}$$

using Propositions 2 and 3. From the derivation of these propositions in Appendix A, $\tilde{H}(r)$ is finite, for finite r for any Generalized Hessian. Since $R(\lambda)$ lies in the compact polyhedral set from (45), we have

$$\|H(\lambda)\| \leq \max_{r \in \mathcal{C}} \|\tilde{H}(r)\| = \Gamma. \tag{47}$$

In the case of $\bar{H}_t(\lambda)$, we consider its construction for an arbitrary Generalized Hessian $H = H(\lambda)$. Observing equation (34) and considering the splitting $H = D - B$, each term in the sum (34) is positive definite with largest eigenvalue upper bounded by $\max_i \text{eig}\big((D_{ii})^{-1}\big)$. Since

$$D_{ii} = H_{ii} + I \succeq I$$

where $D_{ii} \succeq 0$ for any λ, we have $\|\bar{H}^{(N)}(\lambda)\| \leq N+1$.

Since each term in the sum from (34) is positive, we consider the first term for which a lower bound δ is given by $\min_i \text{eig}(D_{ii}^{-1}) = 1/(1 + \max_i \text{eig}(H_{ii}))$ where $\max_i \text{eig}(H_{ii})$ is precisely the maximum achievable eigenvalue of a diagonal block of $H(\lambda)$. Since each $H(\lambda)$ can be expressed as $\tilde{H}(R(\lambda))$ the maximum achievable value is finite for $R(\lambda) \in \mathcal{C}$, guaranteeing that our lower bound δ is strictly positive. ■

Lemma 3. *The dual function $h(\cdot)$ satisfies,*

$$h(\hat{\lambda}) - h(\lambda) \leq \bar{g}(\lambda)'(\lambda - \hat{\lambda}) + \frac{\Gamma}{2}\|\lambda - \hat{\lambda}\|^2 \tag{48}$$

where Γ is the upper bound for all generalized Hessians defined in Lemma 2.

Proof. We begin with the statement of the generalized second order Taylor expansion taken directly from item (iv) of section 2 in [24]. Given a once continuously differentiable function $h(\lambda)$ with gradient $\bar{g}(\lambda)$ and set of generalized Hessians $H(\lambda)$,

$$h(\lambda) - h(\hat{\lambda}) - \bar{g}(\lambda)'(\hat{\lambda} - \lambda) \in \mathcal{S}(\lambda, \hat{\lambda}) \tag{49}$$

where the set $\mathcal{S}(\lambda, \hat{\lambda})$ is defined

$$\mathcal{S}(\lambda, \hat{\lambda}) = \left\{ \frac{1}{2}(\lambda - \hat{\lambda})' H (\lambda - \hat{\lambda}) : H \in \cup_{z \in [\lambda, \hat{\lambda}]} H(z) \right\}. \tag{50}$$

Consider the maximal element in $\bar{s}(\lambda, \hat{\lambda}) = \max \mathcal{S}(\lambda, \hat{\lambda})$,

$$\bar{s}(\lambda, \hat{\lambda}) = \max_{H \in \cup_{z \in [\lambda, \hat{\lambda}]} H(z)} \frac{1}{2}(\lambda - \hat{\lambda})' H (\lambda - \hat{\lambda}). \tag{51}$$

From the uniform upper bound on all generalized Hessians, $||H|| \leq \Gamma$, we have

$$\bar{s}(\lambda, \hat{\lambda}) \leq \frac{\Gamma}{2}(\lambda - \hat{\lambda})'(\lambda - \hat{\lambda}). \tag{52}$$

Since $h(\hat{\lambda}) - h(\lambda) - \bar{g}(\lambda)'(\hat{\lambda} - \lambda)$ belongs to the set $S(\lambda, \hat{\lambda})$, it can be no larger than the maximal element in that set,

$$h(\hat{\lambda}) - h(\lambda) - \bar{g}(\lambda)'(\hat{\lambda} - \lambda) \leq \bar{s}(\lambda, \hat{\lambda}). \tag{53}$$

Applying (52) to (53), completes the proof. ■

Proposition 4. *Consider the ABP algorithm implemented with the dual step* (41)*, decaying step size α_t satisfying $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ and the ADD-N update direction $d_t^{(N)}$ as defined in* (35)*, then*

$$\lim_{t \to \infty} \|\bar{g}(\lambda_t)\| = 0 \tag{54}$$

almost surely.

Proof. See Appendix B. ■

Proposition 4 is the first step in our theoretic convergence guarantee. By implementing a decaying step size when updating the dual variables, convergence to the optimal dual variables is achieved. Since the sequence of dual variables converges to the optimal dual variables almost surely, our routing $R_{ij}^k(\lambda_t)$ becomes

a feasible routing. We proceed to leverage this fact to ensure the queues remain stable.

Proposition 5. *Consider a dual variables process λ_t such that the dual gradient $\bar{g}(\lambda_t)$ satisfies*

$$\lim_{t\to\infty} ||\bar{g}(\lambda_t)|| = 0, \quad a.s. \tag{55}$$

Then, all queues empty infinitely often with probability one, i.e.,

$$\liminf_{t\to\infty} q_i^k(t) = 0, \quad a.s., \qquad \text{for all } k, i \neq o_k \tag{56}$$

Proof. The result in (56) is true if for arbitrary time $\tau \geq 0$ and arbitrary constant ϵ the queue length $q_i^k(t)$ is almost surely smaller than ϵ at some point the future. To write this statement formally define the stopped process

$$\tilde{q}_i^k(t) = \begin{cases} q_i^k(t) & \text{if } q_i^k(u) > 0 \text{ for all } \tau \leq u \leq t, \\ 0 & \text{else .} \end{cases} \tag{57}$$

The stopped process $\tilde{q}_i^k(t)$ follows the queue length process $q_i^k(t)$ until the queue length becomes null for the first time after time τ. If and when the queue empties after time τ the process is stopped and all subsequent values are set to $\tilde{q}_i^k(t) = 0$. With this definition (56) is equivalent to

$$\lim_{t\to\infty} \tilde{q}_i^k(t) = 0, \quad \text{a.s.}, \qquad \text{for all } k, i \neq o_k \tag{58}$$

I.e., the definition of the stopped process allows replacement of the limit infimum in (55) by a regular limit in (58).Without loss of generality we consider $\tau = 0$ in (57) to simplify notation.

Consider the queue update defined in equation (1) rewritten in terms of the routing function in equation (8),

$$q_i^k(t+1) = q_i^k(t) - \sum_{j\in n_i} \left(R_{ij}^k(\lambda_t) - R_{ji}^k(\lambda_t) - a_i^k(t)\right). \tag{59}$$

Define a σ-algebra encoding the histories for the queues and dual variables up to time t,

$$\sigma_t = \{\lambda_\tau, q_\tau \,|\, \forall \tau \leq t\}. \tag{60}$$

Taking expectation with respect to the arrival process and conditioned on the past history as encoded by σ_t,

$$\mathbb{E}\left[q_i^k(t+1) \,\middle|\, \sigma_t\right] = q_i^k(t) - \sum_{j\in n_i} \left(R_{ij}^k(\lambda_t) - R_{ji}^k(\lambda_t) - a_i^k\right) \tag{61}$$

where we recall $\mathbb{E}\left[a_i^k(t)\right] = a_i^k$. Observe from definition (9)

$$\bar{g}_i^k(\lambda_t) + \xi = \sum_{j\in n_i} \left(R_{ij}^k(\lambda_t) - R_{ji}^k(\lambda_t) - a_i^k\right). \tag{62}$$

Substituting into (63) we have

$$\mathbb{E}\left[q_i^k(t+1) \,\middle|\, \sigma_t\right] \leq q_i^k(t) - \bar{g}_i^k(\lambda_t) - \xi. \tag{63}$$

From proposition 4 we know that $\|\bar{g}(\lambda_t)\|$ converges to zero almost surely, therefore there exists a T such that $\|\bar{g}(\lambda_t)\| \leq \xi - \epsilon$ for all $t \geq T$ for any $\epsilon \in (0, \xi)$. For any particular (i, k), we have $|\bar{g}_i^k(\lambda_t)| \leq \|\bar{g}(\lambda_t)\|$, allowing us to conclude that

$$\mathbb{E}\left[q_i^k(t+1) \,\middle|\, \sigma_t\right] \leq q_i^k(t) - \epsilon, \qquad \forall t \geq T \tag{64}$$

for all $q_i^k(t) \geq \epsilon$ because $q_i^k(t) \geq 0$ for all t.

Recalling the definition of the stopped process in (57) it follows from (64) that there exists a time T such that

$$\mathbb{E}\left[\tilde{q}_i^k(t+1) \,\middle|\, \sigma_t\right] < \tilde{q}_i^k(t) \tag{65}$$

for all $\tilde{q}_i^k(t) > 0$, therefore $\tilde{q}_i^k(t)$ converges to zero almost surely. While $q_i^k(t)$ may move away from zero after $\tilde{q}_i^k(t)$ converges, we reinitialize the stopped martingale process to show that $q_i^k(t)$ must eventually return to zero. ■

Corollary 1. *Consider the Accelerated Backpressure algorithm, defined by* (41) *with* $d_t^{(N)}$ *as defined in* (35) *and primal Lagrangian maximizers defined in* (10)*. If the step size sequence is chosen to satisfy* $\sum_t \alpha_t = \infty$ *and* $\sum_t \alpha_t^2 < \infty$*, all queues become empty infinitely often with probability one,*

$$\liminf_{t\to\infty} q_i^k(t) = 0, \quad a.s., \qquad \text{for all } k, i \neq o_k. \tag{66}$$

Proof. From Proposition 4, we know that $||\bar{g}(\lambda_t)||$ converges to zero which allows application of Proposition 5. From Proposition 5 it is guaranteed that each queue has a limit infimum equal to zero, which is equivalent to having each queue become empty infinitely often. ■

Corollary 1 is a sufficient condition for queue stability. In fact it is a much stronger result because the queues are always eventually cleared. This occurs because our dual variables converge to the optimal priorities which yield a routing $R_{ij}^k(\lambda_k)$ which forces the queue lengths to decrease in Expectation whenever they are non-zero.

V. Numerical Analysis

Numerical experiments are performed to compare the ABP algorithm summarized in algorithms 4 and 5 with the Backpressure and Soft Backpressure algorithms summarized in algorithms 2 and 3. In all algorithms we use a fixed step size $\alpha_t = 1$. Observe that constant stepsizes are not covered by the guarantees of Section IV. We use constant stepsizes here to understand this different scenario. The objective function used for Soft Backpressure and ABP is of the form

$$f_{ij}^k(r_{ij}^k) = -\frac{1}{2}\left(r_{ij}^k\right)^2 + \beta_{ij}^k r_{ij}^k. \tag{67}$$

The quadratic terms increase the cost of routing a large number of packets across a single link and help to eliminate myopic routing choices that lead to sending packets in cycles. The linear terma β_{ij}^k are introduced to reward the transmission of packets to their final destinations. In our experiments we set $\beta_{ij}^k = 10$ for all edges routing to their respective data type destinations, i.e., when $j = o_k$, and set $\beta_{ij}^k = 0$ for all other i, j, k. With this choice for f_{ij}^k the derivative inverse function in (15) is simply $F_{ij}^k(y) = \beta_{ij}^k - y$. This simple derivative inverse function simplifies implementation of steps 5 and 6 of algorithms 5 and 3. In all algorithms we make $\xi = 0$ and for ABP we further set the approximation parameter to $N = 1$ so that ABP-1 is implemented. Of all the ABP algorithms, ABP-1 is the one with smallest communication overhead. Higher order members of the ABP family perform better than ABP-1.

We begin by considering the 10 node network shown in Figure 3 with individual link capacities C_{ij} chosen uniformly at random from the interval $[10, 100]$. We consider the problem of routing 5 data types having destinations that are chosen at random and average arrival rates of 5 packets per unit of time for each data type and node. This results in an average total load of 45 packets per flow and 225 packets per unit of time for the network as a whole. Recall that these arrival rates are unknown to nodes which respond to the number of packets that are received in each time slot. The number of packets that arrive in each time slot is selected

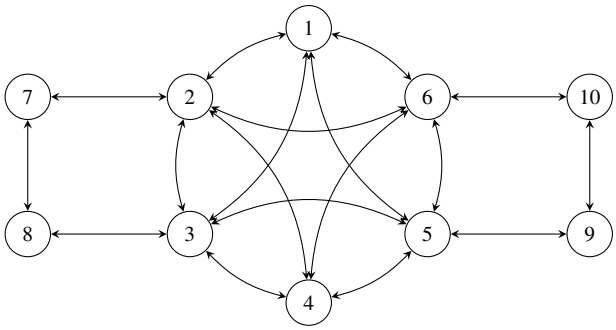

Fig. 3. Several numerical experiments for the ABP algorithm presented in this section are performed on this 10 node network with 5 data types. The destinations are unique for each data type and are chosen randomly.

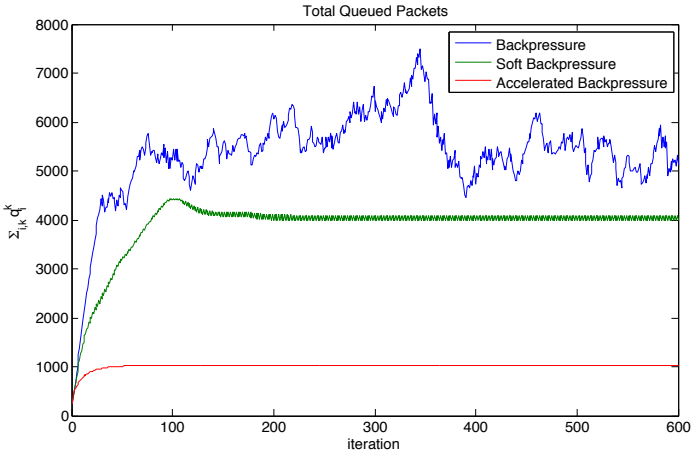

Fig. 4. Without access to arrival statistic, the ABP algorithm learns a routing strategy that stabilizes the queues faster than the Soft Backpressure or Backpressure algorithms, leading to fewer queued packets at steady state.

from the discrete uniform distribution over the set $\{0, 1, \ldots, 10\}$.

Figure 4 shows the total number of packets queued in the system as a function of elapsed time for a sample run. ABP-1 stabilizes the queues after about 50 iterations while Soft Backpressure requires in excess of 100 iterations. The number of iterations that Backpressure needs to stabilize queues is difficult to judge due to its volatility but it seems that it takes in the order of 300 to 400 iterations for the queues to stop growing. Easier to judge is the fact that ABP-1 stabilizes queues at a much shorter length. The total number of packets queued never exceeds 1,110. Soft Backpressure stabilizes with around 4,000 packets in the system, while Backpressure grows to more than 7,000 queued packets before reducing the number of packets to about 5,000. Further observe that in addition to having shorter queues, ABP-1 has smaller variations in queue lengths.

The reason why ABP-1 is able to achieve less volatility in queue lengths is because the dual variables approach their optimal values, which in turn leads to less variation in the routing variables computed from them. This is not true of Backpressure and Soft Backpressure whose dual variables have large oscillations around their optimal values. This is illustrated in Figure 5. The dominant feature of dual variables for Backpressure and Soft Backpressure is their oscillatory behavior. For ABP-1 the dual variables converge to their optimal values in about 50 iterations, which explains why queues stabilize after this much time elapses.

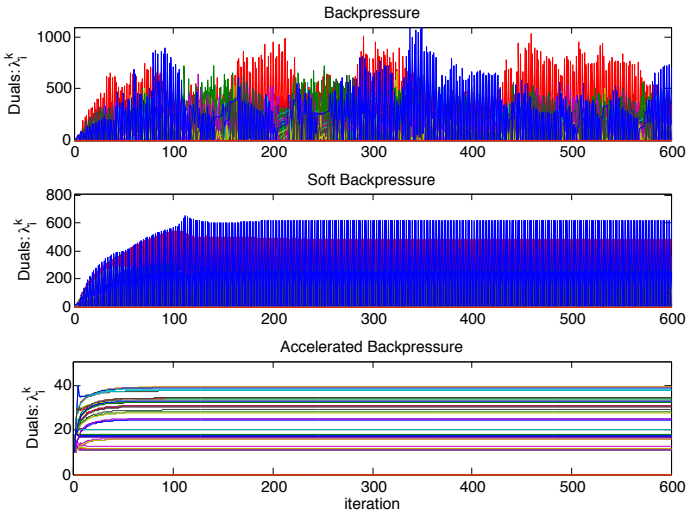

Fig. 5. The Accelerated Backpressure algorithm converges to the optimal dual variables yielding a consistent routing strategy rather than the oscillatory solutions generated by Backpressure and Soft Backpressure.

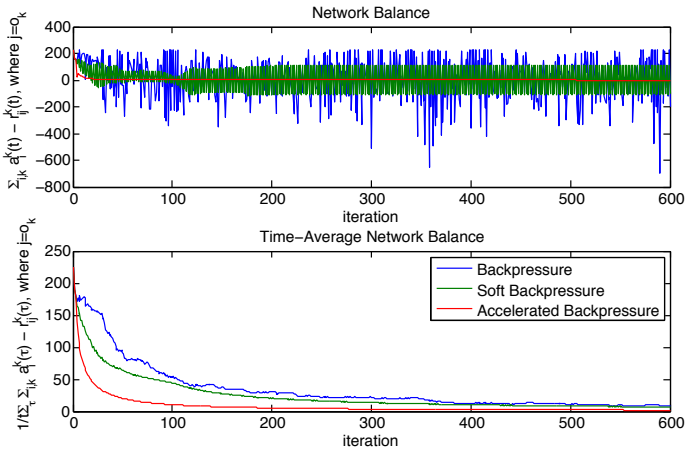

Fig. 6. A successful routing strategy is characterized by routing as many packets out of the network as are arriving exogenously. While Soft Backpressure and soft Backpressure can achieve this state on average, Accelerated Backpressure achieves this balance on a per iteration basis.

The resulting stability in routing variables is illustrated in Figure 6-top. Shown in this figure is the instantaneous value of the queue imbalances $\sum_{j\in n_i} r_{ij}^k(t) - r_{ji}^k(t) - a_i^k(t)$ summed over the whole network. Observe how ABP-1 succeeds in satisfying these constraints for all times – this is not strictly true, there are still random variations around $\sum_{j\in n_i} r_{ij}^k(t) - r_{ji}^k(t) - a_i^k(t) = 0$ but they are small. For Backpressure and Soft Backpressure this is not true instantaneously. It is true on average, as we show in Figure 6-bottom, and this is sufficient to stabilize queues. However, the instantaneous variations in this constraints make the behavior of Backpressure and Soft Backpressure more erratic than the behavior of ABP-1. The cost of achieving this more stable behavior is the added communication overhead as summarized in Table I.

The results in figures 4-6 are illustrative of a sample run. A more comprehensive analysis involves studying statistics across a number of realizations. We do so for the average queue balance constraints in Figure 7, which we can interpret as a statistical version of Figure 6-bottom. The averages in Figure 7 are across

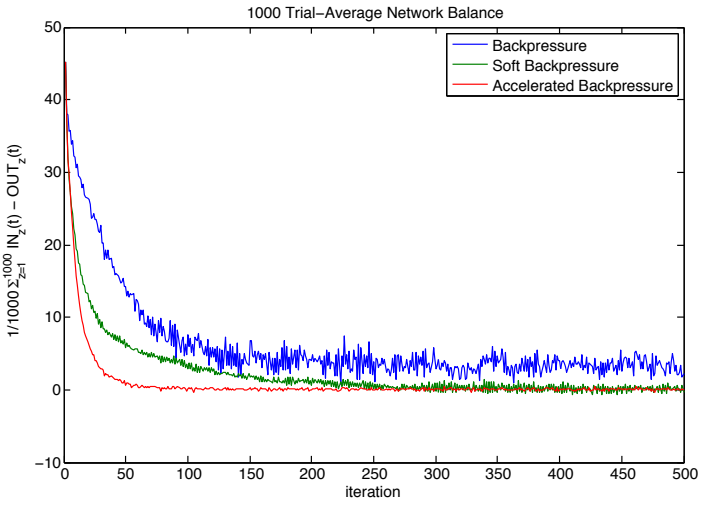

Fig. 7. The rate of convergence for each algorithm is effectively how quickly it can achieve a network balance. Performance can be difficult to judge in a single realization so we compare the average across 1000 trials on barely feasible networks.

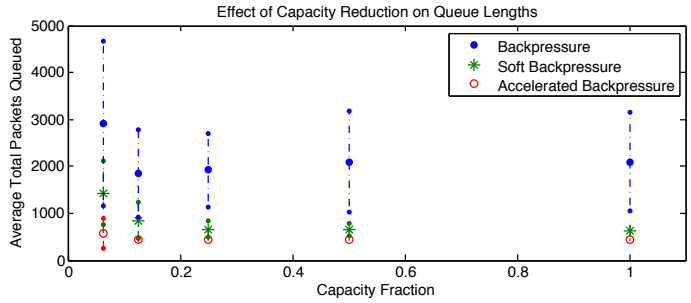

Fig. 8. The total packets queued are averaged over 100 realizations of packet arrive sequences for each data point. The capacities of the edges are reduced by 50% and repeat the experiment until the problem becomes infeasible. Error bars show ± 1 standard deviation. Reduction in network capacity doesn't significantly effect performance until the problem is nearly infeasible.

1,000 trials. ABP-1 achieves queue balance after fewer than 50 iterations, while Soft Backpressure requires over 250 and Backpressure fails to achieve balance even after 500 iterations have passed. While there are small quantitative differences between figures 6-bottom and 7, their qualitative properties are the same.

Since network balance is equivalent to achieving a feasible solution to the primal optimization problem, it is interesting to consider what happens when the edge capacities are reduced, shrinking the set of feasible routing strategies. To evaluate the effect of edge capacities, we consider the queue stabilization problem with capacities $C = \rho C_0$ where $[C_0]_{ij}$ are the capacities for edge (i, j) and the capacity fraction ρ is halved until the problem becomes infeasible, i.e., $\rho = 1, 1/2, 1/4$, etc. Since the arrival rates are stochastic, the experiment is repeated 100 times for each ρ and the mean and standard deviation are shown in Figure 8. The results show that ABP has robust performance as the boundary of feasibility is approached. Error bars which show ± 1 standard deviation, indicate that the ABP and Soft Backpressure solutions are not significantly affected by the realization of packet arrivals until the problem is nearly infeasible. Backpressure solutions had standard deviation in excess of 50% of the mean, implying a significant dependence on the specific packet arrival realization.

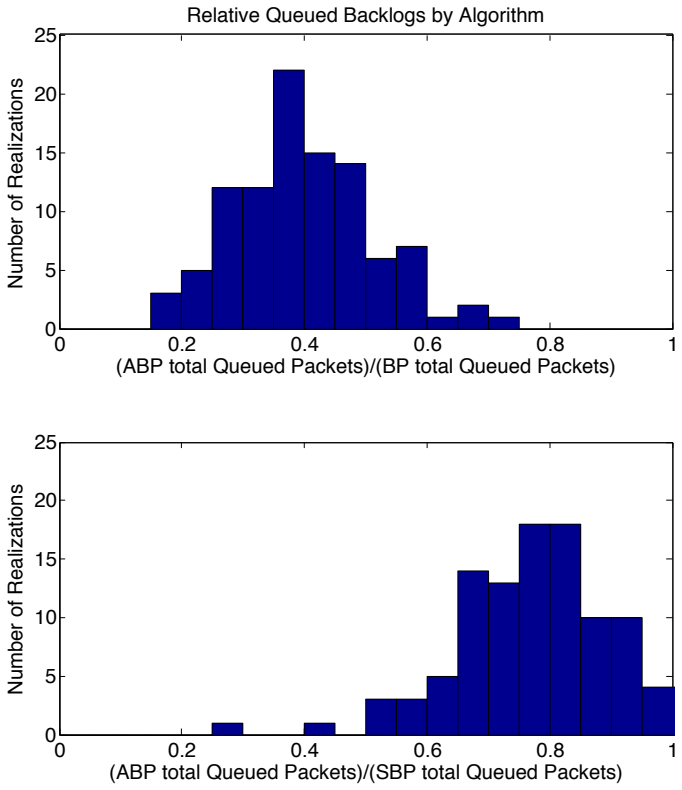

Fig. 9. (top) Over 100 trials on random proximity networks, the total number of packets queued at steady state under the ABP algorithm is on average about 40% the number left in queue by the Backpressure algorithm. (bottom) The Soft Backpressure algorithm occasionally performs as well as Accelerated Backpressure but on average Accelerated Backpressure reduces the steady state queued packets by about 25%.

The analysis that we made for the specific network topology in Figure 3 still hold for more general networks. To demonstrate this statement we consider randomly generated networks. The nodes are placed uniformly at random on $[0, 1] \times [0, 1]$ and are connected if they fall within a connectivity radius of 0.4 of each other. The arrival statistics, edge capacities and objective function remain as before. From Figure 9, it is observed that the ABP algorithm outperforms Soft Backpressure and Backpressure in trials on 100 randomly generated proximity networks with 20 nodes. ABP has fewer steady state queued packets than Soft Backpressure and Backpressure in every trial. The total number of packets queued at steady state using ABP is on average about 40% less than the number of packets queued by Backpressure. The Soft Backpressure algorithm occasionally performs as well as ABP, but on average ABP reduces the total queued packets by 25%.

VI. CONCLUSION

The main contribution of this work is the introduction of a locally computable approximation to the Newton method for solving the queue stabilization problem. This method for packet routing in networks is novel its use queue priorities in place of the queue lengths themselves. This approach retains the distributed information structure necessary for implementing the algorithm efficiently at the node level. Node level protocols are presented and it is proven that the ABP Algorithm stabilizes the queues and drives them to zero infinitely often when applied with a

decaying step size. Numerical experiments demonstrate significant reduction in queue lengths and the ability to stabilize queues in significantly fewer iterations when implemented with a fixed unit step size.

APPENDIX A: PROOF OF PROPOSITIONS 2 AND 3

Proof. We begin by computing the optimal flow rates $R_{ij}^k(\lambda)$ from the optimal queue priorities as defined in (16). Substituting into

$$\frac{\partial \mathcal{L}(R(\lambda),\lambda)}{\partial \lambda_i^k} = \sum_{j\in n_i} R_{ji}^k(\lambda) - R_{ij}^k(\lambda) - a_i^k \tag{68}$$

and differentiating with respect to λ we construct the Hessian. Since a_i^k is a constant the key is differentiating $R_{ij}^k(\lambda)$ with respect to λ. We can differentiate (16) using the chain rule yielding

$$\frac{\partial R_{ij}^k(\lambda)}{\partial y} = \frac{\partial F_{ij}^k(x)}{\partial x}\frac{\partial}{\partial y}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}\right]^+\right) \tag{69}$$

where $y = \lambda_{i'}^{k'}$ for any $k', i' \neq o_{k'}$. The existence of $\partial F_{ij}^k(x)/\partial x$ is guaranteed by our assumptions on the edge costs f_{ij}^k. Differentiating $-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+$ is done by considering two cases. First, when the capacity constraint on edge (i,j) is inactive we have $\mu_{ij}(\lambda) = 0$ from (21). In this case we have

$$\frac{\partial}{\partial \lambda_i^k}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = \begin{cases} -1 & \text{for } \lambda_i^k > \lambda_j^k \\ 0 & \text{for } \lambda_i^k \leq \lambda_j^k \end{cases} \tag{70}$$

and differentiating with respect to λ_j^k, we have

$$\frac{\partial}{\partial \lambda_j^k}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = \begin{cases} 1 & \text{for } \lambda_i^k > \lambda_j^k \\ 0 & \text{for } \lambda_i^k \leq \lambda_j^k \end{cases}. \tag{71}$$

There is no cross dependence between commodities k and l because $\mu_{ij} = 0$. Recall that a commodity is active on edge (i,j) if $\lambda_i^k > \lambda_j^k$, allowing us to substitute the condition $\lambda_i^k > \lambda_j^k$ for $k \in \mathcal{K}_{ij}$ in (70) and (71).

In the second case, the capacity constraint on edge (i,j) is active and $\mu_{ij}(\lambda) > 0$. By our design in Proposition 1, $\mu_{ij}(\lambda)$ produces the reverse water filling effect, producing

$$\frac{\partial}{\partial \lambda_i^k}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = \frac{1}{|\mathcal{K}_{ij}(\lambda)|} - 1 \tag{72}$$

and differentiating with respect to λ_j^k, we have

$$\frac{\partial}{\partial \lambda_i^k}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = 1 - \frac{1}{|\mathcal{K}_{ij}(\lambda)|} \tag{73}$$

where $\mathcal{K}_{ij}$ is the set of active commodities on the edge (i,j). Unlike in the previous case, there are cross terms:

$$\frac{\partial}{\partial \lambda_i^l}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = \frac{-1}{|\mathcal{K}_{ij}(\lambda)|}, \tag{74}$$

differentiating with respect to λ_j^l, we have

$$\frac{\partial}{\partial \lambda_i^l}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = \frac{1}{|\mathcal{K}_{ij}(\lambda)|}. \tag{75}$$

Discontinuities arise precisely when we transition from case one to case two which corresponds to the capacity constraint becoming active, i.e. an edge becoming saturated. In accordance with the generalized Hessian definition in (22), we can select any value in the convex hull of the limit points of our discontinuity. We proceed by selecting values produced by case 1, because many of the terms are null, greatly simplifying the computation. In fact case 1, also contains discontinuities when a commodity becomes active on an edge. In (70), our generalized Hessian definition allows us to choose

$$\frac{\partial}{\partial \lambda_i^k}\left(-\left[\lambda_i^k - \lambda_j^k - \mu_{ij}(\lambda)\right]^+\right) = 0 \tag{76}$$

when $\lambda_i^k = \lambda_j^k$ and the same holds for (71). To complete the proof we must place all the partial derivatives in their appropriate blocks. Observe that the blocks of $H(\lambda)$ correspond to node pairs (i,j). For pairs $(i,j) \notin \mathcal{E}$ the $H_{ij} = 0$ because there are no terms $R_{ij}^k(\lambda)$ for these pairs. Consider H_{ij} for $(i,j) \in \mathcal{E}$, generated by

$$[H_{ij}]_{kl} = \frac{\partial}{\lambda_i^k} g_j^l(\lambda) = \frac{\partial}{\lambda_i^k}\left(R_{ij}^l(\lambda) - R_{ji}^l(\lambda)\right) \tag{77}$$

which also holds in the case where $l = k$. For the diagonal blocks more of our partial derivative terms appear,

$$[H_{ii}]_{kl} = \frac{\partial}{\lambda_i^k} g_i^l(\lambda) = \sum_{j\in n_i}\frac{\partial}{\lambda_i^k}\left(R_{ij}^l(\lambda) - R_{ji}^l(\lambda)\right) \tag{78}$$

which also holds in the case where $l = k$. Applying the definitions of $\mathcal{K}_{ij}(\lambda)$ from (25), $s_{ij}(\lambda)$ from (26) and the partial derivatives from (70)-(75), completes the construction of the generalized Hessian. ∎

APPENDIX B: PROOF OF PROPOSITION 4

Proof. Applying Lemma 3, for $\lambda = \lambda_t$ and $\hat{\lambda} = \lambda_{t+1}$, we have

$$h(\lambda_{t+1}) \leq h(\lambda_t) + \bar{g}(\lambda_t)'(\lambda_{t+1} - \lambda_t) + \frac{\Gamma}{2}\|\lambda_{t+1} - \lambda_t\|^2. \tag{79}$$

Define the dual optimality gap

$$L_t = h(\lambda_t) - h(\lambda^*). \tag{80}$$

Subtracting $h(\lambda^*)$ from both sides of (79), using the definition in (80), and observing that according to the dual ABP update in (41) we have $\lambda_{t+1} - \lambda_t = -\alpha_t \bar{H}(\lambda_t) g_t(\lambda_t)$ we can write

$$L_{t+1} \leq L_t - \alpha_t \bar{g}(\lambda_t)'\bar{H}(\lambda_t) g_t(\lambda_t) + \frac{\Gamma}{2}\|\alpha_t \bar{H}(\lambda_t) g_t(\lambda_t)\|^2. \tag{81}$$

Further recall the upper bound $\|\bar{H}(\lambda)\| \leq 2$ on the norm of the approximate Hessian inverse as well as the upper bound $\|g_t(\lambda)\| \leq \gamma$ on the norm of the stochastic gradient, both derived in Lemma 2. Using these bounds in the third summand on the right hand side we simplify (81) to

$$L_{t+1} \leq L_t - \alpha_t \bar{g}(\lambda_t)'\bar{H}(\lambda_t) g_t(\lambda_t) + \frac{\Gamma}{2}\alpha_t^2 (N+1)^2 \gamma^2. \tag{82}$$

Let $\lambda_{0:t}$ stand in for the history of the dual variables process up until time t and consider the expectation $\mathbb{E}\left[L_{t+1} \,\middle|\, \lambda_{0:t}\right]$ of the duality gap at time $t+1$ given this past history. With $\lambda_{0:t}$ given, λ_t in particular is given in (82). Since the duality gap L_t and the Hessian estimate $H(\lambda_t)$ depend on λ_t only, $H(\lambda_t)$ and L_t are also given in (82) and the only random variable left in the right hand side is $g_t(\lambda_t)$. Thus, it follows from (82) that the conditional expectation $\mathbb{E}\left[L_{t+1} \,\middle|\, \lambda_{0:t}\right]$ can be bounded as

$$\begin{aligned}\mathbb{E}\left[L_{t+1} \,\middle|\, \lambda_{0:t}\right] \\ \leq L_t - \alpha_t \bar{g}(\lambda_t)'\bar{H}(\lambda_t)\mathbb{E}\left[g_t(\lambda_t) \,\middle|\, \lambda_{0:t}\right] + \frac{\Gamma}{2}\alpha_t^2 (N+1)^2\gamma^2.\end{aligned} \tag{83}$$

By definition, the stochastic gradient $g_t(\lambda_t)$ is such that $\mathbb{E}\left[g_t(\lambda_t)\,\middle|\,\lambda_{0:t}\right] = \mathbb{E}\left[g_t(\lambda_t)\,\middle|\,\lambda_t\right] = \bar{g}(\lambda_t)$. Substituting this equality in (83) yields

$$\mathbb{E}\left[L_{t+1}\,\middle|\,\lambda_{0:t}\right] \leq L_t - \alpha_t \bar{g}(\lambda_t)'\bar{H}(\lambda_t)\bar{g}(\lambda_t) + \frac{\Gamma}{2}\alpha_t^2(N+1)^2\gamma^2. \tag{84}$$

Observe that the quadratic form $\bar{g}(\lambda_t)'\bar{H}(\lambda_t)\bar{g}(\lambda_t)$ is positive definite since the smallest eigenvalue of $\bar{H}(\lambda)$ is bounded by a nonnegative constant δ as shown in Lemma 2. Using this eigenvalue lower bound further reduces (84) to

$$\mathbb{E}\left[L_{t+1}\,\middle|\,\mathcal{F}_t\right] \leq L_t - \alpha_t\delta\|\bar{g}(\lambda_t)\| + \frac{\Gamma}{2}\alpha_t^2(N+1)^2\gamma^2. \tag{85}$$

The sequences L_t, $\alpha_t\delta\|\bar{g}(\lambda_t)\|$, and $\frac{\Gamma}{2}\alpha_t^2(N+1)^2\gamma^2$ are nonnegative. The sequence $\frac{\Gamma}{2}\alpha_t^2(N+1)^2\gamma^2$ is also summable because square summability is a condition on the step size sequence α_t. These properties allow application of the supermartingale convergence theorem, see e.g., [25, Theorem E.7.4]. Given nonnegative stochastic processes $V_t \geq 0$, $X_t \geq 0$ and $Y_t \geq 0$ and a sigma algebra $\mathcal{F}_t$ measuring the processes up until time t. If the processes are such that

$$\mathbb{E}\left[V_{t+1}\,\middle|\,\mathcal{F}_t\right] \leq V_t - X_t + Y_t, \tag{86}$$

and the process Y_t is summable with probability 1 then X_t is almost surely summable and that the limit of V_t exists almost surely, i.e.,

$$\lim_{t\to\infty} V_t = V_\infty, \quad \sum_{t=1}^{\infty} X_t < \infty, \quad \text{a.s.} \tag{87}$$

Identifying $\mathcal{F}_t$ with $\lambda_{0:t}$, V_t with L_t, X_t with $\alpha_t\delta\|\bar{g}(\lambda_t)\|$ and Y_t with $\Gamma/2\alpha_t^2(N+1)^2\gamma^2$ it follows from (87) that

$$\lim_{t\to\infty} L_t = L_\infty, \qquad \sum_{t=0}^{\infty} \alpha_t\delta\|\bar{g}(\lambda_t)\|^2 < \infty, \quad \text{a.s.} \tag{88}$$

We prove that L_∞ not only exists but is zero for almost all realizations. To do so, consider a realization for which $L_\infty \neq 0$. Then, the limit must be strictly positive because the sequence L_t is nonnegative. In turn, this implies that there exists a time τ such that for all $t \geq \tau$ we have $L_t \geq \kappa_1$ for some positive constant $\kappa_1 > 0$. Since $L_t = h(\lambda_t) - h(\lambda^*)$ this means that $h(\lambda_t) \geq h(\lambda^*) + \kappa_1$ for all times $t \geq \tau$. Recall now that for a differentiable convex function we can have $\nabla h(\lambda) = 0$ only when $\lambda = \lambda^*$; this is true even if $h(\lambda)$ is not differentiable, as long as we reinterpret $\nabla h(\lambda)$ as a subgradient. Using this fact it follows that if $h(\lambda_t) \geq h(\lambda^*) + \kappa_1$ for all $t \geq \tau$ there must be a constant $\kappa_2 > 0$ such that

$$\|\nabla h(\lambda_t)\| \geq \kappa_2 > 0, \quad \text{for all } t \geq \tau. \tag{89}$$

Observe now that by definition $\bar{g}(\lambda_t) = \nabla h(\lambda_t)$. Thus, we can use (89) to lower bound the series in (88) as

$$\sum_{t=0}^{\infty} \alpha_t\delta\|\bar{g}(\lambda_t)\|^2 \;\geq\; \sum_{t=\tau}^{\infty} \alpha_t\delta\|\bar{g}(\lambda_t)\|^2 \;\geq\; \kappa_2^2\delta\sum_{t=\tau}^{\infty}\alpha_t. \tag{90}$$

Since the step size sequence is nonsummable by assumption, we have that $\sum_{t=\tau}^{\infty}\alpha_t = \infty$ which, upon substitution in (90) yields

$$\sum_{t=0}^{\infty} \alpha_t\delta\|\bar{g}(\lambda_t)\|^2 = \infty \tag{91}$$

Since we know that (91) is true in, at most, a set of zero measure, and also true whenever $L_\infty \neq 0$, it must be that $L_\infty \neq 0$ in a set of zero probability. Thus, we must have $L_\infty = 0$ a.s. Further using the definition of $L_t = h(\lambda_t) - h(\lambda^*)$ we conclude

$$\lim_{t\to\infty} h(\lambda_t) = h(\lambda^*), \quad \text{a.s.} \tag{92}$$

As we already pointed out the dual function is differentiable according to Proposition 1. It then must satisfy the first order optimality condition $\bar{g}(\lambda^*) = \nabla h(\lambda^*) = 0$. Thus, the almost sure convergence of $h(\lambda_t)$ to $h(\lambda^*)$ stated in (92) implies almost sure convergence of $\bar{g}(\lambda^*) = \nabla h(\lambda^*)$ to 0 as stated in (54). ■

REFERENCES

[1] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37:1936–1948, 1992.

[2] L. Georgiadis, M. J. Neely, and L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1:1–144, 2006.

[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*, 23:89–103, 2005.

[4] F.P. Kelly, A.K. Maulloo, and D.K. Tan. Rate control for communication networks: shadow prices, proportional fairness, and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[5] S. Low and D.E. Lapsley. Optimization flow control, I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[6] A. Eryilmaz and R. Srikant. Joint congestion control, routing, and mac for stability and fairness in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(8):1514–1524, Aug 2006.

[7] M. Chiang, S.H. Low, A.R. Calderbank, and J.C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, 2007.

[8] M. J. Neely. Energy optimal control for time varying wireless networks. *IEEE Transactions on Information Theory*, 52:2915–2934, 2006.

[9] A. Ribeiro and G. Giannakis. Separation principles in wireless networking. *IEEE Transactions on Information Theory*, 58:4488–4505.

[10] A. Nedić and A. Ozdaglar. Distributed subradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control, forthcoming*, 2008.

[11] A. Ribeiro. Stochastic soft backpressure algorithms for routing and scheduling in wireless ad-hoc networks. In *IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, 2009.

[12] Bertsekas and Gafni. Projected newton methods and optimization of multicommodity flow. *IEEE Transactions on Automatic Control*, 28:1090–1096, 1983.

[13] J. G. Klincewicz. A newton method for convex separable network flow problems. *Bell Laboratories*, 1983.

[14] S. Authuraliya and S. Low. Optimization flow control with newton-like algorithm. *Telecommunications Systems*, 15:345–358, 2000.

[15] A. Jadbabaie, A. Ozdaglar, and M. Zargham. A distributed newton method for network optimization. In *Proceedings of IEEE CDC*, 2009.

[16] Ermin Wei, A. Ozdaglar, and A. Jadbabaie. A distributed newton method for network utility maximization. *Automatic Control, IEEE Transactions on*, 58(9):2162–2175, Sept 2013.

[17] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie. Accelerated dual descent for network flow optimization. *IEEE Transactions on Automatic Control*, 59(4):905–920, April 2014.

[18] M. Zargham, A. Ribeiro, , and A. Jadbabaie. Accelerated backpressure algorithm. *Proceedings of the IEEE GLOBECOM*, 2013.

[19] A. Ribeiro and G. B. Giannakis. Separation theorems of wireless networking. *IEEE Transactions on Information Theory*, 2007.

[20] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[21] J. Sun and H. Kuo. Applying a newton method to strictly convex separable network quadratic programs. *SIAM Journal of Optimization*, 8, 1998.

[22] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, New York, 1979.

[23] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie. Accelerated dual descent for network optimization. In *Proceedings of IEEE ACC*, 2011.

[24] A. Guerraggio, D. Luc, and N. Minh. Second-order optimality conditions for c1 multiobjective programming problems. *ACTA Mathematica Vietnamica*, 26(3):257–268, 2001.

[25] V. Solo and X. Kong. *Adaptive Signal Processing Algorithms: Stability and Performance*. Prentice-Hall, 1995.