Diff-Max: Separation of Routing and Scheduling in Backpressure-Based Wireless Networks

Hulya Seferoglu and Eytan Modiano

Laboratory For Information and Decision Systems

Massachusetts Institute of Technology

{hseferog, modiano}@mit.edu

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.
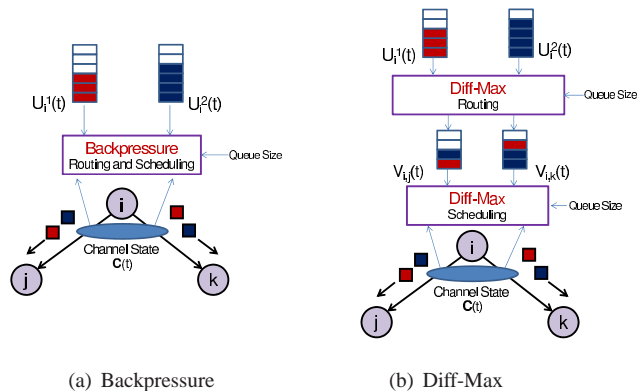
(a) Backpressure   (b) Diff-Max

Fig. 1. Example topology consisting of three nodes; i, j, k, and two flows; 1, 2. Note that this small topology is a zoomed part of a large multi-hop wireless network. The source and destination nodes of flows 1 and 2 are not shown in this example, *i.e.,* nodes i, j, k are intermediate nodes which route and schedule flows 1 and 2. U_i^1 and U_i^2 are per-flow queue sizes and $V_{i,j}$ and $V_{i,k}$ are per-link queue sizes. (a) Backpressure: Node i determines queue backlog differences at time t; $D_{i,j}^s(t) = U_i^s(t) - U_j^s(t)$, $D_{i,k}^s(t) = U_i^s(t) - U_k^s(t)$, where $s \in \{1, 2\}$. Based on these differences as well as the channel state of the network, $\boldsymbol{C}(t)$, it makes joint routing and scheduling decisions. (b) Diff-Max: Node i makes routing decision based on the queue backlog differences at time t; $\tilde{D}_{i,j}^s(t) = U_i^s(t) - U_j^s(t) - V_{i,j}(t)$, $\tilde{D}_{i,k}^s(t) = U_i^s(t) - U_k^s(t) - V_{i,k}(t)$, where $s \in \{1, 2\}$. Separately, node i makes the scheduling decision based on $V_{i,j}(t)$, $V_{i,k}(t)$ and $\boldsymbol{C}(t)$.

Example 1: Let us consider Fig. 1(a) for backpressure operation. At time t, node i makes routing and scheduling decisions for flows 1 and 2 based on the per-flow queue sizes; $U_i^1(t)$, $U_i^2(t)$, as well as the queue sizes of the other nodes, *i.e.,* node j and k in this example, and using the channel state of the network $\boldsymbol{C}(t)$. In particular, the backpressure determines the flow that should be transmitted over link $i - j$ by $s^* = \arg\max\{D_{i,j}^1(t), D_{i,j}^2(t)\}$ such that $s^* \in \{1, 2\}$. The decision mechanism is the same for link $i - k$. Note that this is joint routing (*i.e.,* the next hop decision) and scheduling (*i.e.,* the flow selection for transmission). The scheduling algorithm also determines the link activation policy. In particular, the maximum backlog differences over each link are calculated as; $D_{i,j}(t) = D_{i,j}^{s^*}(t)$ and $D_{i,k}(t) = D_{i,k}^{s^*}(t)$. Based on $D_{i,j}^*(t), D_{i,k}^*(t)$ and $\boldsymbol{C}(t)$, the scheduling algorithm determines the link that should be activated. Note that the decisions of routing and scheduling (also named as max-weight algorithm) are made jointly in the backpressure framework, which imposes several challenges in practice. We elaborate on them next. ∎

Routing algorithms are traditionally designed in the network

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-Max[1], 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
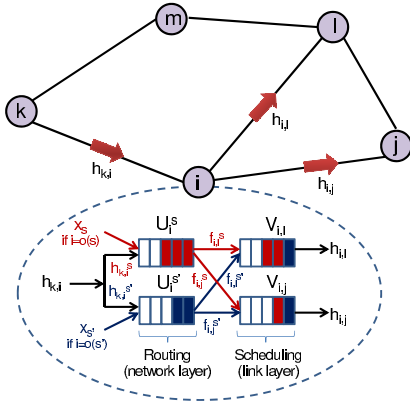
---

[1]The rationale behind the name of our scheme, *i.e.,* Diff-Max is as follows. *Diff* means that the routing part is based on queue *diff*erences, and *Max* refers to the fact that the scheduling part is based on the *max*imum of the (weighted) link layer queues. Finally, the hyphen in Diff-Max is to mention the separated nature of the routing and scheduling algorithms.

Fig. 2. A wireless mesh network. The queues at the network and link layers, and the interaction among the queues, inside node i are shown here in detail. U_i^s and $U_i^{s'}$ are the network layer queues for flows s and s', and $V_{i,j}$ and $V_{i,l}$ are the per-link queues for links; $i - j$ and $i - l$. Diff-Max algorithm makes the routing decision in the network layer, and the scheduling decision in the link layer.

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 $g_s(x_s)$, 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 g_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_i^s is the queue at node $i \in \mathcal{N}$ that only stores packets from flow $s \in \mathcal{S}$. The link layer queues are 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_i^s (assuming that node i is the

---

[2]Note 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.

source node of flow s). The network layer may also receive packets from the other nodes and insert them in U_i^s. 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_i^s changes according to the following dynamics.

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

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 $f_{i,j}^s(t)$. In particular, $f_{i,j}^s(t)$ is the actual rate of the packets, belonging to flow s, from the network layer queue; U_i^s to the link layer queue; $V_{i,j}$ at node i. Note that the optimization of flow rate $f_{i,j}^s(t)$ 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.

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

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.,* $h_{i,j}(t) \geq \sum_{s \in \mathcal{S}} h_{i,j}^s(t)$. 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, $\mathbf{C}(t)$ is the channel state vector, where $\mathbf{C}(t) = \{C_1(t), ..., C_l(t), ..., C_L(t)\}$. $C_l(t)$ 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.

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

2) Capacity Region: Let (λ_s) is the vector of arrival rates $\forall s \in \mathcal{S}$. The network layer capacity region Λ 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]. Λ is fixed and depends only on channel statistics characterized by $\Gamma_{\mathbf{C}(t)}$.

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.,* ϕ as the average value $\phi(t)$ in our NUM formulation, if both ϕ 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, $\forall s \in \mathcal{S}$, 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.,* $h_{i,j}$.

$$\max_{\boldsymbol{x},\boldsymbol{f},\boldsymbol{h},\boldsymbol{\tau}} \sum_{s\in\mathcal{S}} g_s(x_s)$$

$$\text{s.t.} \sum_{j\in\mathcal{N}} f_{i,j}^s - \sum_{j\in\mathcal{N}} h_{j,i}^s = \begin{cases} x_s, & \text{if } i=o(s) \\ 0, & \text{otherwise} \end{cases}, \forall i\in\mathcal{N}, s\in\mathcal{S}$$

$$\sum_{s\in\mathcal{S}} f_{i,j}^s \leq h_{i,j}, \forall (i,j)\in\mathcal{L}$$

$$f_{i,j}^s = h_{i,j}^s, \forall s\in\mathcal{S}, (i,j)\in\mathcal{L}$$

$$\boldsymbol{h} \in \tilde{\Gamma}. \qquad (3)$$

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.,* $\sum_{j\in\mathcal{N}} h_{j,i}^s$ and exogenous traffic, *i.e.,* x_s should be equal to the total outgoing traffic from the network layer, *i.e.,* $\sum_{j\in\mathcal{N}} f_{i,j}^s$. 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.,* $\sum_{s\in\mathcal{S}} f_{i,j}^s$. 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, $\boldsymbol{h} = \{h_1, ..., h_l, ...h_L\}$ should be the element of the available link rates; $\tilde{\Gamma}$. Note that $\tilde{\Gamma}$ is different than $\Gamma_{\boldsymbol{C}(t)}$ in the sense that $\tilde{\Gamma}$ is characterized with the loss probability over each link; p_l, $\forall l \in \mathcal{L}$, rather than the channel state vector; $\boldsymbol{C}(t)$.

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:

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

$$(4)$$

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.[3] The Lagrange function can be re-written as;

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

$$(5)$$

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:

$$x_s = (g_s')^{-1}\left(u_{o(s)}^s\right), \qquad (6)$$

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.

$$\max_{\boldsymbol{f}} \sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}\sum_{j\in\mathcal{N}}(u_i^s f_{i,j}^s - u_j^s h_{i,j}^s) - \sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}} v_{i,j} f_{i,j}^s$$

$$\text{s.t.} \; f_{i,j}^s = h_{i,j}^s, \; \forall i\in\mathcal{N}, j\in\mathcal{N}, s\in\mathcal{S} \qquad (7)$$

The above problem is equivalent to;

$$\max_{\boldsymbol{f}} \sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}} f_{i,j}^s(u_i^s - u_j^s - v_{i,j}) \qquad (8)$$

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

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

$$\text{s.t.} \; \boldsymbol{h} \in \tilde{\Gamma}. \qquad (9)$$

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 [15]. Next, we propose Diff-Max based on the structure of the decomposed solution.

---

[3]Note that u_i^s and $v_{i,j}$ 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 and $V_{i,j}$ 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;

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

$$(10)$$

where F_i^{max} is constant larger than the maximum outgoing rate from node i. According to Eq. (10), $f_{i,j}^s(t)$ 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;

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

$$\text{s.t. } \boldsymbol{h}(t) \in \Gamma_{\boldsymbol{C}(t)}, \forall(i,j) \in \mathcal{L} \qquad (11)$$

  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; $\boldsymbol{C}(t)$, 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; $\boldsymbol{C}(t)$, while the classical backpressure uses maximum queue backlog differences dictated by the routing algorithm. As it is seen the routing
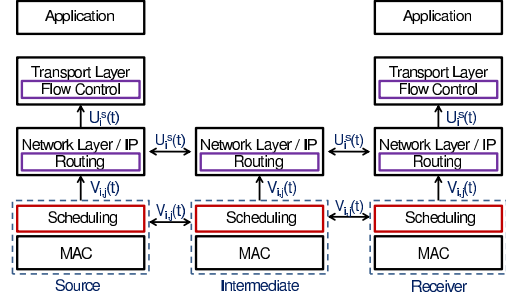

Fig. 3. Diff-Max operations at end-points and intermediate nodes.

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 transport layer to the network layer according to:

$$\max_{\boldsymbol{x}} \sum_{s\in\mathcal{S}|i=o(s)} [Mg_s(x_s(t)) - U_i^s(t)x_s(t)]$$

$$\text{s.t.} \sum_{s\in\mathcal{S}|i=o(s)} x_s(t) \leq R_i^{max} \qquad (12)$$

  where R_i^{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 [15].

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 $t_i^1, t_i^2, ..., t_i^k, ...$, where t_i^k is the beginning of the kth epoch. Let us assume that $t_i^{k+1} = t_i^k + T_i$ 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 are passed to the network layer, and inserted to the network layer queue U_i^s. 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 $\mathcal{N}_i$. 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 kth epoch at node i. Let us assume that $t_i^{'k+1} = t_i^{'k} + T_i'$ where T_i' is the epoch duration. Note that we use $t_i^{'k}$ and T_i' instead of t_i^k and T_i, because these two time epochs do not need to be the same nor synchronized.

At time $t_j^{'k}$, the routing algorithm at the network layer checks $U_i^s(t_i^{'k}) - U_j^s(t_i^{'k}) - V_{i,j}(t_i^{'k})$ for each flow s. Note that $U_j^s(t_i^{'k})$ is not the instantaneous value of U_j^s at time $t_i^{'k}$, instead it is the latest value of U_j^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 $j \in \mathcal{N}_i$ in decreasing order, according to the weight; $U_i^s(t_i^{'k}) - U_j^s(t_i^{'k}) - V_{i,j}(t_i^{'k})$. 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 backpres-

---

Algorithm 1 The routing algorithm at node i for packets from flow s at slot $t_i^{'k}$.

1: **for** $\forall j \in \mathcal{N}_i$ **do**
2:     Read the network layer queue size information of neighbors: $U_j^s(t_i^{'k})$
3:     Read the link layer queue size information: $V_{i,j}(t_i^{'k})$
4:     **if** $U_i^s(t_i^{'k}) - U_j^s(t_i^{'k}) - V_{i,j}(t_i^{'k}) > 0$ **then**
5:         $f_{i,j}(t_i^{'k}) = F_i^{max}$
6:     **else**
7:         $f_{i,j}(t_i^{'k}) = 0$
8:     Remove $f_{i,j}(t_i^{'k})$ packets from U_i^s
9:     Pass $f_{i,j}(t_i^{'k})$ packets to the link layer and insert them to $V_{i,j}$

---

sure 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 $\bar{p}_l$ and $\bar{R}_l$ are the estimated values of p_l and R_l, respectively. $\bar{p}_l$ is calculated as one minus the ratio of correctly transmitted packets over all transmitted packets in a time window over link l.[4] $\bar{R}_l$ is calculated as the average of the recent (in a window of time) link rates over link l. $V_{i,j}$, $\bar{p}_{i,j}$, and $\bar{R}_{i,j}$ 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.,* $\bar{p}_l$, and link rates, *i.e.,* $\bar{R}_l$, 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 qth maximal independent set such that $q = 1, ..., Q$, the policy vector is; $\boldsymbol{\pi}_q = \{\pi_q^1, ..., \pi_q^l, ..., \pi_q^L\}$, where $\pi_q^l = 1$ if link l is in the qth maximal set, and $\pi_q^l = 0$, otherwise. Our scheduling algorithm selects q^*th maximal independent set such that $q^* = \arg\max_{\forall q}\{\sum_{l \in \mathcal{L}} V_l(1 - \bar{p}_l)\bar{R}_l\pi_q^l\}$. Node i solves q^* as one of the parameters; $V_l, \bar{p}_l, \bar{R}_l$ 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

---
[4]Note that we do not use instantaneous channel states $C_l(t)$ in our implementation, since it is not practical to get this information. Even if one can estimate $C_l(t)$ using physical layer learning techniques, $C_l(t)$ should be estimated $\forall l \in \mathcal{L}$, which is not practical in current wireless networks.

Algorithm 2 Diff-Max scheduling algorithm at node i.

1: **if** V_l, $\bar{p}_l$, or $\bar{R}_l$ is updated such that $l \in \mathcal{L}$ **then**
2:      Determine q^* such that $q^* = \arg\max_{\forall q}\{\sum_{l \in \mathcal{L}} V_l(1 - \bar{p}_l)\bar{R}_l \pi_q^l\}$
3:      **if** $\exists(i,j)$ such that $\pi_{q^*}^{(i,j)} = 1, \forall j \in \mathcal{N}_i$ **then**
4:          Reduce 802.11 MAC contention window size and access the medium
5:          Transmit a packet from $V_{i,j}$ according to FIFO rule
6:      **else**
7:          Tell 802.11 MAC that there are no packets in the queues available for transmission

to maximum independent set problem, [9], [13]. Furthermore, it introduces significant amount of overhead; each node needs to know every other node's queue sizes and link loss rates. Due to the hardness of the problem and overhead, we implement this algorithm for small topologies over ns-2 for the purpose of comparing its performance with sub-optimal scheduling algorithms, which we describe next.

B. Diff-subMax

Diff-subMax is a low complexity and low overhead counterpart of Diff-Max. The flow control and the routing parts of Diff-subMax is exactly the same as in Diff-Max. The only different part is the scheduling algorithm, which uses 802.11 MAC protocol without any changes. When a transmission opportunity arises according to underlying 802.11 MAC at time t, then the scheduling algorithm of node i calculates weights for all outgoing links to its neighbors. Let us consider link $i-j$ at time t. The weight is $\omega_{i,j}(t) = V_{i,j}(t)(1 - \bar{p}_{i,j})\bar{R}_{i,j}$. Based on the weights, the link is chosen as; $l^* = \arg\max_{j \in \mathcal{N}_i} \omega_{i,j}(t)$. This decision means that a packet from the link layer queue V_{l^*} is chosen according to FIFO rule and transmitted. Note that this scheduling algorithm only performs intra-scheduling, *i.e.,* it determines from which link layer queue, packets should be transmitted, but it does not determine which node should transmit, which is handled by 802.11 MAC.

Diff-subMax reduces the complexity of the algorithm and overhead significantly. In particular, each node i calculates and compares weights $\omega_{i,j}(t)$ for each neighbor node. Therefore, the complexity is linear with the number of (neighbor) nodes. The overhead is also significantly reduced; each node needs to know the queue size of only its one-hop away neighbors.

C. wDiff-subMax

wDiff-subMax is an extension of Diff-subMax for the scenarios that link layer operations and data exchange (between the network and link layers) are not possible due to wifi firmware or driver restrictions or may not be preferable. Therefore, wDiff-subMax does not employ any scheduling mechanism, but the routing and flow control. The flow control algorithm is the same as in Diff-Max. Yet, the routing algorithm is updated as explained in the next.

Eq. (10) requires per-flow queues as well as per-link queues for routing decision. If per-link queues are not available at the network layer, these parameters should be estimated. wDiff-subMax, window-based routing algorithm, implements Eq. (10) by estimating per-link queue sizes. In particular, the routing algorithm sends a window of packets, and receive

acknowledgement (ACK) for each transmitted packet. The ACK mechanism has three functions: (i) carries per-flow queue size information, (ii) provides reliability, *i.e.,* packets which are not ACKed are re-transmitted, (iii) estimates per-link queue sizes. The algorithm works as follows.

At time $t_i^{'k}$, the window size for link $i-j$ is $W_{i,j}(t_i^{'k})$, the average round trip time of the packets is $RTT_{i,j}$, and the average round trip time of the packets in the last window is $RTT_{i,j}(t_i^{'k})$. If $U_i^s(t_i^{'k}) - U_j^s(t_i^{'k}) > 0$ and $RTT_{i,j}(t_i^{'k}) < RTT_{i,j}$, then $W_{i,j}(t_i^{'k})$ is increased by 1. If $U_i^s(t_i^{'k}) - U_j^s(t_i^{'k}) > 0$ and $RTT_{i,j}(t_i^{'k}) > RTT_{i,j}$, then $W_{i,j}(t_i^{'k})$ is decreased by 1. If none of the packets in the last window is ACKed, $W_{i,j}(t_i^{'k})$ is halved. After $W_{i,j}(t_i^{'k})$ is determined, $f_{i,j}(t_i^{'k})$ is set to $W_{i,j}(t_i^{'k})$ and $f_{i,j}(t_i^{'k})$ packets are passed to the link layer. wDiff-subMax, similar to Diff-subMax, reduces computational complexity and overhead significantly as compared to Diff-Max.

V. Performance Evaluation

A. Numerical Simulations

We first simulate our scheme, Diff-Max as well as classical backpressure in an idealized time slotted system in our in-house simulator. The simulation results show that Diff-Max performs as good as the classical backpressure. Next, we discuss the simulation setup and results in detail.

We consider the triangle and diamond topologies shown in Figs. 4(a) and 4(b). In the triangle topology, there are two flows between sources; S_1, S_2 and receivers; R_1, R_2, respectively. S_1 is originated from node A and ends at node B, and S_2 is originated from node A and ends at node C. In the diamond topology, there are two flows between sources; S_1, S_2 and receivers; R_1, R_2, respectively. S_1 is originated at node A and ends at node B, and S_2 is originated at node A and ends at node D. In both topologies, all nodes are capable of relaying packets to their neighbors. The simulation duration is 10000 slots, and each simulation is repeated for 10 seeds. Each slot is either on ON or OFF state according to the loss probability, which i.i.d. over slots and uniformly distributed at each slot.

Fig. 5 shows throughput vs. the loss probability for the triangle topology. The loss is only over link $A-C$. Fig. 5(a) shows the total throughput of the two flows, *i.e.,* from S_1 to R_1 and S_2 to R_2, while Fig. 5(b) and Fig. 5(c) present individual throughput of flows from S_1 to R_1 and S_2 to R_2, respectively. As it is seen, both the total throughput and individual throughput in Diff-Max scheme is equal to the ones in the classical backpressure. This observation is confirmed for different loss scenarios and for the diamond topology in Figs. 6, 7, 8.

B. ns-2 Simulations

In this section, we simulate our schemes, Diff-Max, Diff-subMax, wDiff-subMax as well as classical backpressure in the ns-2 simulator [10]. The simulation results show that Diff-Max, Diff-subMax and wDiff-subMax significantly improves throughput as compared to the adaptive routing scheme; Ad
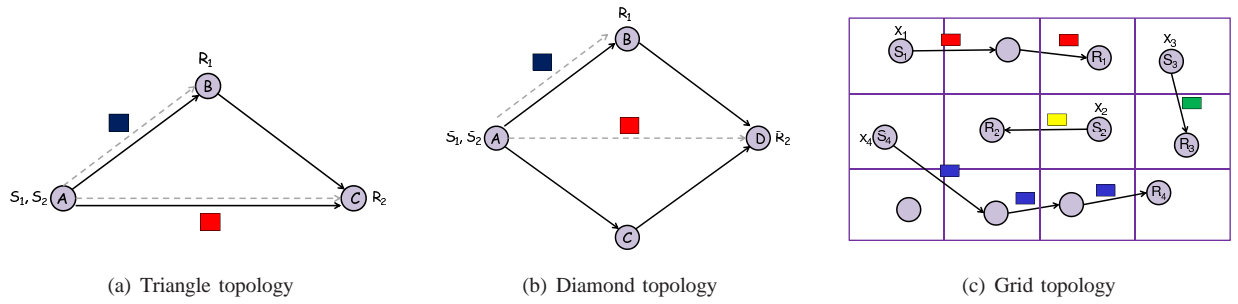
(a) Triangle topology

(b) Diamond topology

(c) Grid topology

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 4×3 grid. An example node distribution and possible flows are illustrated in the figure.

(a) Total throughput vs. loss probability

(b) Throughput of flow from S_1 to R_1 vs. loss probability

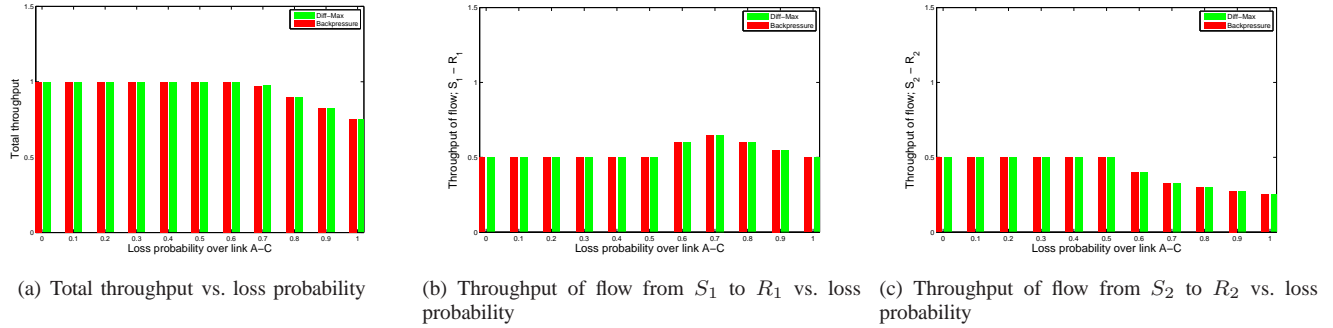(c) Throughput of flow from S_2 to R_2 vs. loss probability

Fig. 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.

(a) Total throughput vs. loss probability

(b) Throughput of flow from S_1 to R_1 vs. loss probability

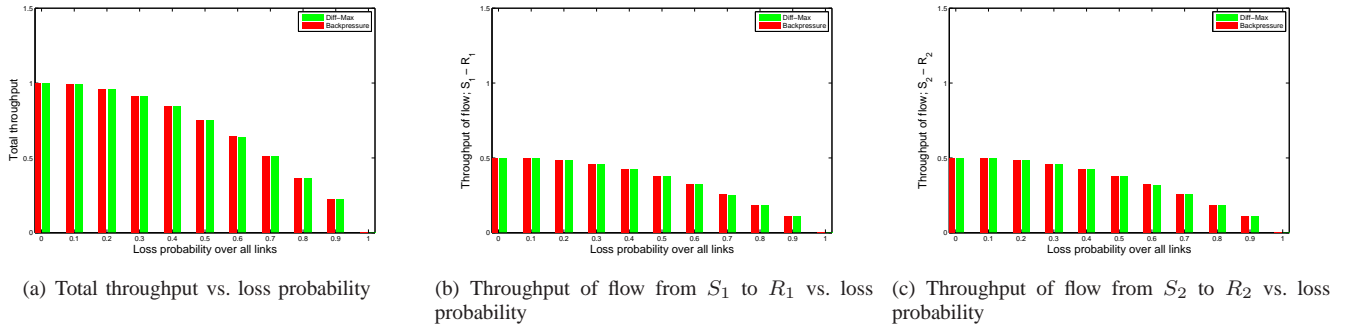(c) Throughput of flow from S_2 to R_2 vs. loss probability

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.

(a) Total throughput vs. loss probability

(b) Throughput of flow from S_1 to R_1 vs. loss probability

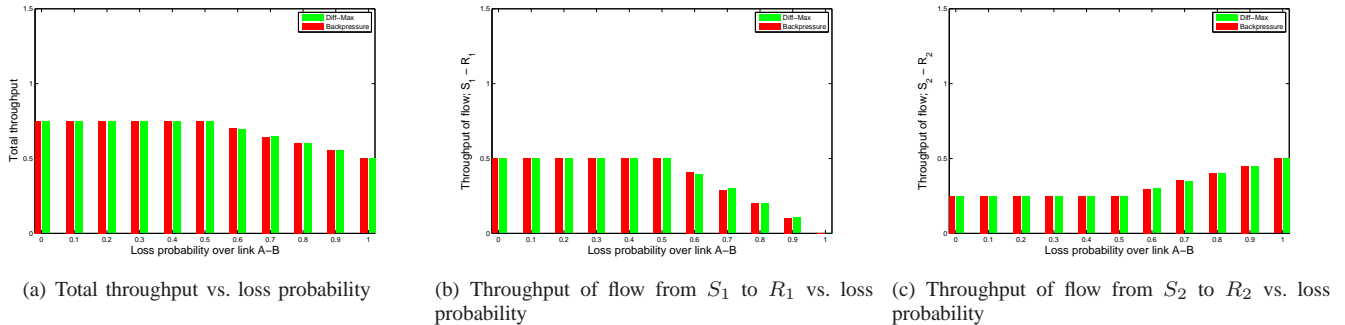(c) Throughput of flow from S_2 to R_2 vs. loss probability

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.

hoc On-Demand Distance Vector (AODV) [11]. Next, we present the simulator setup and results in detail.

1) Setup: We considered two topologies: diamond topology shown in Fig. 4(b); and a grid topology shown in Fig. 4(c). In
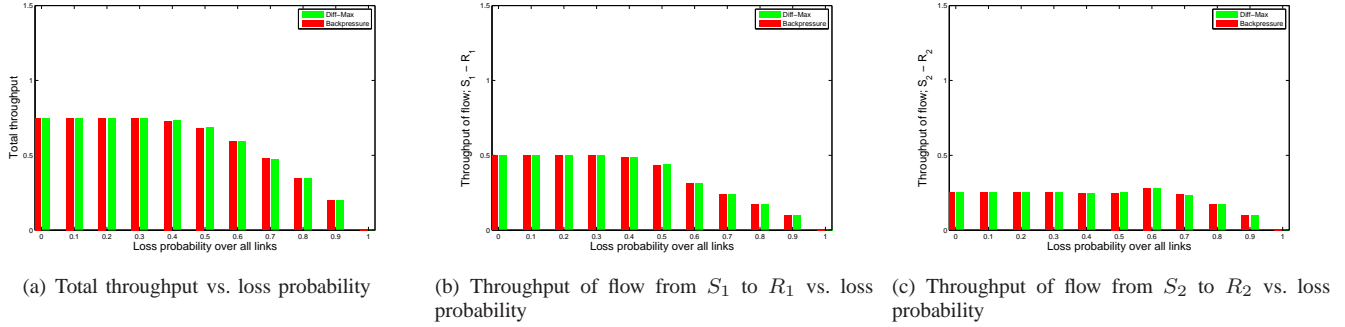
(a) Total throughput vs. loss probability

(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. 8. Diamond topology shown in Fig. 4(b). 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.

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, 4×3 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 $5sec$ and are on until the end of the simulation which is $100sec$. The CBR flows generate packets with inter-arrival times $0.01ms$. 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 $100sec$ simulation for 10 seeds.

The channel capacity is $1Mbps$, 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; $T_i = 80ms$, $R_i^{max} = 20$ packets, $M = 200$. For the routing algorithm; $T_i' = 10ms$, $F_i^{max} = 4$ 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.
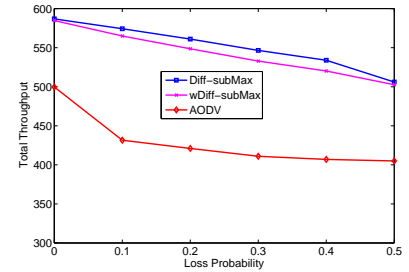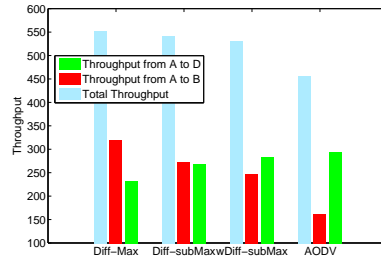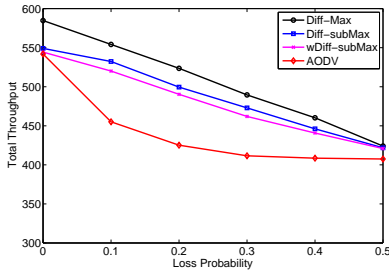
Fig. 9(a) also shows that when loss rate is 50%, the throughput improvement of all schemes are similar, because at 50% loss rate, link $A - B$ becomes very inefficient, and all of the schemes transmit packets mostly from flow A to D over path $A - C - D$ and have similar performance at high loss rates.

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 $16sec$ which is comparable with its TCP counterpart, which

---

[5]We consider the loss rates in the range up to 50%, because recent studies of IEEE 802.11b based wireless mesh networks [17], [18], have reported packet loss rates as high as 50%.

(a) Diamond topology. Total throughput vs. average loss rate

(b) Diamond topology. Throughput of different policies

(c) Grid topology. Total throughput vs. average loss rate

Fig. 9. Total throughput vs. average loss rate for different policies and in two different topologies. (a) Total throughput vs. average loss rate in diamond topology. (b) Total and per-flow throughput for different policies when the average loss rate is set to 10% in the diamond topology. (c) Total throughput vs. average loss rate in grid topology.

was $14sec$. 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 $A - B - D$ and configure phone B so that it drops relaying packets after $10sec$ 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 $40sec$.

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 [16]; especially for wireless and-hoc networks [19], [20], [21], [22], [23], [24]. Also, it has been shown that backpressure can be combined with flow control to provide utility-optimal operation guarantee [3], [23]. 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], [25], [26] 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, [27], [28] 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

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *in IEEE Trans. on Automatic Control*, vol. 37(12), Dec. 1992.

[2] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *in IEEE Trans. on Information Theory*, vol. 39(2), March 1993.

[3] M. J. Neely, E. Modiano, and C. Li, "Fairness and optimal stochastic control for heterogeneous networks," *in IEEE/ACM Trans. on Networking*, vol. 16(2), April 2008.

[4] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: practical differential backlog congestion control for wireless networks," *in Proc. of Infocom*, Rio de Janerio, Brazil, April 2009.

[5] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar, "Joint scheduling and congestion control in mobile ad-hoc networks," *in Proc. of Infocom*, Phoenix, AZ, April 2008.

[6] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key, "Horizon: balancing TCP over multiple paths in wireless mesh network," *in Proc. of ACM MobiCom*, San Francisco, CA, Sep. 2008.

[7] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: the backpressure collection protocol," *in Proc. of ACM IPSN*, Stockholm, Sweden, April 2010.

[8] R. Laufer, T. Salonidis, H. Lundgren, and P. L. Guyadec, "XPRESS: A cross-layer backpressure architecture for wireless multi-hop networks," *in Proc. of ACM MobiCom*, Las Vegas, NV, Sep. 2011.

[9] M. Chiang, S. T. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: a mathematical theory of network architectures," *in Proceedings of the IEEE*, vol. 95(1), Jan. 2007.

[10] The Network Simulator - ns-2, Version 2.35, *available at www.isi.edu/nsnam/ns/*.

[11] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," *RFC 3561, IETF*, July 2003.

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

[13] X. Lin, N. B. Schroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *in IEEE Journal on Selected Areas in Communication*, vol. 24(8), Aug. 2006.

[14] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," *in Proc. of Decision and Control*, vol. 2, Dec. 2004.

[15] H. Seferoglu, E. Modiano, "Diff-Max: separation of routing and scheduling in backpressure-based wireless networks," Tech. Report, available at *newport.eecs.uci.edu/~hseferog/*.

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

[17] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *in Proc. of ACM SIGCOMM*, Portland, OR, Sept. 2004.

[18] C. Steger, P. Radosavljevic, and J. P. Frantz, "Performance of IEEE 802.11b wireless LAN in an emulated mobile channel," *in Proc. of VTC*, Orlando, FL, Oct. 2003.

[19] L. Tassiulas, "Scheduling and performance limits of networks with constantly changing topology," *in IEEE Trans. on Information Theory*, vol. 43(3), May 1997.

[20] N. Kahale and P. E. Wright, "Dynamic global packet routing in wireless networks," *in Proc. of Infocom*, Kobe, Japan, April 1997.

[21] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijaykumar, "Providing quality of service over a shared wireless link," *in IEEE Communications Magazine*, vol. 39(2), Feb. 2001.

[22] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic power allocation and routing for time varying wireless networks," *in IEEE Journal on Selected Areas in Communications*, vol. 23(1), Jan. 2005.

[23] A. L. Stolyar, "Greedy primal dual algorithm for dynamic resource allocation in complex networks," *in Queuing Systems*, vol. 54, 2006.

[24] J. Liu, A. L. Stolyar, M. Chiang, and H. V. Poor, "Queue back-pressure random access in multihop wireless networks: optimality and stability," *in IEEE Trans. on Information Theory*, vol. 55(9), Sept. 2009.

[25] A. Sridharan, S. Moeller, and B. Krishnamachari, "Implementing backpressure-based rate control in wireless networks," *in Proc. of ITA Workshop*, San Diego, CA, Sep. 2008.

[26] A. Sridharan, S. Moeller, and B. Krishnamachari, "Making distributed rate control using Lyapunov drifts a reality in wireless sensor networks," *in Proc. of WiOpt*, Berlin, Germany, April 2008.

[27] E. Athanasopoulou, L. X. Bui, T. Ji, R. Srikant, and A. Stolyar, "Backpressure-based packet-by-packet adaptive routing in communication networks," *to appear in IEEE/ACM Trans. on Networking*, 2012.

[28] L. X. Bui, R. Srikant, and A. Stolyar, "A novel architecture for reduction of delay and queueing structure complexity in the back-pressure algorithm," *in IEEE/ACM Transactions on Networking*, vol. 19(6), Dec. 2011.