Abstract—The backpressure routing and scheduling, with throughput-optimal operation guarantee, is a promising technique to improve throughput over wireless multi-hop networks. Although the backpressure framework is conceptually viewed as layered, the decisions of routing and scheduling are made jointly, which imposes several challenges in practice. In this work, we present Diff-Max, an approach that separates routing and scheduling and has three strengths: (i) Diff-Max improves throughput significantly, (ii) the separation of routing and scheduling makes practical implementation easier by minimizing cross-layer operations; i.e., routing is implemented in the network layer and scheduling is implemented in the link layer, and (iii) the separation of routing and scheduling leads to modularity; i.e., routing and scheduling are independent modules in Diff-Max and one can continue to operate even if the other does not. Our approach is grounded in a network utility maximization (NUM) formulation of the problem and its solution. Based on the structure of Diff-Max, we propose two practical schemes: Diff-subMax and wDiff-subMax. We demonstrate the benefits of our schemes through simulation in ns-2, and we implement a prototype on smartphones.

I. INTRODUCTION

The backpressure routing and scheduling paradigm has emerged from the pioneering work in [1], [2], which showed that, in wireless networks where nodes route packets and make scheduling decisions based on queue backlog differences, one can stabilize queues for any feasible traffic. This seminal idea has generated a lot of research interest. Most importantly: it has been shown that backpressure can be combined with flow control to provide utility-optimal operation guarantee [3].

The strengths of these techniques have recently increased the interest on practical implementation of backpressure framework over wireless networks, some of which are summarized in Section VI. However, the practical implementation of backpressure imposes several challenges mainly due to the joint nature of the routing and scheduling algorithms, which is the focus of this paper.

In classical backpressure, each node constructs per-flow queues. Based on the per-flow queue backlog differences, and by taking into account the state of the network, each node makes routing and scheduling decisions. Although the backpressure framework is conceptually viewed as layered, the decisions of routing and scheduling are made jointly. To better illustrate this key point, let us discuss the following example.

This work was supported by NSF grant CNS-0915988, ONR grant N00014-12-1-0064, ARO Muri grant number W911NF-08-1-0238.
layer, while the scheduling algorithms are implemented in the link layer in current networks. However, the joint routing and scheduling nature of backpressure imposes challenges for practical implementation. To deal with these challenges, [4] implements the backpressure at the link layer, [5] proposes a system in the MAC layer. This approach is practically difficult due to device memory limitations and strict limitations imposed by device firmware and drivers not to change the link layer functionalities. The second approach is to implement backpressure in (or below) the network layer, [6], [7], [8]. This approach requires joint operation of the network and link layers, so that the backpressure framework gracefully work with the link layer. Therefore, the network and link layers should work together synchronously, which may not be practical for many off-the-shelf devices.

Existing networks are designed in layers, in which protocols and algorithms are modular and operate independently at each layer of the protocol stack. E.g., routing algorithms at the network layer should work in a harmony with different types of scheduling algorithms in the link layer. However, the joint nature of the backpressure stresses joint operation and hurts modularity, which is especially important in contemporary wireless networks, which may vary from a few node networks to ones with hundreds of nodes. It is natural to expect that different types of networks, according to their size as well as software and hardware limitations, may choose to employ backpressure partially or fully. E.g., some networks may be able to employ both routing and scheduling algorithms, while others may only employ routing. Therefore, the algorithms of backpressure, i.e., routing and scheduling should be modular.

In this paper, we are interested in a framework in which the routing and scheduling are separated. We seek to find such a scheme where routing is performed independently at the network layer and scheduling decisions are performed at the link layer. The key ingredients of our approach, which we call Diff-Max1 are; (i) per-flow queues at the network layer and making routing decision based on their differences, (ii) per-link queues at the link layer and making scheduling decision based on their size.

**Example 1 - continued:** Let us consider Fig. 1 (b) for Diff-Max operation. (i) Routing: at time \( t \), node \( i \) makes routing decision for flows 1 and 2 based on queue backlogs \( \tilde{D}_{i,j}^s(t) \) and \( \tilde{D}_{i,k}^s(t) \), where \( s \in \{1, 2\} \). This decision is made at the network layer and the routed packets are inserted in the link layer queues. Note that in classical backpressure, routed packets are scheduled jointly, i.e., when a packet is routed, it should be transmitted if the corresponding links are activated. Hence, both algorithms should make decision jointly in classical backpressure. However, in our scheme, a packet may be routed at time \( t \), and scheduled and transmitted at a later time \( t+T \) where \( T > 0 \). (ii) Scheduling: at the link layer, links are activated and packets are transmitted based on per-link queue sizes; \( V_{i,j} \), \( V_{i,k} \), and \( C(t) \). The details of Diff-Max are provided in Section III.

Our approach is grounded in a network utility maximization (NUM) framework [9]. The solution decomposes into several parts with an intuitive interpretation, such as routing, scheduling, and flow control. The structure of the NUM solution provides insight into the design of our scheme, Diff-Max. Thanks to separating routing and scheduling, Diff-Max makes the practical implementation easier and minimizes cross-layer operations. We also propose two practical schemes; Diff-subMax and wDiff-subMax. The following are the key contributions of this work:

- We propose a new system model and NUM framework to separate routing and scheduling. Our solution to the NUM problem, separates routing and scheduling such that routing is implemented at the network layer, and scheduling is at the link layer. Based on the structure of the NUM solution, we propose Diff-Max.
- We extend Diff-Max to employ routing and scheduling parts, but disable the link activation part of the scheduling algorithm. We call the new framework Diff-subMax, which reduces computational complexity and overhead significantly, and provides high throughput improvements in practice. Namely, Diff-subMax only needs information from one-hop away neighbors to make its routing and scheduling decisions.
- We propose a window-based routing mechanism, wDiff-subMax, which implements routing, but disables the scheduling. wDiff-subMax is designed for the scenarios, in which the implementation of the scheduling algorithm in the link layer is impossible (or not preferable) due to device restrictions. wDiff-subMax makes routing decision on the fly, and minimizes overhead.
- We evaluate our schemes in a multi-hop setting and consider their interaction with transport, network, and link layers. We perform numerical calculations confirming that Diff-Max is as good as backpressure. We implement our schemes in a simulator; ns-2 [10], and show that they significantly improve throughput as compared to adaptive routing schemes such as Ad hoc On-Demand Distance Vector (AODV) [11]. Finally, we implemented a prototype of wDiff-subMax on Galaxy Nexus smartphones with Android 4.0 (Ice Cream Sandwich) [12].

The structure of the rest of the paper is as follows. Section II gives an overview of the system model. Section III presents the NUM formulation and solution. Section IV presents the design and development of Diff-Max schemes and their interaction with the protocol stack. Section V presents simulation results. Section VI presents related work. Section VII concludes the paper.  

**II. System Overview**

We consider multi-hop wireless networks, in which packets from a source traverse potentially multiple wireless hops before being received by their receiver. In this setup, each wireless node is able to perform routing, scheduling, and flow
control. In this section, we provide an overview of this setup and highlight some of its key characteristics. Fig. 2 shows the key parts of our system model in an example topology.

A. Notation and Setup

The wireless network consists of $N$ nodes and $L$ edges, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of edges in the network. We consider in our formulation and analysis that time is slotted, and $t$ refers to the beginning of slot $t$.

1) Sources and Flows: Let $\mathcal{S}$ be the set of unicast flows between source-destination pairs in the network. Each flow $s \in \mathcal{S}$ arrives from the application layer to the transport layer with rate $A_s(t)$, $\forall s \in \mathcal{S}$ at time slot $t$. The arrival rates are i.i.d. over the slots and their expected values are: $\lambda_s = E[A_s(t)], \forall s \in \mathcal{S}$, and $E[A_s(t)]^2$ are finite. Transport layer stores the arriving packets in reservoirs (i.e., transport layer per-flow queues), and controls the flow traffic. In particular, each source $s$ is associated with rate $x_s$, considering a utility function $q_s(x_s)$, which we assume to be a strictly concave function of $x_s$. The transport layer determines $x_s(t)$ at time slot $t$ according to the utility function $q_s$, $x_s(t)$ packets are transmitted from the transport layer reservoir to the network layer at slot $t$.

2) Queue Structures: At node $i \in \mathcal{N}$, there are network and link layer queues. The network layer queues are per-flow queues; i.e., $U^s_i$ is the queue at node $i \in \mathcal{N}$ that only stores packets from flow $s \in \mathcal{S}$. The link layer queue stores per-link queues; i.e., at each node $i \in \mathcal{N}$, a link layer queue $V_{i,j}$ is constructed for each neighbor node $j \in \mathcal{N}$ (Fig. 2).2

3) Flow Rates: Our model optimizes the flow rates among different nodes as well as the flow rates in a node among different layers; transport, network, and link layer.

The transport layer determines $x_s(t)$ at time $t$, and passes $x_s(t)$ packets to the network layer. These packets are inserted in the network layer queue; $U^s_i$ (assuming that node $i$ is the source node of flow $s$). The network layer may also receive packets from the other nodes and insert them in $U^s_i$. The link transmission rate is $h_{k,i}(t)$ at time $t$. $h_{k,i}(t)$ is larger than (or equal to) per-flow data rates over link $k-i$. E.g., we can write for Fig. 2 that $h_{k,i}(t) \geq h_{k,i}^s(t) + h_{k,i}^s(t)$ where $h_{k,i}^s(t)$ is the data rate of flow $s$ over link $k-i$. Note that $h_{k,i}^s(t)$ is the actual data transmission rate of flow $s$ over link $k-i$, while $h_{k,i}(t)$ is the available rate over link $k-i$, at time $t$. At every timeslot $t$, $U^s_i$ changes according to the following dynamics.

$$U^s_i(t + 1) = \max[U^s_i(t) - \sum_{j \in \mathcal{N}} f^s_{i,j}(t), 0] + \sum_{j \in \mathcal{N}} h^s_{j,i}(t) + x_s(t)1_{i = o(s)}$$ (1)

where $o(s)$ is the source node of flow $s$ and $1_{i = o(s)}$ is an indicator function, which is 1 if $i = o(s)$, and 0, otherwise.

The data rate from the network layer to the link layer queues is $f^s_{i,j}(t)$. In particular, $f^s_{i,j}(t)$ is the actual rate of the packets, belonging to flow $s$, from the network layer queue; $U^s_i$ to the link layer queue; $V_{i,j}$ at node $i$. Note that the optimization of flow rate $f^s_{i,j}(t)$ is the routing decision, since it basically determines how many packets from flow $s$ should be forwarded (hence routed) to node $j$. At every timeslot $t$, $V_{i,j}$ changes according to the following queue dynamics.

$$V_{i,j}(t + 1) = \max[V_{i,j}(t) - h_{i,j}(t), 0] + \sum_{s \in \mathcal{S}} f^s_{i,j}(t)$$ (2)

The link transmission rate from $i$ to node $j$ is $h_{i,j}(t)$. As mentioned above $h_{i,j}(t)$ upper bounds per-flow data rates; i.e., $h_{i,j}(t) \geq \sum_{s \in \mathcal{S}} h^s_{i,j}(t)$. Note that the optimization of link transmission rate $h_{i,j}(t)$ corresponds to the scheduling decisions, since it determines which packets from which link layer queues should be transmitted as well as whether a link is activated.

B. Channel Model and Capacity Region

1) Channel Model: Consider one-hop transmission over link $l$, where $l = (i,j)$, such that $(i,j) \in \mathcal{N}$ and $i \neq j$. At each slot $t$, $C(t)$ is the channel state vector, where $C(t) = \{C_1(t), ..., C_l(t), ..., C_L(t)\}$. $C_l(t)$ is the state of the link $l$ at time $t$ and takes values from the set $\{ON, OFF\}$ according to a probability distribution which is i.i.d. over time slots. If $C_l(t) = ON$, packets are transmitted with rate $R_l$. Otherwise; (i.e., if $C_l(t) = OFF$, no packets are transmitted.

$$\Gamma_{C_l(t)}$$ denote the set of the link transmission rates feasible at time slot $t$ and for channel state $C_l(t)$. In particular, at every timeslot $t$, the link transmission vector $h(t) = \{h_1(t), ..., h_l(t), ..., h_L(t)\}$ should be constrained such that $h(t) \in \Gamma_{C_l(t)}$.

2) Capacity Region: Let $\Lambda_s$ is the vector of arrival rates $\forall s \in \mathcal{S}$. The network layer capacity region $\Lambda$ is defined as the closure of all arrival vectors that can be stably transmitted in the network, considering all possible routing and scheduling policies [1], [2], [3]. $\Lambda$ is fixed and depends only on channel statistics characterized by $\Gamma_{C_l(t)}$.  

2Note that in some devices, there might be only one queue (per-node queue) for data transmission instead of per-link queues in the link layer. Developing a model with per-node queues is challenging due to coupling among actions and states, so it is an open problem.
III. DIFF-MAX: FORMULATION AND DESIGN

A. Network Utility Maximization

In this section, we formulate and design the Diff-Max framework. Our first step is the NUM formulation of the problem and its solution. This approach (i.e., NUM formulation and its solution) sheds light into the structure of the Diff-Max algorithms. Note that the NUM formulation optimizes the average values of the parameters (i.e., flow rates) that are defined in Section II. By abuse of notation, we use a variable, e.g., \( \phi \) as the average value \( \phi(t) \) in our NUM formulation, if both \( \phi \) and \( \phi(t) \) refers to the same parameter.

1) Formulation: Our objective is to maximize the total utility function by optimally choosing the flow rates \( x_s \), \( \forall s \in S \), as well as the following variables at each node: the amount of data traffic that should be routed to each neighbor node; i.e., \( f_{i,j}^s \), the link transmission rates; i.e., \( h_{i,j} \).

\[
\begin{align*}
\text{max } & \sum_{s \in S} g_s(x_s) \\
\text{s.t. } & \sum_{j \in N} f_{i,j}^s - \sum_{j \in N} h_{j,i} = \begin{cases} x_s, & \text{if } i = o(s), \forall i \in N, s \in S \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, s \in S \\
& \sum_{s \in S} f_{i,j}^s \leq h_{i,j}, \forall (i,j) \in L \\
& f_{i,j}^s = h_{i,j}, \forall s \in S, (i,j) \in L \\
& h \in \bar{\Gamma}.
\end{align*}
\]

The first constraint is the flow conservation constraint at the network layer: at every node \( i \) and for each flow \( s \), the sum of the total incoming traffic, i.e., \( \sum_{j \in N} h_{j,i} \), and exogenous traffic, i.e., \( x_s \), should be equal to the total outgoing traffic from the network layer, i.e., \( \sum_{j \in N} f_{i,j}^s \). The second constraint is also the flow conservation constraint, but at the link layer; the link transmission rate; i.e., \( h_{i,j} \) should be larger than the incoming traffic; i.e., \( \sum_{s \in S} f_{i,j}^s \). Note that this constraint is inequality, because the link transmission rate can be larger than the actual data traffic. The third constraint shows the relationship between the network and link layer per-flow data rates. The last constraint shows that the vector of link transmission rates, \( h = \{h_1, ..., h_{i,l} \} \) should be the element of the available link rates; \( \bar{\Gamma} \). Note that \( \bar{\Gamma} \) is different than \( \Gamma_C(t) \) in the sense that \( \bar{\Gamma} \) is characterized with the loss probability over each link; \( p_l, \forall l \in L \), rather than the channel state vector; \( C(t) \).

The first and second constraints are key to our work, because they determine the incoming and outgoing flow relationships at the network and link layers, respectively. Such an approach separates routing from scheduling, and assigns the routing to the network layer and scheduling to the link layer. Note that if these constraints are combined in such a way that incoming rate from a node and exogenous traffic should be smaller than the outgoing traffic for each flow, we obtain the backpressure solution [13], [14].

2) Solution: By relaxing the first two flow conservation constraints in Eq. (3), we have:

\[
L(x, f, h, u, v) = \sum_{s \in S} g_s(x_s) + \sum_{i \in N} \sum_{s \in S} u_i^s \left( \sum_{j \in N} f_{i,j}^s \right) - \sum_{j \in N} h_{j,i}^s - x_s 1_{\{i = o(s)\}} - \sum_{(i,j) \in L} v_{i,j} \left( \sum_{s \in S} f_{i,j}^s - h_{i,j} \right),
\]

where \( u_i^s \) and \( v_{i,j} \) are the Lagrange multipliers, which can be interpreted as the representative of the network and link layer queues, \( U_i^s \) and \( V_{i,j} \), respectively. The Lagrange function can be re-written as:

\[
L(x, f, h, u, v) = \sum_{s \in S} g_s(x_s) - u_{o(s)} x_s + \sum_{i \in N} \sum_{s \in S} u_i^s f_{i,j}^s - \sum_{i \in N} \sum_{s \in S} u_i^s h_{i,j}^s - \sum_{(i,j) \in L} v_{i,j} f_{i,j}^s + \sum_{(i,j) \in L} v_{i,j} h_{i,j}
\]

Eq. (5) can be decomposed into several intuitive problems such as flow control, routing, and scheduling.

First, we solve the Lagrangian with respect to \( x_s \):

\[
x_s = (g_s^*)^{-1} \left( u_{o(s)} \right),
\]

where \( (g_s^*)^{-1} \) is the inverse function of the derivative of \( g_s \). This part of the solution is interpreted as the flow control.

Second, we solve the Lagrangian for \( f_{i,j}^s \) and \( h_{i,j}^s \). The following part of the solution is interpreted as the routing.

\[
\begin{align*}
\text{max } & \sum_{i \in N} \sum_{s \in S} \sum_{j \in N} \left( u_i^s f_{i,j}^s - u_j^s h_{j,i}^s \right) - \sum_{i \in N} \sum_{s \in S} v_{i,j} f_{i,j}^s \\
\text{s.t. } & f_{i,j}^s = h_{i,j}^s, \forall i \in N, j \in N, s \in S \\
& f_{i,j}^s \in \bar{\Gamma}.
\end{align*}
\]

The above problem is equivalent to:

\[
\begin{align*}
\text{max } & \sum_{(i,j) \in L} \sum_{s \in S} f_{i,j}^s (u_i^s - u_j^s - v_{i,j}) \\
\text{s.t. } & f_{i,j}^s \in \bar{\Gamma}.
\end{align*}
\]

Third, we solve the Lagrangian for \( h_{i,j} \). The following part of the solution is interpreted as scheduling.

\[
\begin{align*}
\text{max } & \sum_{(i,j) \in L} v_{i,j} h_{i,j} \\
\text{s.t. } & h \in \bar{\Gamma}.
\end{align*}
\]

The decomposed parts of the Lagrangian, i.e., Eqs. (6), (8), (9) as well as the Lagrange multipliers; \( u_i^s \) and \( v_{i,j} \) can be solved iteratively via a gradient descent algorithm. The convergence properties of this iterative algorithm are provided in Appendix A. Next, we propose Diff-Max based on the structure of the decomposed solution.

\[\text{Note that } u_i^s \text{ and } v_{i,j} \text{ are Lagrange multipliers. Although they are interpreted as the representation of the queue sizes, they are not actual queue sizes, but the functions of them. On the other hand, } U_i^s \text{ and } V_{i,j} \text{ are actual queue sizes.}\]
B. Diff-Max

Now, we provide stochastic control strategy including routing, scheduling, and flow control. The strategy, i.e., Diff-Max, which mimics the NUM solution, combines separated routing and scheduling together with the flow control strategy.

Diff-Max:

- **Routing.** Node $i$ observes the network layer queue backlogs in all neighboring nodes at time $t$ and determines:

$$f_{i,j}(t) = \begin{cases} F_{i,j}^{\text{max}}, & \text{if } U_{i}^{s}(t) - U_{j}^{s}(t) - V_{i,j}(t) > 0 \\ 0, & \text{otherwise} \end{cases}$$

(10)

where $F_{i,j}^{\text{max}}$ is constant larger than the maximum outgoing rate from node $i$. According to Eq. (10), $f_{i,j}(t)$ packets are removed from $U_{i}^{s}(t)$ and inserted in the link layer queue $V_{i,j}(t)$. This routing algorithm mimics Eq. (8) and has the following interpretation. Packets from flow $s$ can be transmitted to the next hop node $j$ as long as the network layer queue in the next hop (node $j$) is small, which means that node $j$ is able to route the packets, and the link layer queue at the current node (node $i$) is small, which means that the congestion over link $i-j$ is relatively small. Note that if the number of packets in $U_{i}^{s}(t)$ is limited, the packets are transmitted to the link layer queues beginning from the largest $U_{i}^{s}(t) - U_{j}^{s}(t) - V_{i,j}(t)$.

The routing algorithm in Eq. (10) uses per-link queues as well as per-flow queues, which is the main difference of Eq. (10) as compared to backpressure routing. The backpressure routing only uses per-flow queues, and does not take into account the state of the link layer queues (they do not exist due to formulation).

- **Scheduling.** At each time slot $t$, link rate $h_{i,j}(t)$ is determined by:

$$\max_{h} \sum_{(i,j) \in \mathcal{L}} V_{i,j}(t) h_{i,j}(t)$$

s.t. $h(t) \in \Gamma_{C}(t), \forall (i,j) \in \mathcal{L}$

(11)

This scheduling algorithm mimics Eq. (9) and has the following interpretation. The link $i-j$ with the largest queue backlog $V_{i,j}$, by taking into account the channel state vector $C(t)$, should be activated, and a packet(s) from the corresponding queue $V_{i,j}$ should be transmitted. We note that this problem (scheduling or max-weight) is known to be a hard problem, [9], [13]. Therefore, we propose sub-optimal scheduling algorithms that interact well with the routing algorithm in Eq. (10).

The scheduling algorithm in Eq. (11) differs from the classical backpressure in the sense that it is completely independent from the routing. In particular, Eq. (11) makes the scheduling decision based on per-link queues; $V_{i,j}$ and the channel state; $C(t)$, while the classical backpressure uses maximum queue backlog differences dictated by the routing algorithm. As it is seen the routing and scheduling are operating jointly in backpressure, while in Diff-Max, these algorithms are separated.

- **Flow Control.** At every time slot $t$, the flow/rate controller at the transport layer of node $i$ determines the current level of network layer queue backlogs $U_{i}^{s}(t)$ and determines the amount of packets that should be transported from the network layer to the network layer according to:

$$\max_{x} \sum_{s \in \mathcal{S}_{i} | \alpha(s)} [M g_{s}(x_{s}(t)) - U_{i}^{s}(t) x_{s}(t)]$$

s.t. $\sum_{s \in \mathcal{S}_{i} | \alpha(s)} x_{s}(t) \leq R_{i}^{\text{max}}$

(12)

where $R_{i}^{\text{max}}$ is a constant larger than the maximum outgoing rate from node $i$, and $M$ is a constant parameter, $M > 0$. The flow control part of our solution mimics Eq. (6) as well as the flow control algorithm proposed in [3].

The discussions on the analysis and performance bounds of Diff-Max are provided in Appendix B.

IV. SYSTEM IMPLEMENTATION

We propose practical implementations of Diff-Max (Fig. 3) as well as Diff-subMax, which combines the routing algorithm with a sub-optimal scheduling, and wDiff-subMax which makes routing decision based on a window-based algorithm.

A. Diff-Max

1) Flow Control: The flow control algorithm, implemented at the transport layer at the end nodes (see Fig. 3), determines the rate of each flow. We implement our flow control algorithm as an extension of UDP in our simulator ns-2 and in our Android testbed.

The flow control algorithm, at the source node $i$, divides time into epochs (virtual slots) such as $t_{i}^{1}, t_{i}^{2}, ..., t_{i}^{k_{i}}$, where $t_{i}^{k_{i}}$ is the beginning of the $k_{i}$th epoch. Let us assume that $t_{i}^{k_{i}+1} = t_{i}^{k_{i}} + T_{i}$, where $T_{i}$ is the epoch duration.

At time $t_{i}^{k}$, the flow control algorithm determines the rate according to Eq. (12). We consider $g_{s}(x_{s}(t)) = \log(x_{s}(t))$ (note that any other concave utility function can be used). After $x_{s}(t_{i}^{k})$ is determined, corresponding number of packets is passed to the network layer, and inserted to the network layer queue $U_{i}^{s}(t)$. Note that there might be some excessive packets at the transport layer if some packets are not passed to the
network layer. These packets are stored in a reservoir at the transport layer, and transmitted in later slots. At the receiver node, the transport protocol receives packets from the lower layers and passes them to the application.

2) Routing: The routing algorithm, implemented at the network layer of each node (both the end and intermediate nodes) (see Fig. 3), determines routing policy, i.e., the next hop(s) that packets are forwarded.

The first part of our routing algorithm is the neighbor discovery and queue size information exchange. Each node $i$ transmits a message containing the size of its network layer queues; $U_i^s$. These messages are in general piggy-backed to data packets. The nodes in the network operates on the promiscuous mode. Therefore, each node, let us say node $j$, overhears a packet from node $i$ even if node $i$ transmits the packet to another node, let us say node $k$. Node $j$ reads the queue size information from the data packet it receives or overhears (thanks to operating on the promiscuous mode). The queue size information is recorded for future routing decisions. Note that when a node hears from another node through direct or promiscuous mode, it classifies it as its neighbor. The neighbor nodes of node $i$ forms a set $N_i$. As we mentioned, queue size information is piggy-backed to data packets. However, if there is no data packet for transmission for some time duration, the node creates a packet to carry queue size messages and broadcast it.

The second part of our routing algorithm is the actual routing decision. Similar to the flow control algorithm, the routing algorithm divides time into epochs; such as $t_i^1, t_i^2, ..., t_i^{k}, ...$, where $t_i^k$ is the beginning of the $k$th epoch at node $i$. Let us assume that $t_i^{k+1} = t_i^k + T_i^k$ where $T_i^k$ is the epoch duration. Note that we use $t_i^k$ and $T_i^k$ instead of $t_i^k$ and $T_i$, because these two time epochs do not need to be the same nor synchronized.

At time $t_i^k$, the routing algorithm at the network layer checks $U_i^s(t_i^k) - U_i^j(t_i^k) - V_{i,j}(t_i^k)$ for each flow $s$. Note that $U_i^s(t_i^k)$ is not the instantaneous value of $U_i^s$ at time $t_i^k$, instead it is the latest value of $U_i^s$ heard by node $i$ before $t_i^k$. Note also that $V_{i,j}(t_i^k)$ is the per-link queue at node $i$, and this information should be passed to the network layer for routing decision. According to Eq. (10), $f_{i,j}(t_i^k)$ is determined, and $f_{i,j}(t_i^k)$ packets are removed from $U_i^s$ and inserted to the link layer queue $V_{i,j}$ at node $i$. Note that the link layer transmits packets from $V_{i,j}$ only to node $j$, hence the routing decision is completed. The routing algorithm is summarized in Algorithm 1. Note that Algorithm 1 considers that there are enough packets in $U_i^s$ for transmission. If not, the algorithm lists all the links $j \in N_i$ in decreasing order, according to the weight; $U_i^s(t_i^k) - U_i^j(t_i^k) - V_{i,j}(t_i^k)$. Then, it begins to route packets beginning from the link that has the largest weight.

3) Scheduling: The scheduling algorithm in Eq. (11) assumes that time is slotted, and determines the links that should be activated and the (number of) packets that should be transmitted at each time slot. Although there are time-slotted system implementations, and also recent work on backpressure implementation over time-slotted wireless networks [8], IEEE 802.11 MAC, an asynchronous medium access protocol without time slots, is the most widely used MAC protocol in the current wireless networks. Therefore, we implement our scheduling algorithm (Eq. (11)) on top 802.11 MAC (see Fig. 3) with the following updates.

The scheduling algorithm constructs per-link queues at the link layer. Node $i$ knows its own link layer queues, $V_{i,j}$, and estimates the loss probability and link rates. Let us consider that $p_l$ and $R_l$ are the estimated values of $p_l$ and $R_l$, respectively. $p_l$ is calculated as one minus the ratio of correctly transmitted packets over all transmitted packets in a time window over link $l$, $R_l$ is calculated as the average of the recent (in a window of time) link rates over link $l$. $V_{i,j}$, $p_{i,j}$, and $R_{i,j}$ are piggy-backed to the data packets and exchanged among nodes. Note that this information should be exchanged among all nodes in the network since each node is required to make its own decision based on global information. Also, each node knows the general topology and interfering links.

The scheduling algorithm that we implemented mimics Eq. (11). Each node $i$ knows per-link queues, i.e., $V_l$, estimated loss probabilities, i.e., $p_l$, and link rates, i.e., $R_l$, for $l \in \mathcal{L}$ as well all maximal independent sets, which consist of links that are not interfering. Let us assume that there are $Q$ maximal independent sets. For the $q$th maximal independent set such that $q = 1, ..., Q$, the policy vector is; $\pi_q = \{\pi^l_q, ..., \pi^t_q, ..., \pi^L_q\}$, where $\pi^l_q = 1$ if link $l$ is in the $q$th maximal set, and $\pi^L_q = 0$, otherwise. Our scheduling algorithm selects $q^*$th maximal independent set such that $q^* = \arg \max_{q} \{\sum_{l \in \mathcal{L}} V_l(1 - p_l)R_l \pi^l_q\}$. Node $i$ solves $q^*$ as one of the parameters; $V_i, p_i, R_i$ change $\forall l \in \mathcal{L}$. If, according to $q^*$, node $i$ decides that it should activate one of its links, then it reduces the contention window size of 802.11 MAC so that node $i$ can access the medium quickly and transmit a packet. If node $i$ should not transmit, then the scheduling algorithm tells 802.11 MAC that there are no packets in the queues available for transmission. Note that we update 802.11 MAC protocol so that we can implement the scheduling algorithm in Diff-Max. The scheduling algorithm is summarized in Algorithm 2.

Note that Algorithm 2 is a hard problem, because it reduces

---

4Note that we do not use instantaneous channel states $C(t)$ in our implementation, since it is not practical to get this information. Even if one can estimate $C(t)$ using physical layer learning techniques, $C(t)$ should be estimated $\forall t \in \mathcal{L}$, which is not practical in current wireless networks.
Algorithm 2 Diff-Max scheduling algorithm at node $i$.

1: if $V_i \neq \emptyset$ or $R_i$ is updated such that $t \in L$ then
2: Determine $q'$ such that $q' = \arg \max_{q \in \mathcal{C}} \left\{ \sum_{l \in \mathcal{L}} V_l (1 - \tilde{p}_l) R_l \pi_i^l \right\}$
3: if $\exists (i, j)$ such that $\pi_i^{i,j} = 1$, $\forall j \in N_i$, then
4: Reduce 802.11 MAC contention window size and access the medium
5: Transmit a packet from $V_i \rightarrow j$ according to FIFO rule
6: else
7: Tell 802.11 MAC that there are no packets in the queues available for transmission

Algorithm 2 Diff-Max scheduling algorithm at node $i$.

1: if $V_i \neq \emptyset$ or $R_i$ is updated such that $t \in L$ then
2: Determine $q'$ such that $q' = \arg \max_{q \in \mathcal{C}} \left\{ \sum_{l \in \mathcal{L}} V_l (1 - \tilde{p}_l) R_l \pi_i^l \right\}$
3: if $\exists (i, j)$ such that $\pi_i^{i,j} = 1$, $\forall j \in N_i$, then
4: Reduce 802.11 MAC contention window size and access the medium
5: Transmit a packet from $V_i \rightarrow j$ according to FIFO rule
6: else
7: Tell 802.11 MAC that there are no packets in the queues available for transmission

Algorithm 2 Diff-Max scheduling algorithm at node $i$.
The throughput of flow; $S_1 - R_1$ and from node $A$ to $D$ ($S_2 - R_2$).

(c) Grid topology. 12 nodes are randomly placed over $4 \times 3$ grid. An example node distribution and possible flows are illustrated in the figure.

Fig. 4. Topologies used in simulations. (a) Triangle topology. There are two flows between sources; $S_1$, $S_2$ and receivers; $R_1$, $R_2$, i.e., from node $A$ to $B$ ($S_1 - R_1$) and from node $A$ to $C$ ($S_2 - R_2$). (b) Diamond topology. There are two flows between sources; $S_1$, $S_2$ and receivers; $R_1$, $R_2$, i.e., from node $A$ to $B$ ($S_1 - R_1$) and from node $A$ to $D$ ($S_2 - R_2$). (c) Grid topology. 12 nodes are randomly placed over $4 \times 3$ grid. An example node distribution and possible flows are illustrated in the figure.

Fig. 5. Triangle topology shown in Fig. 4(a). The loss is over link $A-C$. (a) Total throughput (sum of the throughput of flows from $S_1$ to $R_1$ and $S_2$ to $R_2$) vs. loss probability. (b) Throughput of flow from $S_1$ to $R_1$ vs. loss probability. (c) Throughput of flow from $S_2$ to $R_2$ vs. loss probability.

Fig. 6. Triangle topology shown in Fig. 4(a). The loss is over all links. (a) Total throughput (sum of the throughput of flows from $S_1$ to $R_1$ and $S_2$ to $R_2$) vs. loss probability. (b) Throughput of flow from $S_1$ to $R_1$ vs. loss probability. (c) Throughput of flow from $S_2$ to $R_2$ vs. loss probability.

Fig. 7. Diamond topology shown in Fig. 4(b). The loss is over link $A - B$. (a) Total throughput (sum of the throughput of flows from $S_1$ to $R_1$ and $S_2$ to $R_2$) vs. loss probability. (b) Throughput of flow from $S_1$ to $R_1$ vs. loss probability. (c) Throughput of flow from $S_2$ to $R_2$ vs. loss probability.

1) Setup: We considered two topologies: diamond topology shown in Fig. 4(b) and a grid topology shown in Fig. 4(c). In the diamond topology, the nodes are placed over $500m \times 500m$ terrain. Two flows are transmitted from node $A$ to nodes $B$ and $D$. In the grid topology, $4 \times 3$ cells are placed over a $800m \times 600m$ terrain. 12 nodes are randomly placed to the cells. In the grid topology, each node can communicate with other nodes in its cells or with the ones in neighboring cells.
Four flows are generated randomly.

We consider CBR traffic. CBR flows start at random times within the first 5 sec and are on until the end of the simulation which is 100 sec. The CBR flows generate packets with inter-arrival times 0.01 ms. IEEE 802.11b is used in the MAC layer (with updates for Diff-Max implementation as explained in Section IV). In terms of wireless channel, we simulated a Rayleigh fading channel with average channel loss rates 0, 20, 30, 40, 50%.5 We have repeated each 100 sec simulation for 10 seeds.

The channel capacity is 1 Mbps, the buffer size at each node is set to 1000 packets, packet sizes are set to 1000B. We compare our schemes; Diff-Max, Diff-subMax, and wDiff-subMax with AODV, in terms transport-level throughput.

The Diff-Max parameters are set as follows. For the flow control algorithm; $T_i = 80 ms$, $R_i^{max} = 20$ packets, $M = 200$. For the routing algorithm; $T_i = 10 ms$, $F_i^{max} = 4$ packets.

2) Results: Fig. 9, presents simulation results in ns-2 simulator over diamond and grid topologies for different loss rates.

Fig. 9(a) shows the results for the diamond topology. The loss rate is over the link between nodes $A$ and $B$. Diff-Max performs better than the other schemes for the range of loss rates. The reason is that Diff-Max activates links based on per-link queue backlogs, loss rates, and link rates. On the other hand, Diff-subMax, wDiff-subMax and AODV uses classical 802.11 MAC, which provides fairness among the competing nodes for the medium, which is not utility optimal. When the loss rate over link $A - B$ increases, the total throughput of all the schemes reduces as expected. As it can be seen, the decrease of our schemes; Diff-Max, Diff-subMax, wDiff-subMax is linear, while the decrease of AODV is quite sharp. The reason is that when AODV experiences loss over a path, it deletes the path and re-calculates new routes. Therefore, AODV does not transmit over lossy links for some time period and tries to find new routes, which reduces throughput.

Fig. 9(b) elaborates more on the above discussion. It shows the throughput of two flows $A$ to $B$ and $A$ to $D$ as well as their total value when the loss rate is 10% over link $A - B$. As it can be seen, the rate of flow $A - B$ is very low in AODV as compared to our schemes, because AODV considers the link $A - B$ is broken at some periods during the simulation, while our schemes continue to transmit over this link.

Let us consider Fig. 9(a) again. Diff-subMax and wDiff-subMax improve throughput significantly as compared to AODV thanks to exploring routes to improve utility (hence throughput). The improvement of our schemes over AODV is up to 22% in this topology. Also, Diff-subMax and wDiff-subMax have similar throughput performance, which emphasis the benefit of routing part and the effective link layer queue estimation mechanism of wDiff-subMax.

Fig. 9(a) also shows that when loss rate is 50%, the throughput improvement of all schemes are similar, because at 50% loss rate, link $A - B$ becomes very inefficient, and all of the schemes transmit packets mostly from flow $A$ to $D$ over path $A - C - D$ and have similar performance at high loss rates.

Fig. 9(b) shows the results for the grid topology. The throughput improvement of our schemes is higher than AODV for all loss rates in the grid topology and higher as compared to the improvement in the diamond topology, e.g., the improvement is up to 33% in the grid topology. The reason is that AODV is designed to find the shortest paths, but our schemes are able to explore interference free paths even if they are not the shortest paths, which is emphasized in larger topologies.

C. Android Prototype

We consider a scenario in which a group of smartphones collaborate in the same geographical area. In our setting, we use four Android 4.0 [12] based Galaxy Nexus phones, and configure them to operate in ad-hoc mode over Wifi. We implement our wDiff-subMax scheme (flow control and routing) as an extension of UDP socket.

We first consider a scenario in which two phones ($A$ and $B$) are connected to each other. Phone $A$ transmits 4MB audio file to phone $B$. The transmission time for wDiff-subMax was 16 sec which is comparable with its TCP counterpart, which was 14 sec. This example shows the efficiency of our algorithm as an extension of UDP, which causes packet losses or too long transmission times.

In the second scenario, we placed/ separated phones to be able to create a topology similar to the diamond topology shown in Fig. 4(b). In this setup, phone $A$ transmits 4MB...
audio file to phone $D$ either using phone $B$ or $C$ as a relay. We first consider TCP connection over the path $A \rightarrow B \rightarrow D$ and configure phone $B$ so that it drops relaying packets after $10\text{sec}$ transmission. As expected, TCP connection fails when $B$ stops relaying packets. On the other hand, wDiff-subMax continues transmission even after $B$ stops, by relaying packets using phone $C$, and completes the transmission in $40\text{sec}$.

VI. RELATED WORK

*Backpressure and follow-up work.* This paper builds on backpressure, a routing and scheduling framework over communication networks [1], [2], which has generated a lot of interest in the research community [15]; especially for wireless ad-hoc networks [18], [19], [20], [21], [22], [23]. Also, it has been shown that backpressure can be combined with flow control to provide utility-optimal operation guarantee [3], [22]. This paper follows the main idea of backpressure framework, and revisit it considering the practical challenges that are imposed by the current networks.

*Backpressure implementation.* The strengths of the backpressure framework have recently increased the interest on practical implementation of backpressure over wireless networks. Multi-path TCP scheme is implemented over wireless mesh networks [6], where TCP flows are transmitted over multiple pre-determined paths and packets are scheduled according to backpressure scheduling algorithm. At the link layer, [4], [5], [24], [25] propose, analyze, and evaluate link layer backpressure-based implementations with queue prioritization and congestion window size adjustment. The backpressure framework is implemented over sensor networks [7] and wireless multi-hop networks [8], which are also the most close implementations to ours. Our main differences are that; (i) we consider separation of routing and scheduling to make practical implementation easier, (ii) we design and analyze a new scheme; Diff-Max, (iii) we simulate and implement Diff-Max over ns-2 and android phones.

*Backpressure and Queues.* According to backpressure framework, each node constructs per-flow queues. There is some work in the literature to stretch this necessity. For example, [26], [27] propose using real per-link and virtual per-flow queues. Such a method reduces the number of queues required in each node, and reduces the delay. Although this approach reduces the backpressure framework to make routing decision using virtual queues and scheduling decision using the real per-link queues by decoupling routing and scheduling, it does not separate routing from scheduling. Therefore, this approach requires strong synchronization between the network and link layers, which is difficult to implement in practice as explained in Section I.

VII. CONCLUSION

In this paper, we proposed Diff-Max, a framework that separates routing and scheduling in backpressure-based wireless networks. Diff-Max improves throughput significantly. Also, the separation of routing and scheduling makes practical implementation easier by minimizing cross-layer operations and it leads to modularity. Our design is grounded on a network utility maximization (NUM) formulation of the problem and its solution. Simulations in ns-2 demonstrate the performance of Diff-Max as compared adaptive routing schemes, such as AODV. The evaluations on an android testbed confirm the efficiency and practicality of our approach.

REFERENCES


[12] Android 4.0, Ice Cream Sandwich, developer.android.com/about/versions/android-4.0-highlights.html

**APPENDIX A**

**Lagrange Multipliers.** The Lagrange multipliers; \( u_i \) and \( v_{ij} \) are calculated using gradient descent:

\[
\begin{align*}
    u_i(t+1) &= (u_i(t) - \alpha_t \sum_{j \in N} f_{ij}(t) - \sum_{j \in N} h_{ij}(t)) + \\
    v_{ij}(t+1) &= (v_{ij}(t) + \beta_t \sum_{s \in S} f_{sij}(t) - h_{ij}(t))
\end{align*}
\]

where \( t \) is the iteration number, \( \alpha_t \) and \( \beta_t \) are the step sizes of the gradient descent algorithm, and the \( \top \) operator makes the Lagrange multipliers positive.

**Convergence.** The convergence of the solution set, Eqs. (6), (8), (9), (13) follows directly from the convergence of convex optimization problems through gradient descent [14], [28]. In particular, if \( \lim_{t \to \infty} \alpha_t = 0, \sum_{t=0}^{\infty} \alpha_t = \infty \) and \( \lim_{t \to \infty} \beta_t = 0 \), then the solution converges, i.e., \( \lim_{t \to \infty} \| x(t) - x^* \| = 0 \).

**Numerical Calculations.** We confirm the convergence of our solution through numerical calculations. Fig. 10 shows total throughput (sum of the two flows) vs. iteration number graphs for the triangle topology shown in Fig. 4(a) and for different loss probabilities, i.e., 0.2, 0.5, and 0.8. It is seen that the total throughput converges to the optimal value for all loss rates. We repeated the same simulations for the diamond topology whose results are shown in Fig. 11. These results also confirm the convergence of our solution.

**APPENDIX B**

Let us recall the queue dynamics in Eq. (1) and Eq. (2):

\[
U_i(t+1) = \max[U_i(t) - \sum_{j \in N} f_{ij}(t), 0] + \sum_{j \in N} h_{ij}(t) + x_s(t)1_{i = o(s)}
\]

\[
V_{ij}(t+1) = \max[V_{ij}(t) - h_{ij}(t), 0] + \sum_{s \in S} f_{sij}(t) + x_s(t)1_{i = o(s)}
\]

Now let us consider a virtual queue \( Z_i^v(t) \):

\[
Z_i^v(t+1) = \max[Z_i^v(t) - \sum_{j \in N} f_{ij}(t), 0] + \sum_{j \in N} f_{ij}(t) + x_s(t)1_{i = o(s)}
\]

In the following, we consider a variant of Diff-Max algorithm using the virtual queues \( Z_i^v(t) \) instead of \( U_i^v(t) \).

- **Diff-Max with virtual queues:**
  - **Routing.** Node \( i \) observes the network layer queue backlogs in all neighboring nodes at time \( t \) and determines:
    \[
    f_{ij}(t) = \begin{cases} 
    F_{ij}^\text{max}, & \text{if } Z_i^v(t) - Z_j^v(t) - V_{ij}(t) > 0 \\
    0, & \text{otherwise}
    \end{cases}
    \]
  - **Scheduling.** At each time slot \( t \), link rate \( h_{ij}(t) \) is determined by:
    \[
    \max_{h} \sum_{(i,j) \in \mathcal{L}} V_{ij}(t) h_{ij}(t)
    \]
    subject to:
    \[
    \mathbf{h}(t) \in \mathcal{G}(t), \forall (i,j) \in \mathcal{L}
    \]

**Theorem 1:** If channel states are i.i.d. over timeslots, and the arrival rates \( \lambda_s \), \( s \in \mathcal{S} \) are interior to the capacity region \( \Lambda \), then Diff-Max with virtual queues stabilizes the network and the total average backlog is bounded.

**Proof:** We first define the Lyapunov function as:

\[
L(H(t)) = \sum_{i \in N} \sum_{s \in S} Z_i^v(t)^2 + \sum_{i \in N} \sum_{j \in N} V_{ij}(t)^2,
\]

Note that we consider in this section that arrival rates are inside the capacity region. The extension of the analysis provided in this section is straightforward when the arrival rates are outside the capacity region by using an appropriate flow control algorithm. Some remarks regarding this issue are added at the end of this section.
where \( H(t) = \{Z(t), V(t)\}, Z(t) = \{Z_i(t)\}_{i \in N, s \in S}, \) and \( V(t) = \{V_{i,j}(t)\}_{i \in N, j \in N}. \)

We now evaluate the Lyapunov drift [3, 15]:

\[
\Delta(H(t)) = E[L(H(t+1)) - L(H(t))|H(t)]
\]

\[
\Delta(H(t)) \leq E[|\sum_{i \in N} \sum_{s \in S} (Z_i(t+1))|^2 + \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t+1))^2
- \sum_{i \in N} \sum_{s \in S} (Z_i(t))^2 - \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t))^2|H(t)]
\]

Using the fact that \((\max(Q-b, 0) + A)^2 \leq Q^2 + A^2 + b^2 + 2Q(A-b),\) we have;

\[
\Delta(H(t)) \leq E[|\sum_{i \in N} \sum_{s \in S} (Z_i(t))^2 + \sum_{j \in N} f_{i,j}(t)]^2
+ (\sum_{j \in N} f_{i,j}(t) + x_s(t)1_{[i=0,s]}))^2
+ 2Z_i(t)(\sum_{j \in N} f_{i,j}(t) + x_s(t)1_{[i=0,s]})
- \sum_{j \in N} f_{i,j}(t))^2 + \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t))^2 + (h_{i,j}(t))^2
- \sum_{j \in N} f_{i,j}(t))^2 + \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t))^2 + (h_{i,j}(t))^2
- \sum_{i \in N} \sum_{s \in S} (Z_i(t))^2 - \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t))^2|H(t)]
\]

\[
\Delta(H(t)) \leq B + 2E[|\sum_{i \in N} \sum_{j \in N} \sum_{s \in S} Z_i(t)(f_{i,j}(t) - f_{i,j}(t))
+ \sum_{i \in N} \sum_{j \in N} V_{i,j}(t)1_{[i=0,s]} + \sum_{i \in N} \sum_{j \in N} V_{i,j}(t)1_{[i=0,s]}
- \sum_{i \in N} \sum_{j \in N} V_{i,j}(t)h_{i,j}(t)|H(t)]
\]

\[
\Delta(H(t)) \leq E[\sum_{i \in N} \sum_{s \in S} (\sum_{j \in N} f_{i,j}(t))^2
+ \sum_{i \in N} \sum_{s \in S} (\sum_{j \in N} f_{i,j}(t)) + x_s(t)1_{[i=0,s]}))^2
+ \sum_{i \in N} \sum_{j \in N} (h_{i,j}(t))^2 + \sum_{i \in N} \sum_{j \in N} (h_{i,j}(t))^2
- \sum_{i \in N} \sum_{j \in N} (V_{i,j}(t))^2|H(t)]
\]
\[ \Delta(H(t)) \leq B - 2E \left[ \sum_{i \in N} \sum_{j \in S} f_{i,j}(t) (Z_i(t) - Z_j(t) - V_{i,j}(t)) | H(t) \right] \\
- 2E \left[ \sum_{i \in N} \sum_{j \in S} V_{i,j}(t) h_{i,j}(t) | H(t) \right] \\
+ 2E \left[ \sum_{i \in N} \sum_{j \in S} Z_i(t) x_s(t) 1_{i = o(s)} | H(t) \right] \\
(25) \]

Note that minimizing the upper bound on the drift (Eq. 25) corresponds to maximizing the terms: (i) \( E \sum_{i \in N} \sum_{j \in S} f_{i,j}(t) (Z_i(t) - Z_j(t) - V_{i,j}(t)) | H(t) \). This maximization is equivalent to our routing algorithm in Eq. (34). (ii) \( E \sum_{i \in N} \sum_{j \in S} V_{i,j}(t) h_{i,j}(t) | H(t) \). This maximization is equivalent to our scheduling algorithm in Eq. (35). The drift inequality in Eq. (25) is expressed as:

\[ \Delta(H(t)) \leq B - 2 \sum_{i \in N} \sum_{j \in S} Z_i(t) (\lambda_s 1_{i \in o(s)} + \epsilon_1) - 2 \sum_{i \in N} \sum_{j \in N} V_{i,j}(t) \epsilon_2 \\
+ 2 \sum_{i \in N} \sum_{j \in S} Z_i(t) \lambda_s 1_{i \in o(s)} \]

(26)

If the vector of arrival rates are interior to the capacity region, there always exist (i) \( \epsilon_1 > 0 \) such that \( E \sum_{i \in N} \sum_{j \in S} (\sum_{j \in N} f_{i,j}(t) - Z_i(t) - V_{i,j}(t)) | H(t) \) \( \leq - (\lambda_s 1_{i \in o(s)} + \epsilon_1) \). (ii) \( \epsilon_2 \geq 0 \) such that \( E \sum_{i \in N} \sum_{j \in N} (\sum_{j \in S} f_{i,j}(t) - h_{i,j}(t)) | H(t) \) \( \leq - \epsilon_2 \).

Substituting these into Eq. (26):

\[ \Delta(H(t)) \leq B - 2 \sum_{i \in N} \sum_{j \in S} Z_i(t) (\lambda_s 1_{i \in o(s)} + \epsilon_1) - 2 \sum_{i \in N} \sum_{j \in N} V_{i,j}(t) \epsilon_2 \\
+ 2 \sum_{i \in N} \sum_{j \in S} Z_i(t) \lambda_s 1_{i \in o(s)} \]

(27)

\[ \Delta(H(t)) \leq B - 2 \sum_{i \in N} \sum_{s \in S} Z_i(t) \epsilon_1 - 2 \sum_{i \in N} \sum_{j \in N} V_{i,j}(t) \epsilon_2 \]

(28)

The time average of Eq. (28) yields

\[ \limsup_{t \to \infty} \frac{1}{t} \sum_{t=0}^{t-1} \left( \sum_{i \in N} \sum_{j \in S} Z_i(t) \epsilon_1 + \sum_{i,j \in L} V_{i,j}(t) \epsilon_2 \right) \leq B, \]

(29)

which shows that the time average of \( Z_i(t) \) is bounded. Note that, since \( \epsilon_2 \geq 0 \), Eq. (28) does not directly show that the time average of \( V_{i,j}(t) \) is bounded. Yet, Eq. (28) implicitly shows that the time average of \( V_{i,j}(t) \) is bounded. Note that \( Z_i(t) \) includes both \( U_i(t) \) and \( V_{i,j}(t) \). Therefore, since the time average of \( Z_i(t) \) is bounded, the time averages of \( U_i(t) \) and \( V_{i,j}(t) \) are bounded. Thus, Diff-Max algorithm with virtual queues stabilizes the network and the total average backlog is bounded.

Theorem 2: If channel states are i.i.d. over timeslots, the arrival rates \( \lambda_s \), \( s \in S \) are interior to the capacity region \( \Lambda \), the link layer queues are bounded, i.e., \( V_{i,j}(t) \leq K \), \( \forall i,j \in N \) and for some finite real number \( K \), then \( U_i(t) \) is used in Eq. (34) instead of \( Z_i(t) \). The resulting algorithm, i.e., Diff-Max (Eqs. (10), (11)) stabilizes the network and the total average queue backlog is bounded.

Proof: Note that \( \sum_{s \in S} Z_i(t) = \sum_{s \in S} U_i(t) + \sum_{j \in N} V_{i,j}(t) \), \( \forall i,j \in N \). If \( V_{i,j}(t) \leq K \), \( \forall i,j \in N \) and for some finite real number \( K \), then the difference between \( Z_i(t) \) and \( U_i(t) \) is finite at each slot \( t \). Let us consider \( U_i(t) \) as an estimator of \( Z_i(t) \). It is known that the estimator (\( U_i(t) \)) of the real queue (\( Z_i(t) \)) can be used instead of the real queue in the algorithms as long as the difference between the estimator and the real queue is finite at each slot \( t \), and this stabilizes the queues [15]. Therefore, if \( V_{i,j}(t) \leq K \), \( \forall i,j \in N \) and for some finite real number \( K \), \( U_i(t) \) can be used instead of \( Z_i(t) \). The resulting algorithm, i.e., Diff-Max stabilizes with virtual queues and packet dropping is summarized in the following.

Diff-Max with virtual queues and packet dropping:

- **Routing.** Node \( i \) observes the network layer queue backlogs in all neighboring nodes at time \( t \) and determines:

\[ f_{i,j}(t) = \begin{cases} F_{\text{max}} & \text{if } Z_i(t) - Z_j(t) - V_{i,j}(t) > 0 \\ 0 & \text{otherwise} \end{cases} \]

(30)

- **Packet Dropping.** If \( V_{i,j}(t) > K \), \( f_{i,j}(t) \) packets are dropped from \( V_{i,j}(t) \), \( \forall i,j \in N \).

- **Scheduling.** At each time slot \( t \), link rate \( h_{i,j}(t) \) is determined by:

\[ \max_{h_{i,j}(t)} \sum_{(i,j) \in L} V_{i,j}(t) h_{i,j}(t) \]

s.t. \( h(t) \in \Gamma_C(t), \forall (i,j) \in L \)

(31)
Theorem 3: If channel states are i.i.d. over timeslots, and the arrival rates $\lambda_s, \forall s \in S$ are interior to the capacity region $\Lambda$, then Diff-Max with virtual queues and packet dropping stabilizes the network and the total average backlog is bounded.

Proof: Due to packet dropping, the queue evolution inequalities are updated as follows.

$$Z_i^s(t+1) \leq \max[Z_i^s(t) - \sum_{j \in N} f^s_{i,j}(t), 0] + \sum_{j \in N} f^s_{i,j}(t)1_{[V_j^s(t) \leq K]} + x_s(t)1_{[i = o(s)]}$$

(32)

$$V_{i,j}(t+1) \leq [V_{i,j}(t) - h_{i,j}(t), 0] + \sum_{s \in S} f^s_{i,j}(t)1_{[V_{i,j}(t) \leq K]}$$

(33)

Let us consider Eq. (32). The outgoing traffic from $Z_i^s(t)$ is $\sum_{j \in N} f^s_{i,j}(t)$, even if some of these packets may be dropped at $V_{i,j}(t)$, $\forall j \in N$. The incoming traffic to $Z_i^s(t)$ is $\sum_{j \in N} f^s_{i,j}(t)1_{[V_j^s(t) \leq K]}$ where $1_{[V_j^s(t) \leq K]} = 1$ if $V_j^s(t) \leq K$, and $1_{[V_j^s(t) \leq K]} = 0$ otherwise. Since $1_{[V_j^s(t) \leq K]} \leq 1$, Eq. (32) can be updated as; $Z_i^s(t+1) \leq \max[Z_i^s(t) - \sum_{j \in N} f^s_{i,j}(t), 0] + \sum_{j \in N} f^s_{i,j}(t)1_{[i = o(s)]}$ which is equivalent to Eq. (16).

Now, let us consider Eq. (33). The incoming traffic to $V_{i,j}(t)$ is $f^s_{i,j}(t)1_{[V_j^s(t) \leq K]}$. Since $1_{[V_j^s(t) \leq K]} \leq 1$, Eq. (33) can be expressed as $V_{i,j}(t+1) \leq [V_{i,j}(t) - h_{i,j}(t), 0] + \sum_{s \in S} f^s_{i,j}(t)$, which is equivalent to Eq. (15).

Since Eq. (16) and Eq. (15) are valid queue expressions for Diff-Max with virtual queues and packet dropping, the analysis in the proof of Theorem 1 holds for Diff-Max with virtual queues and packet dropping, and Diff-Max with virtual queues and packet dropping stabilizes the queues.

Corollary 1: If channel states are i.i.d. over timeslots, and the arrival rates $\lambda_s, \forall s \in S$ are interior to the capacity region $\Lambda$, then Diff-Max with packet dropping, which is summarized in the following, stabilizes the network and the total average backlog is bounded.

Diff-Max with packet dropping:

- **Routing.** Node $i$ observes the network layer queue backlogs in all neighboring nodes at time $t$ and determines;

$$f^s_{i,j}(t) = \begin{cases} F_{i,j}^\max, & \text{if } U_{i,j}^s(t) - U_j^s(t) - V_{i,j}(t) > 0 \\ 0, & \text{otherwise} \end{cases}$$

(34)

- **Packet Dropping.** If $V_{i,j}(t) > K$, $f^s_{i,j}(t)$ packets are dropped from $V_{i,j}(t)$, $\forall i, j \in N$.

- **Scheduling.** At each time slot $t$, link rate $h_{i,j}(t)$ is determined by;

$$\max_{h} \sum_{(i,j) \in \mathcal{L}} V_{i,j}(t)h_{i,j}(t)$$

s.t. $h(t) \in \Gamma_{C(t)}, \forall (i,j) \in \mathcal{L}$

(35)

Proof: The proof directly follows from Theorem 2 and Theorem 3.

Remarks on Diff-Max with packet dropping: Note that Diff-Max with packet dropping stabilizes queues when the arrival rates $\lambda_s, \forall s \in S$ are interior to the capacity region $\Lambda$. We would like to make two comments on this:

- Noting that depending on the value of $K$, some packets are dropped and may not arrive to their destination. Let us assume an extreme case that $K = 0$. In this case, all packets are dropped at the link layer queues ($V_{i,j}(t), \forall i, j \in N$). In this case, no packets can be delivered to their destinations. When $K$ increases, the number of dropped packets decreases. For large values of $K$, very few or no packets dropped. Therefore, for large values of $K$, Diff-Max with packet dropping performs similar to Diff-Max, i.e., very few or no packets are dropped.

- The analysis provided in this section on the performance of Diff-Max with virtual queues, Diff-Max with virtual queues and packet dropping, and Diff-Max with packet dropping directly holds when the arrival rates are outside the capacity region by employing a flow control algorithm. Noting that flows are generated by only one node in our setup, then $Z_i^s(t) = U_i^s(t)$ if $i = o(s)$, because $V_{i,i}^s(t) = 0$ if $i = o(s)$. Considering this fact, and using the Drift-Plus-Penalty considered in [15], the flow control algorithm in Eq. (12) is derived.