DiffQ: Practical Differential Backlog Congestion Control for Wireless Networks

4 authors, including:

Sangtae Ha
Institute of Electrical and Electronics Engineers
98 PUBLICATIONS   **4,745** CITATIONS

SEE PROFILE

Injong Rhee
North Carolina State University
82 PUBLICATIONS   **9,438** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Elastic OS View project

DiffQ: Practical Differential Backlog Congestion Control for Wireless Networks

Ajit Warrier, Sankararaman Janakiraman, Sangtae Ha and Injong Rhee

Dept of Computer Science

North Carolina State University

{acwarrie,sjanaki,sha2,rhee}@ncsu.edu

Abstract—Congestion control in wireless multi-hop networks is challenging and complicated because of two reasons. First, interference is ubiquitous and causes loss in the shared medium. Second, wireless multihop networks are characterized by the use of diverse and dynamically changing routing paths. Traditional end point based congestion control protocols are ineffective in such a setting resulting in unfairness and starvation. This paper adapts the optimal theoretical work of Tassiulas and Ephremedes [33] on cross-layer optimization of wireless networks involving congestion control, routing and scheduling, for practical solutions to congestion control in multi-hop wireless networks. This work is the first that implements in real off-shelf radios, a differential backlog based MAC scheduling and router-assisted backpressure congestion control for multi-hop wireless networks. Our adaptation, called DiffQ, is implemented between transport and IP and supports legacy TCP and UDP applications. In a network of 46 IEEE 802.11 wireless nodes, we demonstrate that DiffQ far outperforms many previously proposed "practical" solutions for congestion control.

I. INTRODUCTION

Wireless networks are increasingly becoming popular because of the ease of access that they enable. However they are affected by several unique problems. Interference is one of the primary such problems causing loss in the medium. The problem is aggravated with the wireless medium being a shared medium over the air. The common form of media access control (MAC) is CSMA such as IEEE 802.11 where a radio transmission can affect a geographically scoped region instead of a specific receiver. In such a network, contention occurs not only among those flows that share the same links or routers, but also among neighboring flows that do not necessarily share the links. Directly applying TCP to these networks causes severe unfairness in resource usage among competing flows and this problem has been well-documented in a number of papers (see [11]).

This problem is unique in wireless multi-hop networks because in wired networks, when a flow in a congested link receives congestion indications, all the other flows competing in the same link receive a similar amount of congestion indications. In wireless networks, although several flows share the same resource, it is possible that only a subset of those flows get congestion indications depending on their topology. This unfairness problem does not disappear by replacing the congestion indications from losses to delays, ECNs or explicit rate estimation at the link as in ATP [3], and ECN [7].

It requires coordinated rate control among competing nodes within the *same interference range*.

Tassiulas and Ephremedes [33] provide a classical theoretical solution, called *differential backlog based backpresssure* (in short, *differential backlog*), to congestion control for multi-hop wireless networks; the work was originally proposed for a multi-commodity flow problem, but in many follow-up studies [24], [25], [28], was adapted for the cross-layer optimization of multi-hop wireless networks involving congestion control, routing and MAC scheduling. However, although the solution is optimal, it has never been adopted by practitioners because of its complexity and unrealistic convenient assumptions on scheduling and routing. Instead, practitioners have adopted solutions often arising from intuitions, but without much theoretical underpinning, for the same problem (e.g., [3], [7], [32], [30], [36], [23], [4], [13]). There have never been any performance studies comparing the optimal differential backlog scheme to the existing practical solutions.

In this paper, we provide the first practical adaptation and implementation of differential backlog that involves a cross layer optimization of both congestion control and MAC scheduling in real multi-hop wireless. We call our adaptation *DiffQ*. One of the main goals of this work is to measure the performance difference between differential backlog and the other existing practical, but ad hoc, solutions. Since we adapt the original solution for practical implementation using several heuristics for scheduling and routing, our implementation does not guarantee the optimal performance. However, our experimental work shows that DiffQ far outperforms many contemporary practical solutions including WCP [30], ATP [3], TCP-FeW [23], TFRC [8], TFRC with ECN [7] and TCP-SACK[22].

DiffQ is implemented in a layer between transport and IP inside the Linux kernel and modifies IEEE 802.11 MadWiFi driver for efficient heuristic MAC scheduling based on differential backlog and supports source rate control as well as router-assisted backpressure rate control. Because DiffQ is implemented below transport, it supports any legacy applications like Web, Email and FTP using TCP or UDP.

Although DiffQ is an approximation of the optimal solution, our interest lies in demonstrating and quantifying the performance advantage of a theoretically motivated solution, such as DiffQ, over existing practical solutions, and encouraging more practical follow-up work on closing the performance gap

between the optimal and practical solutions. We hope to inform the research community about the practical performance of differential backlog based congestion control and show how its predicted gains can be realized successfully in practice using off the shelf radios.

The rest of the paper is organized as follows. Section II motivates for our work, Section III presents related work, Sections IV and V describe the design of DiffQ and implementation details of DiffQ into Linux respectively, and Section VI discusses the experimental results.

II. MOTIVATIONS

This section examines the key congestion-related performance problems observed in wireless multi-hop networks.

We first describe the Flow in the Middle problem. Then, we show the severity of unfairness and starvation that occurs when using TCP like congestion control algorithms in wireless multihop networks. We trace the cause of this starvation to the *Flow in the Middle* (FIM) problem.

A. Flow in the Middle Problem

Figure 1 describes the flow in the middle problem. In this test, we run three flows as illustrated in Figure 1 (a) where one multi-hop flow (flow 1) starts first and the other two one-hop flows join later. Figure 1 (b) shows the instantaneous throughput of each flow over time. As soon as the third flow joins, the first flow quickly starves off. The following explains the reason. While the second and third flows get interference from only the first flow, the first flow does so from the other two flows as well as its own flow being forwarded in different nodes. As a result, the channel accesses of node B interfere with those of four other nodes, A, C, E and G. Thus, the first flow gets less channel accesses especially at node B than the other two flows. This causes congestion at node B as it is unable to flush the queues. In the mean time, flows 2 and 3 do not get any losses because their destinations are not under any interference. This enables the TCP senders of flows 2 and 3 to increase their rates according to their AIMD algorithm while the TCP sender of flow 1 reduces its rate drastically. This gives the other two flows more chances for channel access and their TCP senders further increase their rates as they interpret the increased channel accesses as increase in available bandwidth. The situation escalates to the eventual starvation of the first flow.

B. Severity of Unfairness and Starvation

We conduct multiple experiments with 4, 8, 16 and 32 concurrent flows in each experiment lasting for 1 minute and record the throughput of each flow. The source and destination of each flow is selected randomly among the 46 nodes on the testbed.

Figure 4 illustrates the unfairness problem of TCP in a particular run of the experiments. While a few flows get a disproportionately large amount of bandwidth, many flows are starving. Further all the TCP variants we evaluated show a high degree of unfairness.
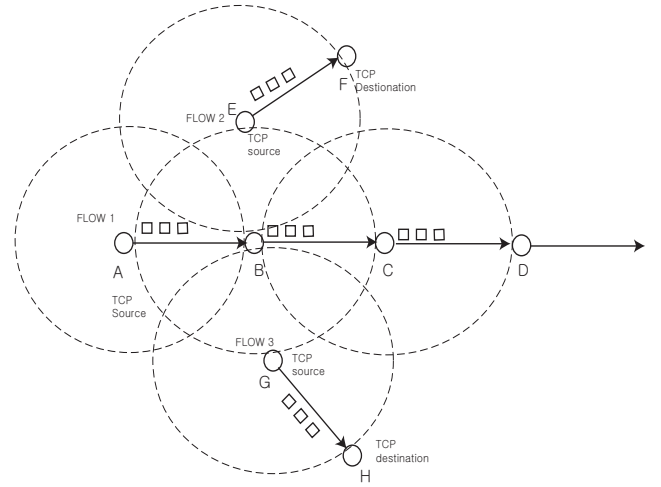
(a) A test scenario; dotted circles represent interference ranges.
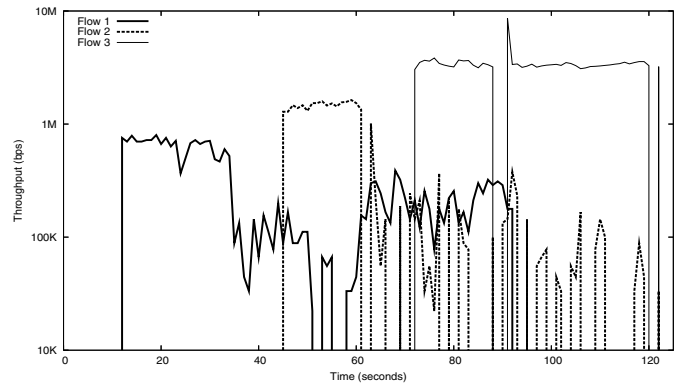
(b) Instantaneous throughput of three TCP flows

Fig. 1. The fairness problem of TCP in multi-hop wireless networks. We test three flows – one flow over three hops started at time 0 and the two other flows over one-hop started at seconds 45 and 70 respectively. We find that the throughput of flow 1 quickly reduces to zero as flows 2 and 3 join due to the fairness problem.
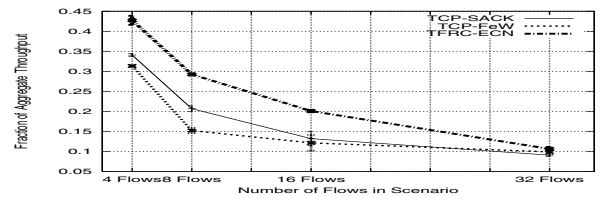

Fig. 2. Fraction of aggregate throughput obtained by the bottom 75% of the flows in each experiment.

We repeat each experiment for 20 runs with a different configuration of source and destination nodes and report the average and 95% confidence intervals for performance metrics.

Figure 2 shows the average fraction of the aggregate throughput in an experiment taken up by the bottom 75% of the flows. Even with only 4 flows in the network, 75% of the flows account only for less than 35% of the total aggregate throughput for TCP-SACK and TCP-FeW. TFRC-ECN fares somewhat better by pushing the allocation up to about 43% of the total throughput. However, the allocation quickly becomes
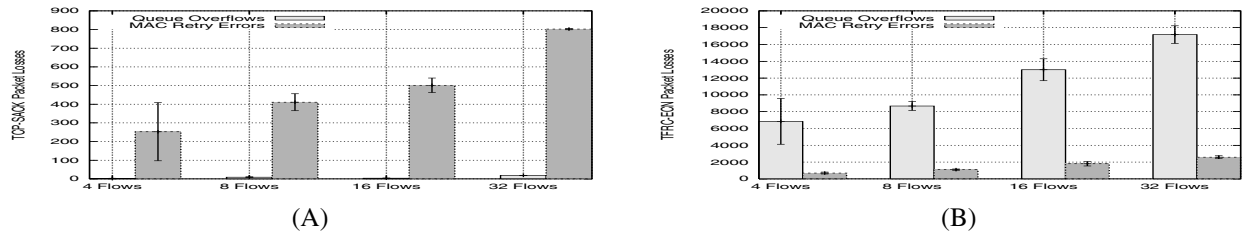
Fig. 3.   The comparison of packet losses for TCP-SACK (A) and TFRC-ECN (B).
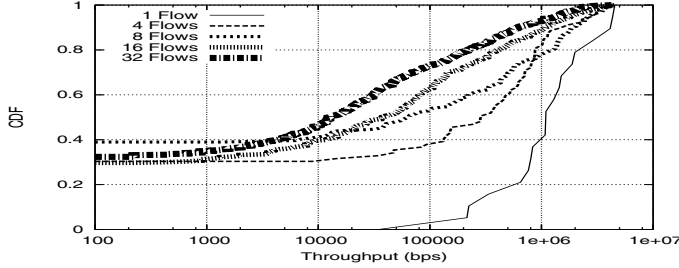

Fig. 4.   CDF of per-flow average throughput for a particular run of TCP

highly skewed with the addition of more flows, and by the time 32 flows are introduced, 75% of the flows in SACK, FeW and TFRC-ECN are allocated only 10% of the total throughput.

C. Analysis of Starvation

We trace the cause for starvation by analyzing the errors that get generated when starvation occurs. We look at two type of errors 1) Queue Overflows 2) MAC Retry errors. The flow in the middle problem results in a queue buildup eventually leading to queue overflows as the affected node is not able to flush packets fast enough. Wireless link errors and hidden terminals result in MAC retry errors.

We compare TCP-SACK and TFRC-ECN to find out the dominant cause for unfairness problem. Figures 3 (A) and (B) show the average number of per-run packet losses seen in TCP-SACK and TFRC-ECN respectively, with respect to buffer overflows and MAC errors. Note that MAC errors dominate for TCP-SACK while buffer overflows dominate for TFRC-ECN. The consequence of these MAC errors for TCP-SACK is that queue overflows are rarely seen and much before they happen, the TCP-SACK source clamps down its rate due to packet losses. In contrast, TFRC-ECN uses AQM for rate control and is independent of packet losses. TFRC-ECN causes the FIM problem to manifest itself. Buffer overflows outnumber MAC errors about 8:1 for TFRC-ECN (Figure 3 (B)), which shows that the FIM problem far overshadows other reported problems with TCP congestion control.

III. RELATED WORK

A. Congestion control for wireless multi-hop networks

Lochert et al. [21] gives an extensive overview on proposals for improving TCP performance in wireless multi-hop networks. Many involve retaining the end-point congestion control of TCP with slight modifications in router functionality

to give notifications about link failures (e.g., [12], [20]) and congestion states (e.g., [7], [34], [36], [30]) or to use other congestion metrics such as delays (e.g., [9], [3]) or explicit rate calculation (e.g., [4], [32]) measured inside routers. Some (e.g., [10], [23]) fix the problems of TCP over-estimation of path bandwidth by reducing the source rate of TCP.

These solutions ignore the fairness issues arising from radio interference. Recently, there have been several proposals to fix the fairness problems. Some of them can be applied only to mesh networks or sensor networks where traffic patterns always follow many-to-one transmissions involving one or more gateways or sink nodes (e.g., [11], [29], [13], [31]).

WCP [30] has been proposed to provide max-min fairness in wireless multihop networks. WCP relies on congestion sharing and synchronized AIMD increase/decrease among flows that are in the same transmission range. However WCP suffers from interference from nodes outside the neighborhood. We evaluate and compare DiffQ with WCP and ATP in Section VI-B.

B. Cross-layer optimization

Our solution is inspired by theoretical cross-layer optimization approaches, especially differential backlog backpressure, first proposed for wireless multi-hop networks by [33]. There have been many follow-up studies (e.g., [25], [5], [28], [19], [24]) that use the framework to jointly optimize various components of the protocol stack including scheduling, routing, congestion control, multi-receiver diversity and coding. However, these studies use interference models such as the one-hop interference model to avoid solving an NP-hard problem for the optimal solution. But the interference in real wireless networks commonly affects a multi-hop neighborhood. More important, most of them require the network to be sufficiently loaded to offer the optimization in order to measure queue size differences. In these solutions, when the network is lightly loaded, packets are sent almost everywhere randomly to "create congestion". This incurs high transport delays (see [35]). Since all these solutions optimize for maximizing a function of throughput, the high delays do not affect their optimality. But in practice, delay is an important issue for Internet applications (e.g. web, instant messaging) whose requirement is not real time, but fast response time is a key for their success. Recently, Neely and Urgaonkar [25] combine opportunistic routing, and Radunovic et al. [28] combine network coding, into the cross-layer optimization framework. They also propose some practical adaptations of their optimal solutions. Both solu-

tions require a reliable link-level signal plane that transports acknowledgment for each broadcast transmission. Providing reliable link-level acknowledgments for broadcast incurs high overhead in each packet transmission. These solutions are also not completely free from the problems cited above for differential-backlog cross layer optimization. For instance, [25] does not solve packet reordering problems due to the use of diverse paths, and both solutions, to a varying degree, still allow packets to reach many parts of the network that may not lead to destinations just to "create congestion". In general, these cross-layer optimization solutions require changes in existing application-level routing schemes. From the systems perspective, it is of more utility for a congestion control service module to support many application-level protocols without requiring their modifications.

We first applied differential backlog based congestion control in wireless sensor networks [35]. Akyol et al., [2] show using simulations that differential backlog based congestion control yields near optimal rates. Also [16], [17] theoretically analyze the problem of MAC scheduling for utility maximization and propose distributed MAC scheduling algorithms similar to DiffQ. While their work is similar in spirit, their analysis is restricted to theory and simulations, while DiffQ focuses on real world implementation.

Concurrently with our work, Horizon [27] also proposes the use of differential backlog based congestion control for TCP with multi-path routing in a practical setting. Horizon, however, uses IEEE 802.11 and does not support backlog-based MAC scheduling. Our work shows that MAC scheduling is a key component that enhances the performance of congestion control in multi-hop networks.

IV. DIFFQ DESIGN

In DiffQ, each node maintains a queue for each destination of the flows whose packets it forwards. At the reception of a packet, it can be delivered to the application if the node is the destination or placed into its destination queue in the FIFO order for forwarding to next hops. The queue information and states in a node are soft-state – a destination queue may disappear when there is no packet for that destination.

Each node keeps track of the sizes of destination queues located at its neighboring nodes. When sending a packet, a node piggybacks the queue size of its destination queue where that packet has been queued. This information is overheard by its neighbors. If the next hop is the destination of a packet, then the size of the corresponding queue in that next hop is set to zero. The queue size information is used in a unique way to apply backpressure-based congestion control.

We assume that routing is determined by the underlying routing layer and each node knows its next hop forwarding nodes for destinations of received packets. The per-hop congestion control algorithm of DiffQ works as follows. Let us denote $Q_i(d)$ to be the destination queue for destination d in node i. For each destination queue $Q_i(d)$, it maintains the following information every time a new packet is received or overheard from a neighbor destined to destination d. Suppose

that j is the next hop node toward the destination d from node i according to the underlying routing layer.

$$QD_i(d) = |Q_i(d)| - |Q_j(d)| \qquad (1)$$

where $|Q_i|$ is the size of queue Q_i. We call $QD_i(d)$ the *queue differential* or *differential backlog* of destination d at node i. For the head-of-line (HOL) packet of each destination queue, we assign a priority which is used for resolving MAC contention in IEEE 802.11 when it is transmitted. At each time that a new packet needs to be transmitted, node i evaluates the priority of the HOL packet of each queue based on its queue differential – the larger the queue differential, the higher priority the packet gets. Since we can only support a finite number of priority levels, we quantize the queue differential value. For simplicity, we use a linear quantization by dividing the queue differential by a fixed interval that is set by dividing the maximum queue size by the number of supported priority levels. Node i chooses the HOL packet of the highest priority among all HOL packets in its destination queues, for transmission next time. Ties are broken arbitrarily. The priority of the HOL packet is used to resolve the channel access. We modified the IEEE 802.11 MadWiFi driver to support prioritized access among competing nodes by adjusting the contention window sizes and AIFS. More details are given in Section V. It ensures higher chances for channel access to the node with a higher priority packet for transmission. The pseudo code for the source and forwarder are given in *Source Rate Control()* and *Forwarder Algorithm()* respectively.

A source node regulates its flow rate based on its own queue size. Its queue size increases or reduces based on the function of backpressure coming from the network based on our DiffQ scheduling. The rate control must reduce its rate when the queue increases and increase its rate when the queue decreases. As the particular choice of rate control has significant effect on the utility of the system in terms of throughput, it must be carefully designed. In our study, we evaluate AIMD or logarithmic adjustment (which optimizes for the sum of log of per-flow throughput). Our pseudo-code of DiffQ below describes a version of AIMD.

Algorithm *Source Rate Control*()
1.   F = Destination of flow originating at this node
2.   qlen $\leftarrow |Q_i(F)|$;
3.   **if** qlen $>$ QUEUE_THRESH
4.   rate = rate/β;
5.   **else**
6.   rate = rate + α;

Algorithm *Forwarder Algorithm*()
1. $\Delta \leftarrow$ Number of priority levels supported by MAC;
2. $D \leftarrow$ Maximum per-destination queue size;
3.
4.   **Flow Scheduling**
5.   F $\leftarrow \text{argmax}_d QD_i(d)$;
6.   P $\leftarrow$ HOL packet of $Q_i(F)$;
7.   P.priority $\leftarrow$ MAX($\lceil \frac{QD_i(F)}{D}\Delta \rceil$, 0); P.qlen $\leftarrow |Q_i(F)|$;
8.   Transmit P;

9.
10. **On receiving packet P from local application**
11. Encapsulate P with DiffQ header;
12. **if** P is the first packet
13.   Create flow entry for P's destination;
14. F ←Destination of P;
15. Enqueue P into $Q_i(F)$;
16.
17. **On reception of packet P from node j**
18. F ←Destination of P;
19. **if** F is this node
20.   Decapsulate DiffQ Header;
21.   Send it up to the application;
22.   **else**
23.   **if** No flow entry exists for F
24.   Create flow entry for F;
25.   **if** node j is the routing next-hop for F
26. $QD_i(F) \leftarrow |Q_i(F)| - |Q_j(F)|$;
27.   **else**
28.   Enqueue P into $Q_i(F)$;

We now provide the rationale for the above algorithm. DiffQ looks a lot like a scheduling algorithm. But it also has a unique way of applying backpressure. Suppose that a flow f is forwarded through a chain of nodes X,Y,Z and so on in that order, and suppose the size of the destination queue of flow f at Z is reducing as somehow Z can forward the packets of f fast to its next hop. Then it will cause the queue differential at Y to increase. This has effect of increasing the forwarding rate at Y because the channel access priority increases with the queue differential. As Y gets more prioritized access, X will be waiting and its queue builds up. After Y's queue gets depleted, then again X's priority increases because its queue differential against Y's queue is rising, and then X will have a higher chance to the channel next. For the opposite case, suppose that Z's next hop is congested so Z cannot forward its packets. Then Z's queue will build up while increasing its priority. In the mean time, Y's queue differential will reduce because Z's queue is increasing and allows less chance accesses for Y. Consequently, Y's queue builds up. This backpressure will propagate to the source if Z's congestion does not resolve soon enough.

Through backpressure, the size of a destination queue in a node reflects the aggregated condition of all paths from that node to the destination: small queues represent good path conditions on the paths as packets can leave the queue fast and indicate that the paths may be able to support additional load. Thus, when the destination queue size of the next hop node gets smaller, the priority of the flow being forwarded to that node gets increased, ultimately increasing the flow rate to that node. On the other hand, large queues represent bad path conditions from that node to the destination. This reduces the queue differential of the preceding hop node which reduces its transmission. This condition propagates until the backpressure reaches the source unless the situation improves soon.

The differential backlog scheme improves fairness among neighboring flows although those flows do not share the

same routers or links. When a flow is continually denied of transmission at a node in a network path due to contention from its neighbors, then its queue size at that node will increase. This has an effect of increasing the queue differential for that flow at that node, and thus its channel access priority. Consider the scenario in Figure 1. As the one-hop flows send at a high rate, the first flow continues to be denied of channel access. In DiffQ, the destination queue of node B will increase and eventually, get a higher priority than the nodes with single-hop flows (E and G). Thus, B will be allowed to forward the packets of flow 1. DiffQ ensures the nodes with more congestion to have a higher channel access priority so that they can relieve the congestion faster before the backpressure reaches the source. This feature allows DiffQ to enforce fairness among competing flows in wireless multi-hop networks.

V. IMPLEMENTATION DETAILS

A. Overall architecture

DiffQ has been implemented as a kernel module for the 2.6 series of the Linux kernel (2.6.18 onwards). DiffQ performs queuing and scheduling on every packet being transmitted or received in the Linux networking stack. DiffQ is primarily implemented on top of IP except for prioritized channel access (link layer) and source rate control (transport layer).

B. DiffQ scheduler

The DiffQ scheduler schedules the queued packets for transmission. The scheduler works as follows: Given that some packets can be transmitted, the scheduler looks into its destination queues and schedules the destination with the highest queue differential. The HOL packet for that destination is then dequeued and re-injected back into the IP stack. Once re-injected, the packet traverses the remaining part of the IP stack to the MAC layer and finally over the air. The MAC layer priority of the packet is loaded into the TOS field of IP header to inform the MAC of the priority to use while transmitting the packet.

C. MAC prioritization extension

The packet scheduled by the DiffQ scheduler needs to be transmitted over the air by the MAC with a priority (from the IP TOS field) commensurate with its queue differential; higher the queue differential, higher the priority. We use a linear quantization to allocate packets to one of the priority levels based on the queue differential values as discussed in Section IV. DiffQ uses a modified MADWiFi-NG driver where 8 MAC priority levels have been implemented. This is an extension of the IEEE 802.11e scheme with 4 priority levels. The driver has code that checks the TOS field and puts the packet in the appropriate hardware queue. Most legacy applications do not use IP TOS and hence TOS field is 0 by default. The parameters for various queues (CWmin, CWmax, AIFS) are modified (with values shown in Table I) to implement MAC priority levels. All non-DiffQ traffic is always sent at priority 0 whose value settings correspond to the default settings of IEEE 802.11b.

TABLE I
802.11E CONFIGURATION PARAMETERS

Parameter	0	1	2	3	4	5	6	7
AIFS	7	6	6	6	3	3	3	2
CWMin	5	5	5	5	4	3	2	1
CWMax	10	9	8	7	5	4	3	3

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

Our testbed [1] consists of 46 wireless nodes distributed over three floors in a building of about 100,000 sq ft space. Nodes are PCs with 128MB RAM and 266 MHz processor. Each node is equipped with two Atheros-based 802.11 a/b/g wireless interfaces (AR 5213/5112) connected to omnidirectional antennas. All wireless interfaces are configured to operate in the ad-hoc mode with RTS/CTS disabled, transmission power set to 19 dBm and PHY rate set to 11 Mbps. We use an implementation of the OLSR [14] routing protocol which uses ETX [6] to generate routes. We compare the following single path congestion control algorithms:
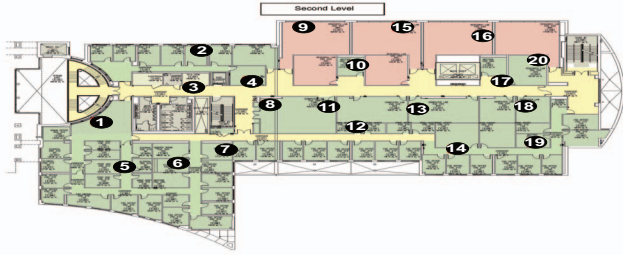

Fig. 5. One floor of our testbed with 20 nodes. The full testbed has 46 nodes over three floors.

TCP: We use TCP-SACK [22] as the baseline.
TCP-FeW: TCP-FeW [23] is a variant of TCP proposed for wireless networks. It addresses the congestion window over-estimation problem of TCP-Reno style algorithms by reducing their window growth rate. We implement the algorithm in Linux kernel (it is available only in NS-2).
TFRC-ECN: We implement TFRC-ECN where we switch off TFRC's [8] response to packet losses. Instead, intermediate nodes monitor the interface queue size and set the ECN bit in the IP header whenever the queue size exceeds 75% of the maximum queue size. Marked packets within an RTT are treated as a congestion event.
ATP: We have implemented ATP [3] on top of UDP.
WCP: We have implemented WCP [30] on top of UDP. RTT and congestion information are disseminated using overhearing techniques.
DiffQ-TCP: We implement a TCP-compatible version of DiffQ, where we retain TCP's reliability algorithm but switch off TCP's loss-based congestion control scheme. Instead, the TCP source transmits packets at the rate dictated by the DiffQ source rate control.

DiffQ-UDP: We augment UDP with the DiffQ source rate control. It does not provide any reliability and so its performance is compared to unreliable TFRC-ECN.

We construct scenarios with varying number of concurrent flows. The source and destination of each flow is chosen randomly. For a given number of flows, we construct 20 scenarios with different random configurations of sources and destinations. The packet size for all the above algorithms is set to 1400 bytes. In the experiment, we set α and β factors of the DiffQ source rate control to be 0.01 and 0.5 respectively.

The duration of each run is 1 minute. At the end of a run, the throughput of each flow is recorded. We report the mean and 95% confidence intervals of the following metrics. For N flows and average per-flow throughput r_i, $1 \le i \le N$),
Fairness index per run [15] is given by $(\sum_{i=1}^{N} r_i)^2/(N \sum_{i=1}^{N} r_i^2)$. A fairness index closer to 1 indicates almost equal bandwidth shares among N flows.
Log utility per run is given by $\sum_{i=1}^{N} log(r_i)$. It measures the degree of proportional fairness [18]. This is a utility function commonly used to measure whether a congestion control algorithm achieves high efficiency as well as fairness.
Aggregate throughput per run is $\sum_{i=1}^{N} r_i$. Although the aggregate throughput of a run may be high, the fairness index and log utility for that run may be low, indicating unfair bandwidth allocation among concurrently running flows.

B. Experimental Evaluation

We run the same setup as in Section II with various single-path algorithms. Figure 6 shows the CDF of per-flow average throughput of DiffQ-TCP. It clearly shows that DiffQ-TCP gets a far less number of starved flows than TCP (see Figure 4 for comparison). We also consider the 3-flow unfair scenario for TCP that we presented in Figure 1. Figure 7 shows the performance over DiffQ for the same scenario. The routing path for flow 1 is 20→13→12→7→6→5, flow 2 is 4→8→10 and flow 3 is 18→19, on nodes in the Figure 5. Even after all three flows have started, they co-exist and share the medium fairly without starving any one flow. This is due to the DiffQ scheduling based on differential queues. The fairness index for TCP in this scenario is 0.34 whereas the fairness index of DiffQ-TCP is 0.69 for the same scenario.
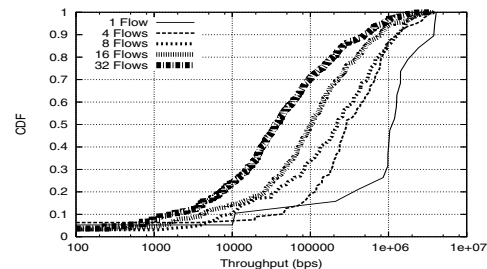

Fig. 6. The CDF of per-flow average throughput of DiffQ-TCP. It shows that DiffQ-TCP do not incur much flow starvation.

Figure 8 (b) compares the average per-run aggregate throughput of various reliable transfer algorithms under different numbers of concurrent flows. We find that the average
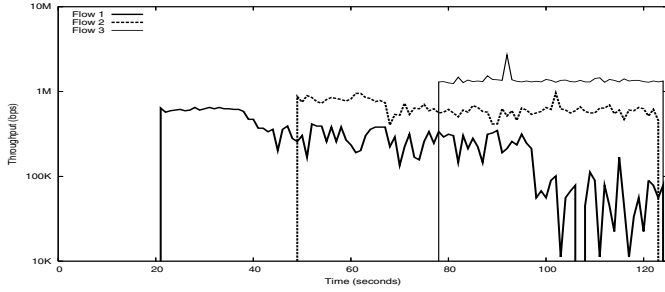
Fig. 7. The instantaneous throughput of three DiffQ-TCP flows for the same scenario described in Figure 1.

throughput of DiffQ-TCP is slightly less than TCP especially under 8 flows, but overall, it achieves comparable average performance to that of TCP and TCP-FeW.

Figures 8 (a) compares the fairness index of these protocols. The fairness of DiffQ-TCP is much higher than that of TCP. The fairness index is about two times higher for DiffQ-TCP than that of TCP. These results are manifested in their median throughput. Table II shows the ratio of average median per-flow throughput of DiffQ-TCP over that of TCP.

Finally, 8 (c) compares the log utility of these protocols. It can be seen that DiffQ-TCP performs consistently better than all the other reliable transport protocols in terms of utility.

TABLE II
RATIO OF AVERAGE PER-RUN MEDIAN THROUGHPUT OF DIFFQ-TCP
OVER TCP-SACK

Number of Flows	4	8	16	32
Median Ratio: $\frac{DiffQ-TCP}{TCP-SACK}$	1.96	3.45	4.35	8.18

Figure 10 shows a scatter plot comparing flow-by-flow the performance of TCP and DiffQ. For all runs involving 32 concurrent flows, for each source and destination pair, we plot the per-flow throughput of TCP and DiffQ-TCP on a scatter plot. Points on the straight line ($y = x$) indicate that both algorithms got the same throughput over the same path and points on top of the line indicates DiffQ-TCP achieves more throughput than TCP. We can see that for a fraction of flows starved for TCP getting no or less than 1 kbps throughput, DiffQ-TCP achieves about 10 to 100 kbps. Further, TCP has a small fraction of flows around 1 to 5 Mbps but at the cost of a significant fraction of flows around low throughput areas. On the other hand, DiffQ-TCP distributes more evenly around the middle section of plot (around 50 to 500 kbps).

We also find that the performance problem of TCP cannot be fixed by solving only the over-estimation problem of TCP window control [10]. This can be seen from the performance of TCP-FeW which shows about similar performance as TCP. This implies that the solutions for TCP performance problems require much more coordination of different techniques. On the same note, we find that the unfairness problem of TCP is not completely due to use of packet losses for congestion indications. This can be seen from the performance of TFRC-ECN in Figure 11. DiffQ-UDP achieves still much

better fairness than TFRC-ECN. In this case, DiffQ-UDP also achieves up to 20% higher average throughput. These results imply that even though we remove the effect of losses as congestion indications, TCP-like end-to-end protocols still show significant unfairness to a good fraction of flows.

We also compare DiffQ-UDP with existing ad-hoc schemes ATP and WCP. Since we have implemented ATP and WCP on top of UDP, we compare ATP and WCP along with the other unreliable congestion control protocols. The lack of reliability does not in any way affect fairness or throughput. Implementing them over UDP was a choice made out of convenience, as we did not have access to implementations of WCP or ATP.

ATP and WCP perform well in terms of fairness for a small number of flows – WCP has better fairness than DiffQ-UDP for small number of flows. However, fairness of both ATP and WCP deteriorate as more flows are added. For 16 and 32 flow cases, DiffQ-UDP has better fairness than both WCP and ATP. The vulnerability of WCP's congestion sharing scheme to interference from nodes outside transmission range contributes to this deterioration in fairness. We quantify the extent of this problem below.

We follow the *link interference ratio* based methodology as defined in [26]. We conduct multiple experiments with link pairs AB and CD where A, B, C and D are randomly selected nodes such that there does not exist a link between AC or AD or BC or BD. Using UDP traffic, we calculate the link interference ratio of AB and CD as given in [26].

Following [26], we consider a link pair to be interfering if its link interference ratio is less than 0.9. We calculate the link interference ratio in 40 different experiments using UDP and find that 17.5% of link pairs are interfering even though their endpoints are outside the transmission range of each other. Figure 9 shows the CDF of link interference ratio. Nodes on such links would not be able to disseminate congestion information to their interfering counterparts.
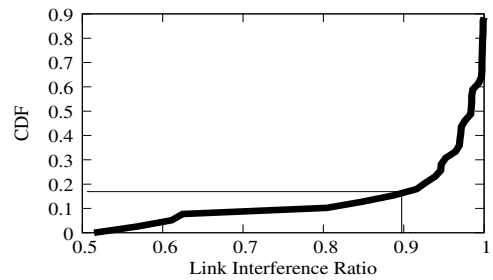

Fig. 9. The CDF of link interference ratio. 17.5% of link pairs experience interference.

Note that this problem which is due to the discrepancy between interference range and transmission range is not specific to WCP but rather is common to all ad-hoc solutions. In contrast, the fairness of DiffQ-UDP is not affected by this interference as it uses prioritized MAC scheduling. Congested nodes will transmit with high priority and these transmissions will detected by other senders in the carrier sense range due to
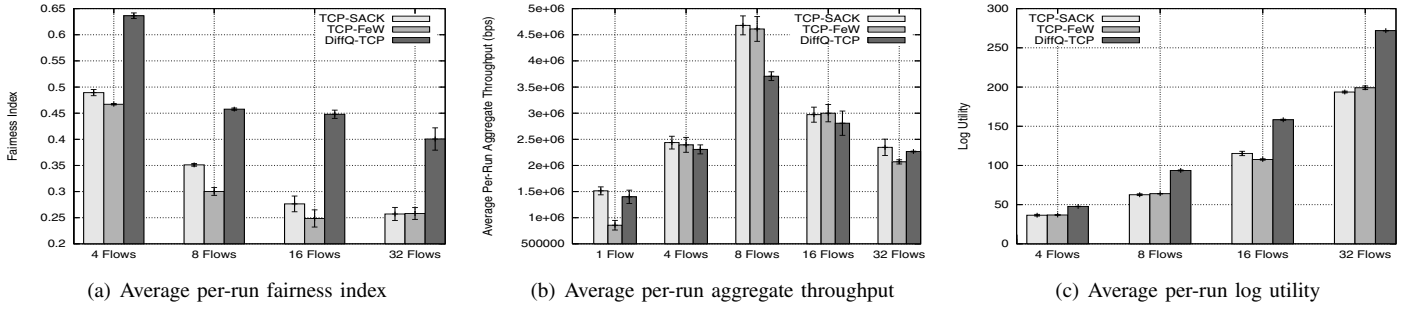
Fig. 8.   Performance metrics for reliable single-path congestion control algorithms.

CSMA and they perform a backoff allowing this transmission to succeed. DiffQ is vulnerable though, by design to *hidden terminals*. However from Figure 6, we find that DiffQ is characterized by only a very small amount of starvation even with 32 concurrent flows.

Figures 11 (B) and (C) compare aggregate throughput and log utility of the protocols. It can be seen that DiffQ performs substantially better following the same trend.

The use of differential backlog using prioritized channel access improves fairness as well as efficiency. This can be visualized from Figure 12 which shows the CDF of per-flow throughput of all the single path algorithms from a particular run involving 32 flows. The figure shows that many flows of TCP and TCP-FeW are starving up to 60% of flows (ATP and WCP experience similar, albeit lesser amounts of starvation), while TFRC-ECN can significantly reduce the number of starved flows, but more than 40% of flows are achieving less than 1 kbps throughput. Since TFRC does not react to packet losses, this indicates that much of TCP starvation is due to heavy losses causing TCP connections to timeout. We also find that these paths where TCP flows are starving can support more than 500 kbps when run without any competing flows.
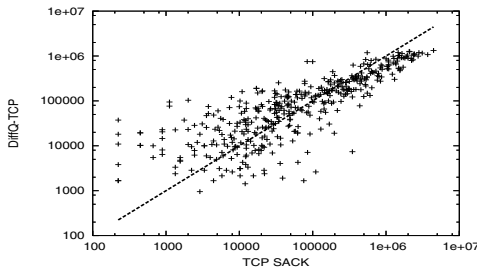

Fig. 10.   Scatter plot of per-flow throughput for the 32 flow scenarios.

C. Contribution of rate control, scheduling and MAC prioritization

How much does each DiffQ component (source rate control, differential backlog based scheduling, and MAC prioritization) contribute to the overall fairness and efficiency ? To answer this we construct two variants of DiffQ-UDP: a) *DiffQ-UDP Only RC* – we retain only source rate control and switch off the scheduling and MAC priority components of DiffQ. Flows are

scheduled based on a simple round-robin scheduling scheme, and packets are transmitted with priority 0. b) *DiffQ-UDP RC + Scheduling* – we retain both source rate control *and* scheduling, but transmit all packets at priority 0.

We run both variants along with the full DiffQ-UDP algorithm for a single experiment consisting of 16 flows. In Figure 13 we plot the number of queue overflows on intermediate nodes with respect to the path length of the flow (in hops). DiffQ-UDP incurred only about 2500 queue drops over a 2 minute experiment duration, 70% of which occurred along nodes on the 8-hop path. However, both variants of DiffQ-UDP suffer more than 70,000 queue drops over the same period. Source rate control alone is effective only for the 1-hop and 2-hop flows which incur 0 queue drops. The large number of queue drops observed for *DiffQ-UDP RC + Scheduling*, albeit somewhat lower than *DiffQ-UDP RC* indicate that over multi-hop paths, just scheduling flows with higher differential backlog is not enough to control congestion. MAC prioritization is key to flush out the packets for the flows that are scheduled by the scheduling algorithm. This is reflected in the log utilities for this experiment in the same figure.

VII. CONCLUSION

DiffQ is a flexible and scalable congestion control algorithm for wireless multi-hop networks. It does not put any restriction on the traffic patterns of flows and can be applied to general ad hoc networks. We implemented DiffQ in the Linux kernel for support of UDP and TCP.
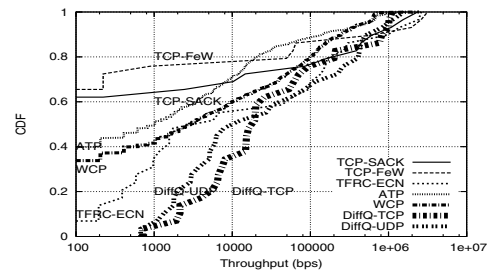

Fig. 12.   Distribution of throughput of each flow when run together.

(a) Average per-run fairness index

(b) Average per-run aggregate throughput

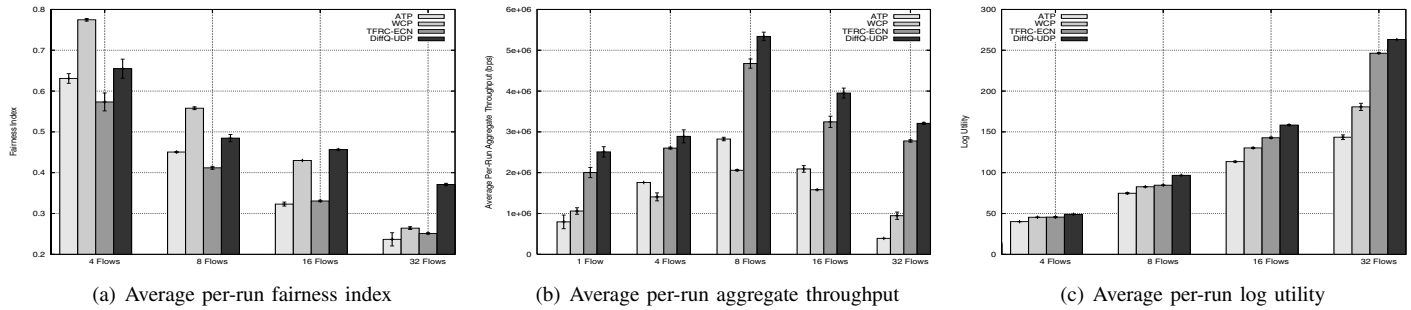(c) Average per-run log utility

Fig. 11.   Performance metrics for unreliable single-path congestion control algorithms.
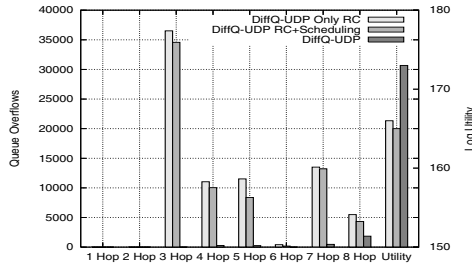

Fig. 13.   Number of queue overflows in a 2 minute experiment with variants of DiffQ-UDP.

References

[1] Wisenet Testbed: http://netsrv.csc.ncsu.edu/twiki/bin/view/Main/Facilities.

[2] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 619–627, April 2008.

[3] V. Anantharaman, K. Sundaresan, H.-Y. Hsieh, and R. Sivakumar. ATP: A reliable transport protocol for ad hoc networks. *IEEE Transactions on Mobile Computing*, 4(6):588–603, 2005.

[4] K. Chen, K. Nahrstedt, and N. Vaidya. The utility of explicit rate-based flow control in mobile ad hoc networks. In *WCNC'04*.

[5] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle. Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *INFOCOM'06*.

[6] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03*, pages 134–146, New York, NY, USA, 2003. ACM.

[7] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[8] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM'00*.

[9] Z. Fu, B. Greenstein, X. Meng, and S. Lu. Design and Implementation of a TCP Friendly Transport Protocol for Adhoc Wireless Networks. In *ICNP'02*.

[10] Z. Fu, P. Zerfos, k Xu, H. Luo, S. Lu, L. Zhang, and M. Geda. On TCP performance in multihop wireless networks. In *WING Technical Report*, 2002.

[11] V. Gambiroza, B. Sadeghi, and E. W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *MobiCom '04*.

[12] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *MobiCom '99*.

[13] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04*, pages 134–147, New York, NY, USA, 2004. ACM.

[14] P. Jacquet, P. Mhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol. In *INMIC'01*.

[15] R. Jain. The Art of Computer Systems Performance Analysis, First ed. Wiley, 1991.

[16] L. Jiang and J. Walrand. A Distributed CSMA Algorithm for Throughput and Utility Maximization in Wireless Networks. In *Allerton Conference on Communication, Control, and Computing, 2008*.

[17] A. M. J.Liu, Y.Yi and H.V.Poor. Maximizing Utility via Random Access Without Message Passing. In *Microsoft Research Technical Report*.

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

[19] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer congestion control in wireless networks. *IEEE/ACM Trans. Netw.*, 14(2):302–315, 2006.

[20] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE J–SAC*, 19(7):1300–1315, 2001.

[21] C. Lochert, B. Scheuermann, and M. Mauve. A Survey on Congestion Control for Mobile Ad Hoc Networks: Research Articles. *Wirel. Commun. Mob. Comput.*, 7(5):655–676, 2007.

[22] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options.

[23] K. Nahm, A. Helmy, and C.-C. J. Kuo. TCP over multihop 802.11 networks: issues and performance enhancement. In *MobiHoc '05*, pages 277–287, New York, NY, USA, 2005. ACM.

[24] M. J. Neely. Energy optimal control for time varying wireless networks. In *INFOCOM'05*.

[25] M. J. Neely. Optimal backpressure routing for wireless networks with multi-receiver diversity. In *CISS'06 (invited paper)*.

[26] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill. Estimation of link interference in static multi-hop wireless networks. In *IMC '05*, pages 28–28, Berkeley, CA, USA, 2005. USENIX Association.

[27] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: Balancing TCP over multiple paths in wireless mesh network. In *Mobicom' 2008*.

[28] B. Radunovic, C. Gkantsidis, P. Key, P. Rodriguez, and W. Hu. An Optimization Framework for Practical Multipath Routing in Wireless Mesh Networks. In *MSR-TR-2007-81, June 2007*.

[29] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):63–74, 2006.

[30] S. Rangwala, A. Jindal, K.-Y. Jang, K. Psounis, and R. Govindan. Understanding Congestion Control in Multi-hop Wireless Mesh Networks. In *Mobicom' 2008*.

[31] A. Raniwala, P. De, S. Sharma, S.Krishnan, and T. Chiueh. End-to-end flow fairness over ieee 802.11-based wireless mesh networks. In *INFOCOM'07*.

[32] Y. Su and T. Gross. WXCP: Explicit congestion control for wireless multi-hop networks. In *IWQoS'05*.

[33] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1949, 1992.

[34] C. Wang, B. Li, Y. Hou, K. Sohraby, and Y. Lin. LRED: a robust active queue management scheme based on packet loss rate. In *INFOCOM'04*.

[35] A. Warrier, L. Le, and I. Rhee. Cross-Layer Optimization Made Practical. In *Broadnets'07 (Invited Paper)*.

[36] K. Xu, M. Gerla, L. Qi, and Y. Shu. Enhancing TCP fairness in ad hoc wireless networks using neighborhood RED. In *MobiCom '03*, pages 16–28, New York, NY, USA, 2003. ACM.