ABSTRACT

WARRIER, AJIT C. DiffQ - Differential Backlog Congestion Control for Wireless Multi-hop Networks. (Under the direction of Associate Professor Injong Rhee).

There has been a recent spur of interest in the use of wireless multi-hop networks especially in the context of a replacement for traditional wired networks in small to medium-sized enterprises. However, as such networks increase in size, data rates, and number of users, it was quickly realized that contemporary congestion control protocols designed for wired networks, especially TCP, do not translate well into the wireless medium.

With TCP being so ubiquitous, it was a natural first step to take a top-down approach and identify the problems that plague TCP in such wireless environments. Several key problem areas have been identified; to name a few – TCP’s reliance on losses as a congestion signal, TCP’s interaction with wireless routing protocols, TCP’s aggressive probing for bandwidth and self-interference between TCP data and ACK packets. As a response to these problems, the research community has proposed several enhancements to TCP, e.g. TCP-FeW [72], TCP-Veno [31], TCP-Westwood [18], TCP-ELFN [45]. Such protocols focus on improving the *achievable capacity* in such networks. However, as the number of such concurrent TCP flows increase, the focus has now shifted to improving the *fairness* in such networks. The wireless medium is shared among neighboring nodes; thus bandwidth must be allocated fairly among neighboring flows that do not necessarily share the same link. Protocols like TCP face severe unfairness or even starvation of a large number of flows [36].

A second challenge is the recent interest in exploiting the broadcast feature of wireless networks, leading to the proposal of many creative protocols including opportunistic routing and network coding. These protocols enable the use of many diverse, yet dynamically changing routing paths. Congestion control for these protocols using traditional end-to-end protocols such as TCP may result in too conservative rate control.

Inspired by existing theoretical solutions of cross-layer optimization, we propose DiffQ, a protocol for congestion control in wireless multi-hop networks. DiffQ can support congestion control for either single-path or opportunistic multi-path routing. The protocol is currently implemented in Linux 2.6.x and tested in a network of 46 IEEE 802.11 wireless nodes. It is observed that DiffQ greatly improves the efficiency and fairness of existing transport protocols that use application-level multi-path routing and single-path routing.

DiffQ - Differential Backlog Congestion Control for Wireless Multi-hop Networks

by
Ajit C. Warriar

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Rudra Dutta

Dr. Do Young Eun

Dr. Injong Rhee
Chair of Advisory Committee

Dr. Khaled Harfoush

DEDICATION

Dedicated to my family, for their love and unwavering faith and support.

BIOGRAPHY

Ajit Chakrapani Warriier was born on July 7, 1980 in Jabalpur, India. He obtained his Bachelor of Engineering degree in Computer Engineering from Nirma Institute of Technology, Gujarat University, India in June 2002 and his Master of Science degree in Computer Science at North Carolina State University, Raleigh, NC in August 2004, where he is currently working towards a PhD in Computer Science.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Injong Rhee for being a constant source of inspiration and insight. His valuable ideas have played an important and indispensable role in the making of this thesis.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES	viii
1 Introduction	1
1.1 Thesis Organization	4
2 Prior Work	6
2.1 TCP-variants	6
2.2 Wireless Sensor and Mesh Networks	6
2.3 Network Coding	7
2.4 Multi-path Routing - Spatial Path Diversity	8
2.5 Multi-path Routing - Temporal Path Diversity	9
2.6 Cross-Layer Optimization	10
3 Motivations	12
4 DiffQ - Design	17
4.1 Congestion control for single path routing	18
4.2 Congestion control for multi-path routing	21
4.3 Prioritized MAC	23
5 QoS MAC	24
5.1 Introduction	25
5.2 Related Work	27
5.3 Medium Access Mechanisms for Fairness and QoS	29
5.3.1 Differentiated backoff-based schemes	29
5.3.2 Beacon-based schemes	29
5.3.3 Scheduling-based schemes	30
5.4 MAC Modifications for ZigBee and Multihop Environments	31
5.4.1 CC2420 Transceiver – Overview	31
5.4.2 Large Packet Size Emulation	32
5.4.3 Overhearing	32
5.4.4 802.11e	32
5.4.5 EY-NPMA	33
5.4.6 DWOP	38
5.4.7 Relative Overheads	39
5.5 Implementing Fairness Policies	40
5.5.1 Proportional Rate Allocation and Time Fairness	40
5.5.2 EDF Scheduling in Multi-hop Networks	42
5.5.3 DiffQ Congestion Control	42

5.6	Testbed Results	43
5.6.1	Experimental Methodology	43
5.6.2	EDF Scheduling	44
5.6.3	Static Priority Scheduling	46
5.6.4	DiffQ Performance	48
5.6.5	Proportional Rate Allocation	51
5.7	Conclusion	51
6	DiffQ - Architecture and Implementation	55
6.1	Overall architecture	55
6.2	Interface into IP Layer	56
6.3	DiffQ header format	57
6.4	DiffQ scheduler	58
6.5	MAC prioritization extension	59
6.6	Integration with MORE	59
7	DiffQ - Experimental Results	61
7.1	WiSeNet Testbed	61
7.2	Experimental Setup	61
7.3	Multi-path congestion control	65
7.4	Single-path congestion control	67
7.5	Contribution of rate control, scheduling and MAC prioritization	70
8	Conclusion and Future Work	77
	Bibliography	79

LIST OF TABLES

Table 6.1 802.11e configuration parameters	59
Table 7.1 Throughput of MORE and TCP	64
Table 7.2 Ratio of average per-run median throughput of DiffQ-TCP over TCP-SACK	68

LIST OF FIGURES

Figure 3.1 CDF of per-flow average throughput of MORE. Many concurrent MORE flows running without congestion control experience congestion collapse.	13
Figure 3.2 CDF of per-flow average throughput of TCP. Many concurrent TCP flows experience unfairness issues.	14
Figure 3.3 Average per-run total throughput for MORE and TCP.	15
Figure 3.4 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. ..	16
Figure 4.1 The operation of DiffQ. The thick lines represent channel accesses. According to DiffQ, nodes <i>B</i> and <i>F</i> get the channel access first as they have the largest queue differential values within their sensing ranges.	22
Figure 5.1 The working of Siren for nodes <i>A</i> , <i>B</i> , <i>C</i> and <i>D</i> with priorities 1,1,2 and 3 respectively is shown above. Both <i>A</i> and <i>B</i> send beacons after one CCA time. <i>C</i> and <i>D</i> sense the beacons after two CCA times, and hence they postpone their transmissions to the next time slot. Both <i>A</i> and <i>B</i> enter the CR phase and contend for the channel. <i>A</i> picks a shorter backoff and successfully transmits a packet while <i>B</i> senses this packet and defers its transmission to the next slot. In the third time slot, <i>C</i> transmits a beacon after two CCA times which is sensed by <i>D</i> after three CCA times. Hence <i>D</i> defers its transmission while <i>C</i> transmits its packet. Finally <i>D</i> transmits its beacon and packet in the fourth time slot.....	34
Figure 5.2 The expanded view of one time slot which clearly shows the PR and CR phases.....	35
Figure 5.3 Figure shows how node <i>A</i> uses a higher beacon transmission power to block out potential interfering nodes for receiver <i>B</i> . <i>C</i> is a one-hop node whose transmission to <i>G</i> would interfere with both <i>A</i> 's transmission and <i>B</i> 's reception. <i>F</i> and <i>E</i> are two-hop nodes whose transmission would affect <i>B</i> 's reception. <i>I</i> 's transmission to <i>D</i> would not affect <i>A</i> or <i>B</i> , however <i>D</i> 's acknowledgment to <i>I</i> would interfere with <i>B</i> 's reception. Node <i>H</i> 's transmission would not affect <i>A</i> or <i>B</i> , but it will get blocked out. This is an instance of the exposed terminal due to Siren's beacon transmission.....	36

Figure 5.4 DWOP problem scenario in lossy networks. Node <i>A</i> reserves the channel with a high priority packet using an RTS/CTS exchange. The small size of RTS/CTS allows nodes far away from <i>A</i> (e.g. <i>C</i> and <i>D</i>), to overhear <i>A</i> 's priority and enter <i>A</i> 's priority in their priority cache. The consequent data packet from <i>A</i> informs neighboring nodes that <i>A</i> 's transmission is over and also advertises the priority of the <i>next</i> HOL packet. However, due to the higher packet loss rate caused by the large data packet size, <i>A</i> 's transmission to <i>B</i> is not overheard by <i>C</i> . Hence <i>C</i> never knows that <i>A</i> has completed transmission. This causes <i>C</i> to defer its own transmissions needlessly.	38
Figure 5.5 Normalized priority resolution overhead for Siren, EY-NPMA and 802.11e with non-overlapped backoff periods with different data packet sizes.	39
Figure 5.6 Above figure shows the 35 node mote testbed distributed across one floor of the MRC building. Lines between nodes indicate connectivity achieved with data transmission power of -5 dBm.	43
Figure 5.7 A single MicaZ node. Each node is equipped with 4K RAM, 128KB flash ROM, a 2.4 GHz ZigBee compatible radio capable of a raw data rate of 250 Kbps, and an 8 MHz CPU.	44
Figure 5.8 Distribution of delays in the EDF scheduling experiment.	45
Figure 5.9 Number of runs for each MAC where the average per-flow delay is below the EDF deadline.	46
Figure 5.10 Aggregate throughput for all runs where the average per-flow delay is below the EDF deadline.	47
Figure 5.11 Above figure illustrates the DWOP problem in lossy networks. Flows $33 \rightarrow 32 \rightarrow 31$ and $7 \rightarrow 8 \rightarrow 14$ are real-time flows from one of the runs in the EDF scheduling experiment. Node 3 is the source of a flow which was observed to be starved. We monitored node 3 and found that it was continuously deferring its own transmission due to perceived high priority packets in its neighborhood. The figure on the right represents the normalized fraction of time spent in deferring attributed to the constituent flow segments. About 75% of the time 3 defers due to the RTS heard from node 32 ($32 \rightarrow 31$) or the CTS heard from 32 ($33 \rightarrow 32$).	48
Figure 5.12 Average normalized per-flow throughput (with 90% confidence intervals) of high-priority flows in the static priority scheduling experiment. Note that the MACs are presented in the increasing order of normalized throughput.	49
Figure 5.13 Average normalized per-flow throughput (with 90% confidence intervals) of low priority flows in the static priority scheduling experiment. Note that the MACs are presented in the increasing order of normalized throughput.	50

Figure 5.14 Fraction of low priority flows getting starved (0 throughput) in the static priority scheduling experiment.	51
Figure 5.15 Distribution of aggregate utility $\sum \log(x)$ for CLO experiment.....	52
Figure 5.16 Comparison of per-flow throughput of baseline CSMA with CLO-Siren for one specific run. Note that CSMA's aggregate throughput is higher, but CLO-Siren gets better utility since no flow is starved.	53
Figure 5.17 Proportional rate allocations for DWOP and Siren compared with the ideal rate allocations for 4 and 6 sources.	54
Figure 6.1 Linux Netfilter hooks; DiffQ uses these hooks to capture packets from IP and reinject processed packets into IP.	56
Figure 6.2 DiffQ Architecture and Packet Header Format - DiffQ sits on top of IP and provides congestion control services to upper layer transport modules. It also controls the MAC priority of packets for scheduling and performs source rate controls for the transport flows (for support of TCP, it disables TCP's congestion control..	58
Figure 6.3 DiffQ integration with MORE - DiffQ provides congestion control service to MORE which sits in the application layer.....	60
Figure 7.1 One floor of our testbed with 20 nodes. The full testbed has 46 nodes over three floors.	62
Figure 7.2 Soekris mesh nodes used on WiSeNet.	63
Figure 7.3 CDF of per-flow average throughput for DiffQ-MORE.....	65
Figure 7.4 Average per-run fairness index of MORE and DiffQ-MORE.....	66
Figure 7.5 Average per-run aggregate throughput of MORE and DiffQ-MORE.....	67
Figure 7.6 Average per-run log utility of MORE and DiffQ-MORE.....	68
Figure 7.7 Relative throughput difference between DiffQ-MORE and MORE with respect to path ETX.	69
Figure 7.8 The CDF of per-flow average throughput of DiffQ-TCP. It shows that DiffQ-TCP do not incur much flow starvation.	70
Figure 7.9 The instantaneous throughput of three DiffQ-TCP flows for the same scenario described in Figure 3.4.	71
Figure 7.10 Average per-run aggregate throughput for single path reliable algorithms..	72

Figure 7.11 Average per-run fairness index for single path reliable algorithms.....	72
Figure 7.12 Average per-run logarithmic utility for single path reliable algorithms.....	73
Figure 7.13 Scatter plot of per-flow throughputs for the 32 flow scenarios.....	73
Figure 7.14 Average per-run jain's fairness index for single path congestion control algorithms (unreliable transport).....	74
Figure 7.15 Average per-run log utility for single path congestion control algorithms (unreliable transport).....	74
Figure 7.16 Average per-run aggregate throughput for single path congestion control algorithms (unreliable transport).....	75
Figure 7.17 Distribution of throughput of each flow when run together.....	75
Figure 7.18 Number of queue overflows in a 2 minute experiment with variants of DiffQ-UDP.....	76

Chapter 1

Introduction

Wireless networks have proliferated significantly over the past decade – recent applications of wireless communications include habitat monitoring [90], Internet access through mesh networks [5], wireless video surveillance [61], etc. These deployments have been made possible due to several innovative research developments – on the hardware side, radio transceivers have become smaller and cheaper and on the software side, several innovative algorithms have been developed which exploit the features exclusive to the wireless environment. As such networks have progressively become larger in size, the applications being deployed on them are also demanding higher data rates. Hence congestion control for large scale multi-hop wireless networks is crucial to the success of such applications.

TCP is the dominant transport protocol in the Internet today, with an end-to-end approach to congestion control. TCP sources control their rates in an Additive Increase, Multiplicative Decrease (AIMD) manner, with rates being reduced on the detection of packet losses either by duplicate ACKs or timeouts. This approach works well on Internet backbone networks which predominantly consist of highly reliable wired links. However, with the advent of wireless links – first as one-hop access networks like wireless lans (WLANs) and later as full fledged multi-hop networks e.g. wireless mesh networks – it has become apparent that the TCP algorithm has finally begun to show its age. Applications using TCP over wireless networks observed significant performance problems [41, 12, 45].

With TCP being so ubiquitous, it was a natural first step to take a top-down approach and identify the problems that plague TCP in wireless environments. Several key problem areas have been identified; to name a few – TCP’s reliance on losses as a congestion signal, TCP’s interaction with wireless routing protocols, TCP’s aggressive probing for

bandwidth and self-interference between TCP data and ACK packets. As a response to these problems, the research community has proposed several enhancements to TCP, e.g. TCP-FeW [72], TCP-Veno [31], TCP-Westwood [18], TCP-ELFN [45]. Note that such protocols focus on improving the *achievable capacity* in such networks. However, as the number of such concurrent TCP flows increase, the focus has now shifted to improving the *fairness* in such networks.

The wireless medium is shared among neighboring nodes. 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 [36]).

The second major issue is the recent advances in wireless error control techniques which have completely changed the status quo and created environments which require a re-think of the way we approach the problem of wireless congestion control. The major challenge comes from the liberal use of multi-paths by schemes such as opportunistic routing [17] and network coding [53], which leverage the broadcast nature of wireless networks to improve network capacity. In these protocols, packets of a single flow simply “flows” through many paths toward the destination just like a river flowing through valleys. They make possibly dynamic and local changes in routing paths adapting to dynamically varying wireless link condition. Routing is also non-deterministic as it occurs opportunistically through intermediate routers that *happen to* receive packets through overhearing.

Unfortunately, there has been little research on congestion control for non-deterministic multi-path routing for wireless multi-hop networks. Most existing work in multi-path routing pertains to wired networks (e.g., [55, 96]). Applying such algorithms directly to flows taking non-deterministic routing paths is difficult – since the path traversed by a packet is non-deterministic, it is not feasible to allocate traffic to specific paths. Moreover, most congestion control enhancement proposals for wireless networks (see [67]) pertain to single path routing and rely on end-to-end congestion control after gathering feedback on path characteristics from the networks such as link estimation [15, 20, 89], ECN [29, 91] or AQM marking [96] or losses [99, 72, 94]. Applying these protocols to opportunistic multi-path routing is not effective because of a potentially large number of paths that a single flow may

take. As the number of possible paths gets large, receiving feedback from all (possibly temporary) congestion incidents from various parts of the network can cause feedback implosion commonly seen in reliable multicast [26]. Protocols using TCP as end-point congestion control must react to all these uncorrelated congestion signals from the network which limits the transmission rate to a very low rate, causing low resource utilization. Techniques to aggregate the feedback inside the network such as reporting only the maximum delays (e.g., ATP [15]) or minimum link rate (e.g., EXACT [20]) lead to too conservative rate control because the end-point rate control protocols must react only to the worst case condition in the network.

Congestion control over multi-path routing have also been studied in cross-layer optimization. Many existing cross-layer optimization frameworks are highly theoretical with unrealistic assumptions about wireless interference models and have too strong limitations to be applied directly to real networks. For instance, to achieve the optimal performance requires solving an NP-hard problem in general interference models. As a result, none of the existing solutions have been implemented in real systems.

In this thesis, we propose a new congestion control protocol for general purpose wireless multi-hop networks. This protocol is designed with the following informally defined goals: (1) it must support traffic carried by non-deterministic multi-path routing over possibly diverse and many paths as well as single path routing. This means congestion control must be scalable to the number of paths being used, (2) congestion control is a service for diverse applications so that it should not require any changes in application operations such as reliability (e.g., coding) and application-level routing (e.g., opportunistic multi-path or single path), and (3) it must improve efficiency in resource usage to achieve high throughput and fairness in resource sharing among concurrent flows.

In developing our solutions, we borrow heavily from existing theoretical work. Our solution is adapted from differential backlog based backpressure, conveniently called *differential backlog*. The technique was first applied to wireless multi-hop networks by Tassiulas and Ephremides [58] and later used in several follow-up studies [75, 76, 77, 80]. We call our adapted version of differential backlog as *DiffQ*. DiffQ implements router-assisted rate control, source rate control and MAC scheduling.

1.1 Thesis Organization

Congestion control for multi-hop networks has been studied extensively over the past decade. In Chapter 2, we present a survey of this work and try to put our work in the context of recent developments. In Chapter 3, we present the motivation for our work. Chapter 4 presents the DiffQ algorithm. DiffQ requires a MAC which supports prioritized packet transmissions. The most popular implementation of a prioritized MAC is the IEEE 802.11e [13] protocol which is available on most 802.11 [25] wireless cards. However, as we demonstrate in Section 5.6, 802.11e and its variants have numerous performance issues. As a result, we first conduct a performance evaluation of various MACs which support prioritized transmissions in Chapter 5.

In Chapter 6, we present the system architecture and implementation of DiffQ in an 802.11b-based mesh network. DiffQ is integrated into the Linux kernel and supports various transport protocols including UDP, TCP (without its congestion control), as well as application-level routing supported by MORE and ExOR.

In Chapter 7, we present the performance results of DiffQ in a testbed of 46 IEEE 802.11b nodes deployed over a 100,000 sq ft building. Our experiments show that DiffQ significantly improves the performance of MORE by 2 to 3 times when tested under 32 concurrent flows of MORE. The gain drastically improves with the increasing number of concurrent flows. This gain is possible because the congestion control mechanism of DiffQ avoids congestion collapse and ensures fair sharing of bandwidth among concurrently running MORE flows. The results demonstrate that DiffQ effectively handles multi-path routing used in opportunistic routing and network coding in wireless multi-hop networks. We also compare the performance of DiffQ augmented TCP-SACK and UDP with that of commonly used single-path congestion control algorithms such as TCP-SACK [70], TCP-FeW [72], and TFRC/ECN [30, 29]. TCP-FeW and TFRC/ECN are designed specifically for wireless multi-hop networks; TCP-FeW solves the capacity over-estimation problem of TCP [35] in wireless networks and TFRC is augmented with ECN [29] to remove TCP's dependence on using packet losses as congestion indications. TFRC/ECN uses AQM marking as congestion indications and ignores packet losses. Our experiments show that the DiffQ augmented protocols show superior fairness properties than these protocols while achieving comparable average throughput.

We finally end the thesis with discussion about the future applications of DiffQ in

Chapter 8 especially in the context of real-time video transmissions.

Chapter 2

Prior Work

Lochert et al. [67] have compiled a comprehensive survey of wireless congestion control protocols. In this section, we build on top of this survey by describing recent advances, especially opportunistic routing and network coding, and put them in context with prior work.

2.1 TCP-variants

Lochert et al. [67] 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., [45, 66, 95, 57]) and congestion states (e.g., [29, 94, 99, 33]) or to use other congestion metrics such as delays (e.g., [32, 15, 18]) or explicit rate calculation (e.g., [102, 20, 89]) measured inside routers. Some (e.g., [34, 72]) fix the problems of TCP over-estimation of path bandwidth by reducing the source rate of TCP. These solutions however ignore the fairness issues arising from radio interference.

2.2 Wireless Sensor and Mesh Networks

Sensor networks are usually deployed so that all sensors report sensor data back to one or more sink nodes. Similarly in wireless mesh networks, wireless routers route traffic towards (or down from) one or more gateways. This many-to-one flow pattern is exploited by several congestion control algorithms. Fusion [47] and CODA [93] are two such schemes

that perform *congestion mitigation*, in that they aim to detect and reduce the effects of congestion. As noted by Rangwala et al. [81] this goal is slightly different from congestion control, where the aim is to regulate rates of all sources to achieve some optimal operating point.

Both Fusion [47] and CODA [93] use queue sizes to measure the degree of congestion. CODA goes one step further and also uses the MAC utilization at each node as a congestion indicator. Fusion also rate-limits the transit and self-traffic to achieve a per-node notion of fairness. Interference-aware Fair Rate Control (IFRC) [81] differs from the previous approaches by being a full-fledged congestion control protocol. In IFRC, nodes rely on the routing tree rooted at the sink node for congestion control. Each node knows the child nodes pumping data to itself, as well as its own parent and sibling nodes within radio range. This information is used to calculate the max-min fair rate for that node. This rate is then propagated down the routing tree. Again, congestion is detected from increasing queue sizes, and sources control their transmission rate by AIMD rate control. On similar lines, algorithms like C3L [82] and TAP-fairness [36] have been proposed for mesh networks where the specific topology and flow patterns in the network are used to calculate the fair share of rates for each flow explicitly.

DiffQ shares some aspects of the above algorithms. For instance, it uses queue sizes as a measure of congestion, and a prioritized MAC to flush a full queue. Also, like in IFRC, sources control their rate in an AIMD manner. However, the exact interaction of source rate control, queue size based scheduling and MAC prioritization is different in DiffQ. It is this difference that makes DiffQ more flexible and allows it to support general traffic patterns.

2.3 Network Coding

Wireless links are known to be highly lossy. In fact, about 50% of operational links on Roofnet [5] have a loss rate higher than 30%. Network coding has been recently advocated as mechanism to improve the wireless link capacity in the presence of such loss. Nodes leverage the broadcast nature of wireless networks to cache overheard packets from neighbors, and also inform them about the packets being cached. With this shared knowledge, each node intelligently codes several new packets into one packet so that its next hops can decode the packet they need using their cached packets.

Note that network coding as performed in this manner does not alter the routing path being used. Hence it is possible to combine it seamlessly with TCP and other single path congestion control algorithms. It is in fact highly suited to TCP due to its bidirectional traffic pattern – consider nodes A, B and C in a chain topology. A transmits data to C via B. Node B can code TCP data packets being forwarded to C and TCP ACK packets being forwarded from C to A into the same packet. Since A and C already have the packets they transmitted in their cache, they can decode the packet they need, i.e. the TCP ACK and data respectively from the coded packet transmitted by B.

Along this line, Katti et al. proposed COPE [54] where they integrate TCP and XOR based network coding. The impact of network coding on TCP is studied in more detail by Huang et al. [46]. They found that if intermediate nodes delay their packet transmissions by a small time period, coding opportunities increase, but at the same time TCP throughput suffers due to increase in RTT. A per-node delay of 1-2 ms is advocated by the authors on small mesh networks where they observe a performance gain of 50-70% in throughput. This is of course an experimental result, and more analysis of the interaction between TCP and network coding needs to be done in the future. Also note that network coding as performed by COPE is a form of inter-flow coding – packets from different flows are coded together. Hence unidirectional flows, and even bi-directional flows with asymmetric network paths will not benefit from this form of network coding. In Section 2.5 we will study an intra-flow network coding technique which benefits unidirectional flows and does not rely on network paths to be symmetric.

2.4 Multi-path Routing - Spatial Path Diversity

Most congestion control studies for multi-path routing are on wired networks with the perspective of load balancing [65, 96, 55]. Load balancing is especially important in wireless networks to improve capacity. This is because routing algorithms usually choose the best links for routing, causing most routes to go through a set of nodes which happen to be well-connected. This set of nodes carry a disproportionate amount of traffic compared to the rest of the network. Multi-path routing was proposed as one way to counter this problem by performing load-balancing among available *spatially* diverse routing paths.

However, Ganjali et al. [38] show that just using the K shortest paths (which is the standard approach in multi-path routing algorithms e.g. in [74]) does not guarantee

load-balancing, since most of the K shortest paths actually lie close to each other, thus increasing self-interference. Approaches like SMR [59] and AOMDV [68] which find node-disjoint and link-disjoint paths may also face the problem of self-interference. Clearly, achieving effective load-balancing requires finding K disjoint and interference-free paths in a wireless network. Popa et al [79] achieve this by using location information and geographic routing. In their solution, instead of selecting direct shortest paths toward a destination, packets are forwarded along curves. This mitigates the self-interference problem. Traffic is split locally at intermediate nodes as well as the source. The source node performs rate control along each path. Although very effective at load-balancing, this approach is limited due to the requirement of geographical information.

As the number of independent routes increase, the congestion control algorithm has two options: aggregate the path statistics from multiple routes or maintain distinct state on a per-path basis. Maintaining per-path statistics, e.g. MPLOT [84] does not scale well. Aggregating path statistics causes two problems. First, feedback from multiple uncorrelated temporary events in the wireless network may overwhelm the sender, similar to the *feedback implosion* problem [26] in multicast. Second, the sender will conservatively tune its rate to the worst case among the available paths (since majority of the aggregated congestion events would be due to the worst path), leading to reduced performance.

2.5 Multi-path Routing - Temporal Path Diversity

Wireless being a broadcast medium, packets transmitted by a node may be overheard by its neighbors, who may help in forwarding it. This basic principle is used in opportunistic routing [17] to improve network capacity. Packets traverse several *temporally* existent network paths on the way to the destination. The challenge is to ensure that among all nodes that overhear a packet, the node closest to the destination is the one who forwards it. This is achieved in [17] by maintaining a strict MAC scheduler for all intermediate nodes between the source and the destination. To decouple the MAC from routing, Katabi et al. [19] propose MORE where opportunistic routing is combined with intra-flow network coding. In MORE, intermediate nodes randomly code overheard packets before forwarding them. Coding prevents forwarding nodes from transmitting redundant copies, since each packet is generated from the local cache of the node's overheard packets.

However, opportunistic routing improves network capacity at the cost of unpre-

dictability in wireless resource usage. The path taken by the packet is no longer known to the source, and hence conventional wired multi-path congestion control algorithms effectively break down in such environments.

2.6 Cross-Layer Optimization

DiffQ is inspired by theoretical cross-layer optimization approaches, especially differential backlog backpressure, first proposed for wireless multi-hop networks by [58]. There have been many follow-up studies (e.g., [77, 22, 80, 63, 75, 76]) 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 affect 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 [97]). 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. Because of combination of the above factors, we do not find any implementation of these solutions in real systems.

Recently, Neely et al. [77] have combined opportunistic routing, and several other approaches [80, 56, 27] have combined network coding into the cross-layer framework. Note that the main goal in such proposals is to manipulate the congestion control, routing, MAC and/or network coding components together to optimize a particular utility function. This differs from DiffQ’s goal, which is to provide congestion control as a *service* to an application without requiring application-specific modifications, e.g. in the case of MORE, DiffQ takes as input the potential forwarders for a node and performs congestion control among them. Hence the performance of DiffQ when combined with such applications would be lower than that obtained by cross-layer optimization approaches.

However, such theoretical solutions usually are difficult to implement on contemporary wireless systems, e.g. [77, 80] 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, [77] 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.

Chapter 3

Motivations

In this chapter, we motivate the need for DiffQ by studying two key congestion-related performance problems observed in wireless multi-hop networks.

The following results are derived from a testbed network of 46 indoor IEEE 802.11 nodes deployed over a three story building of 100,000 sq. ft. space (more detailed description in Section 7). Tests are conducted between random pairs of source and destination nodes. The maximum number of hops in our testbed is around 6 or 7 hops. MORE is representative of the multi-path schemes which can benefit from using DiffQ. In each experiment, we increase the number of concurrently running MORE flows. We repeat each experiment 20 times. Figure 3.1 (a) shows the CDF of per-flow throughput of MORE as we increase the number of flows in each run. We do not employ any congestion control for this experiment. As the number of flows increase, we observe that the number of starved flows increases quickly. With 32 flows, over 50% of flows are completely starved.

We also run TCP in the same setup and compare the result in Figure 3.2(b). TCP also experiences much starvation with about 30 to 40% of flows starving. Figure 3.3 (c) compares their average per-run total throughput. From these graphs, we find that the problems of these two protocols under high traffic load are quite different. For MORE, since it does not use any congestion control, under higher load, the total throughput of MORE substantially reduces and eventually under 32 flows, not many flows get much utility out of the network; from its CDF of the 32 flow case, we see that over 90% of flows are getting less than 100 kbps. This is a clear sign of congestion collapse. On the other hand, the maximum throughput that TCP flows achieve is almost invariant of the number of concurrent flows running. This shows that TCP experiences extremely unfair bandwidth sharing among its

Figure 3.1: CDF of per-flow average throughput of MORE. Many concurrent MORE flows running without congestion control experience congestion collapse.

competing flows; while a few flows get a disproportionally larger amount of throughput, many flows are starving.

Figure 3.4 further illustrates the fairness problem of TCP. In this test, we run three flows as illustrated in Figure 3.4 (a) where one multi-hop flow (flow 1) starts first and the other two one-hop flows join later. Figure 3.4 (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*. 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

Figure 3.2: CDF of per-flow average throughput of TCP. Many concurrent TCP flows experience unfairness issues.

bandwidth. The situation escalates to the eventual starvation of the first flow.

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 [15], EXACT [20], WXCP [89] and ECN[29]. It requires coordinated rate control among competing nodes within the same interference range.

Figure 3.3: Average per-run total throughput for MORE and TCP.

(a) A test scenario; dotted circles represent interference ranges.

(b) Instantaneous throughput of three TCP flows

Figure 3.4: 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.

Chapter 4

DiffQ - Design

In this chapter, we present the design of DiffQ for single and multi-path routing algorithms.

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. We define a *neighbor* by its symmetric relation; if a node A is a neighbor of another node B , then B is also a neighbor of A and we say there is a *link* between the two nodes. In wireless networks, asymmetric links are common. However, most wireless routing protocols rely on symmetric links where at least probes (e.g., ETX) can be acknowledged. Any ETX-metric based routing protocols including OLSR [49], SRCR [16], ExOR [17] and MORE [19] use this property implicitly.

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. Below we first describe congestion control for single-path flows and then for multi-path flows.

4.1 Congestion control for single path routing

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.

$$QD_i(d) = |Q_i(d)| - |Q_j(d)| \quad (4.1)$$

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 6. 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) as used in [23] or [83]. Our pseudo-code of DiffQ below describes a version of AIMD.

Algorithm *Source Rate Control()*

1. $F = \text{Destination of flow originating at this node}$
2. $qlen \leftarrow |Q_i(F)|;$
3. **if** $qlen > \text{QUEUE_THRESH}$
4. $rate = rate/\beta;$
5. **else**
6. $rate = rate + \alpha;$

Algorithm *Forwarder Algorithm()*

1. $\Delta \leftarrow \text{Number of priority levels supported by MAC};$
2. $D \leftarrow \text{Maximum per-destination queue size};$
- 3.
4. **Flow Scheduling**
5. $F \leftarrow \text{argmax}_d QD_i(d);$
6. $P \leftarrow \text{HOL packet of } Q_i(F);$
7. $P.\text{priority} \leftarrow \text{MAX}(\lceil \frac{QD_i(F)}{D} \Delta \rceil, 0); P.qlen \leftarrow |Q_i(F)|;$
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. $F \leftarrow \text{Destination of } P;$
15. Enqueue P into $Q_i(F);$
- 16.
17. **On reception of packet P from node j**
18. $F \leftarrow \text{Destination of } P;$
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 node 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 node Y to increase. This has effect of increasing the forwarding rate at node 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 al-

though 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 3.4. 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.

DiffQ is different from the queue occupancy based technique used in [48, 47] which sets the contention window size inversely proportional to the queue size, thus giving a higher priority to a node with a larger queue. While it also provides more congested nodes to send packets fast, the difference comes when an area is congested and many neighboring nodes are congested together. In that situation, the queue occupancy scheme allows the node with the largest queue to send packets first whether or not its next hop is congested or not. Thus, it is not clear whether those transmissions will relieve congestion. With DiffQ, those nodes with the largest queue differential make the first transmissions, allowing transmissions to occur always to the direction where congestion can be relieved. This difference is illustrated in Figure 4.1 where node E has a largest queue. According to the queue occupancy scheme, E will get the highest channel access priority. But in this case, E 's transmission only adds congestion already occurring in node F . But in DiffQ, nodes B and F get a channel access first as they have the largest queue differential values within their sensing ranges. This relieves congestion from the locations where it first happens and also backpressure quickly propagates back to the up-stream nodes.

4.2 Congestion control for multi-path routing

For the single-path algorithm, we have the information about the next hop router for each packet from the routing layer. For multi-path routing, especially opportunistic routing, each packet does not know the next forwarding node(s); this information is available only after the reception of that packet by the neighboring nodes. One or more neighboring

Figure 4.1: The operation of DiffQ. The thick lines represent channel accesses. According to DiffQ, nodes B and F get the channel access first as they have the largest queue differential values within their sensing ranges.

nodes receiving that packet may become the next forwarding routers.

Existing opportunistic routing protocols use source-based routing where each packet contains information about the set of candidate forwarding nodes in the network. This means that at each node i , we can narrow down the possible set of forwarding routers by taking the intersection of the candidate forwarding router list in a forwarded packet p and the one-hop neighbors of that node. Let us call that intersection F_{ip} . We assume that each packet provides the candidate forwarding list. This is not a difficult requirement as most multi-path routing algorithms such as MORE and ExOR use source routing. This assumption requires information sharing between routing and congestion control, but does not require changes in the operation of routing algorithms.

When evaluating the priority of an HOL packet p with destination d , each node i computes F_{ip} and then computes the queue differential of that packet with respect to a neighbor $j \in F_{ip}$. The queue differential value, $QD_i(d)$, of p at node i is set to be the

minimum of such queue differential values. We use that value for setting the priority of that packet at node i . The following defines this operation.

$$QD_i(d) = \min_{j \in F_{ip}} \{|Q_j(d)| - |Q_i(d)|\} \quad (4.2)$$

Node i schedules for transmission the packet with the highest priority (i.e., the maximum queue differential) among all HOL packets and the priority of that packet is used by the MAC layer to schedule the transmission. Each node schedules its transmission based on the worst case next hop router. If there is a queue buildup at any next hop in F_{ip} , then the priority of packet p gets reduced and thus that packet will get a lower channel access priority, which effectively reduces the transmission rate of the flow packet p belongs to. This approach is conservative because the packet may not be received by that worst case next hop.

4.3 Prioritized MAC

DiffQ's scheduler assigns priorities to packets and passes them on to the MAC for transmission. The MAC needs to honor this priority relative to the packets already queued at the MAC as well as priorities of packets from neighboring nodes. Hence DiffQ requires a MAC capable of prioritized packet transmissions. There already exist several MACs in literature which allow prioritized transmissions. In the next chapter, we shall take a systematic look these proposals and compare them from a design and performance perspective.

Chapter 5

QoS MAC

DiffQ requires a MAC which supports prioritized transmissions. There exists a rich history of literature on MACs capable of prioritized transmissions. As part of the design process of DiffQ, it is important to pick and choose the “best” MAC which performs this function. Surprisingly, we find no comparative study of such MACs in literature. The core reason for this is as follows. In response to various MAC layer fairness and QoS¹ issues, the wireless community have come up with numerous “point solutions”, with one-to-one comparison of their solution with existing MACs for the specific fairness model they target. Such solutions are closely tied to a particular fairness model, making a cross-model comparison of MACs difficult.

However, with some tradeoff of flexibility and efficiency, many of these MACs can be made relatively free of a specific fairness model. This decoupling of the MAC from the fairness model is desirable because it allows objective testing of the MAC under different fairness models and also allows independent evolution of fairness policies and research into medium access issues. Hence, in this chapter, we take a brief detour from our main theme – DiffQ, and focus on a systematic evaluation of various MACs for their support of prioritized transmissions. We find that the ability to prioritize packet transmissions in the MAC is a desirable property not just for DiffQ but also to implement several different fairness policies like proportional rate allocation, EDF scheduling, proportional fairness and temporal fairness.

¹We consider QoS to be a form of a user-defined notion of fairness under heterogeneous environments e.g. a particular policy may enforce the rule that VoIP flows should always have higher priority compared to best effort flows like TCP, so even though at the flow level this policy is inherently “unfair”, from the user’s perspective it is “fair”. We will henceforth use fairness and QoS interchangeably in this chapter.

5.1 Introduction

One of the frequently cited problems of current wireless networks is related to lack of fairness and QoS. The reported problems of MAC include, to name a few, starvation [86, 39, 99], priority inversion [100], inequitable allocation of bandwidth [99], lack of QoS support [51] and multi-rate LAN unfairness [43]. The research community has come up with several novel solutions [86, 100, 51] to such problems. The common thread among such solutions is that they often require altering the MAC protocol significantly to achieve the authors' notion of "fairness", in effect embedding a fairness policy into MAC. Clearly, this makes them "point solutions", wherein the proposed changes to MAC proposed for one solution are incompatible with those for other solutions. To address this issue, Nandagopal et al. [73] propose a general analytical framework to allow automatic generation of MAC layer algorithms tailored to specific utility functions. In addition, extensive prior work [21, 64] on cross-layer optimization optimizes MAC functions to support a specific predefined utility function. However, note that in their work, each specific instance of a MAC algorithm is still specific to a fairness policy (or a utility function) used to generate it, and may not be used to support other fairness policies.

We perceive fairness to be application-defined properties governed strictly by the utility of the application. In an ideal wireless infrastructure, MAC simply does what it does best – medium contention resolution, while exporting enough control "knobs" to the application so that the application may enforce its own fairness policy. This *application-centric* notion of fairness follows directly from the well-known system design principle to separate mechanisms from policies. This decoupling between the fairness policy and fairness mechanism is highly desirable since (1) it allows independent evolution of fairness policies and research into medium contention mechanisms and (2) it allows objective testing of MAC under different fairness models (The second property is especially valuable to us in the context of DiffQ, since it allows objective evaluation of different MACs for the support of the fairness policy of DiffQ.). In fact, many of the existing MACs do exhibit such decoupling, albeit with varying degrees of flexibility and efficiency. This brings up an important question – how do these different MAC protocols measure up in the support of different fairness policies, and in the process how much do we lose in terms of flexibility and efficiency? To the best of our knowledge, this question has remained unanswered so far especially in the context of a real-world multihop wireless networks.

One main reason behind this lack of a comprehensive evaluation study is that most of the cited solutions [86, 100, 51, 52] require significant changes to MAC layer which were not possible to be implemented on current commercial radios. This is because efficiency considerations make most of the MAC layer functions in current radios be performed in firmware which is difficult to modify. This limitation forces even the one-to-one comparison of most cited solutions with existing work for a specific fairness policy to be limited to simulation. In the process significant real-world issues such as system latency, ambient channel noise, time synchronization issues, wireless channel errors, specific radio limitations such as minimum/maximum packet sizes etc. are simply glossed over. To some extent, this problem has been mitigated by the development of Software Defined Radios (SDRs) [3], which allow all components of the radio (including PHY modulation) to be specified in software. However the prohibitive per-unit costs of SDR radios make it difficult to be used for large testbed deployments. In this chapter, we address this issue by conducting an extensive experimental evaluation of existing MACs which provide fairness support in a real multihop wireless testbed environment.

Our methodology is as follows. We first survey existing work on fairness issues in single/multihop wireless environments and their corresponding proposed solutions. A common technique to construct mechanisms for implementing various fairness policies is to (1) assign priorities to data packets, (2) manipulate these priorities dynamically to implement a specific policy dictated by applications, and (3) ensure channel access to be ordered in terms of the priorities. Several schemes [52, 51, 87, 92] follow this approach where fairness mechanisms are supported by *priority resolution* (PR) – the mechanisms to implement the prioritized access of packets. We classify such MACs supporting PR into three categories based on the way they perform PR, and identify three representative schemes for each of the three categories.

1. Differentiated backoff based schemes, represented by 802.11e [13, 60].
2. Beacon-based schemes represented by EY-NPMA [88, 100, 11].
3. Scheduling-based schemes represented by DWOP [51, 52].

This categorization significantly reduces the number of combinations that need to be evaluated without overly compromising the breadth of our study. We implement these three schemes on the ZigBee-based CC2420 radio on the MicaZ sensor platform.

This radio is different from contemporary 802.11 radios in that it exposes a considerable amount of functionality such as clear channel assessment (CCA), transmission power and frequency control, etc. to software while still maintaining a raw bitrate of 250Kbps. The implementation of the above MACs on such a radio for multihop environments is a significant engineering and design challenge since it is important to make sure that our design decisions do not affect the experimental results and that the conclusions are valid not just for the CC2420 radio but also for other CSMA-based radios. The implementation of EY-NPMA for multi-hop environments is particularly difficult as it is designed only for wireless LAN single-hop networks and thus does not provide any mechanisms for hidden terminals. We extend EY-NPMA for multihop environments and call our extension *Siren*. In the next step, we identify and implement a list of commonly used fairness policies including static priority, temporal fairness proportional fairness and earliest deadline first (EDF) on top of these three MAC protocols. Finally we compare the performance of these MAC protocols in enforcing these fairness schemes on our 30 node multihop wireless testbed.

The rest of the chapter is structured as follows. Section 5.2 contains the related work. In Section 5.3 we describe the three MAC protocols we implement, while in Section 5.4 we present implementation details on the CC2420 ZigBee radio. In Section 5.5 we describe the fairness schemes we use for comparison. In Section 5.6 we present experimental results and we conclude in Section 5.7.

5.2 Related Work

Substantial research exists on the individual components of a fairness control framework i.e. fairness policy, PR, and medium access. However we will focus only on the PR scheme in this section. Priority resolution has been achieved in the MAC layer by various means. Broadly, they can be classified into three approaches – beacon based [88, 11, 100] or backoff based [13] or scheduling based [52, 51]. This bifurcation can be traced back to the original two competing standards for the wireless MAC – 802.11/802.11e [13], which uses a differentiated backoff strategy, and HIPERLAN/1 [11] which uses beacons (the scheduling-based algorithms came later).

IEEE 802.11e [13] supports 4 priority classes. Service differentiation between them is achieved by assigning each class different values of AIFS and CW_{min} and CW_{max} . High priority classes have a short AIFS and small values of CW_{min} and CW_{max} allowing them

prioritized access to the channel. Absolute service differentiation (where high priority traffic is *never* interfered by lower priority traffic) can be achieved in IEEE 802.11e by assigning CW_{min} of lower priority classes to be larger than $AIFS + CW_{max}$ of higher priority classes.

Vaidya et al. propose a PR scheme, BTPS [100] (Busy Tone Priority Scheduling) supporting two priority levels. In this scheme, nodes are required to listen on three channels during idle periods – a data channel and two narrow-band busy-tone channels. A node with a backlogged high priority packet transmits a busy tone signal (BT1) every M slots. Neighbors that hear BT1 will forward this signal to the node’s two-hop neighbors on a different band (BT2). The node’s two-hop neighbors with backlogged low priority data that hear BT2 will defer their transmissions. This ensures the transmission of high priority packets. BTPS solves the hidden terminal problem effectively, but requires nodes to listen on multiple channels during idle periods. It is also constrained to two priority levels.

In black burst contention [88, 85], nodes with backlogged packets first wait for a short time period and then transmit small pulses of energy or *black bursts* which serve to “jam” the channel. The length of a black burst is an increasing function of the priority of the backlogged packet and/or how long a node has been waiting for access to the channel. After transmitting the black burst, a node observes the channel for a short period of time to check if any other node is transmitting a longer black burst. If not it will transmit the data packet.

Kanodia et al. propose a scheduling-based PR algorithm for multihop wireless networks [51, 52]. In this scheme, nodes piggyback their priorities onto RTS, CTS, DATA and ACK packets. By overhearing such packets, nodes become aware of the priorities of the nodes in their neighborhood. This information is then used to co-ordinate channel access so that the order of packet transmissions agree with the order of priorities among the nodes. Since both schemes rely on overheard information, their performance degrades in lossy channels.

Nandagopal et al. [73] take an analytical approach and propose a general framework to allow automatic generation of MAC layer algorithms tailored to specific utility functions. This approach is promising since it allows any given fairness policy represented by a utility function to be implemented in a MAC. However, each such MAC is still a “point solution” since the specifics of the fairness policy is embedded into it, preventing it to be used for other fairness policies.

5.3 Medium Access Mechanisms for Fairness and QoS

We classify MAC schemes into three categories based on the manner in which they perform PR – (a) differentiated backoff-based schemes, (b) beacon-based schemes, and (c) scheduling based schemes. In this section, we describe these categories and their representative schemes used for our evaluation.

5.3.1 Differentiated backoff-based schemes

In such schemes, nodes access the channel with a backoff-period which is a function of the priority of their head-of-line packet, e.g. a node A with higher priority² will access the channel with a shorter backoff compared to neighboring node B with lower priority. This ensures that A captures the channel with high probability compared to B , with the exact probability of channel capture being a function of the relative backoff periods of A and B . IEEE 802.11e and its numerous variants [13, 60] fall in this category. Note that the original standard defines 4 priority levels with varying but overlapping backoff periods for each priority. This allowed lower priority nodes to sometimes capture the channel in the presence of high priority nodes, or priority inversion [100]. In order to achieve *absolute prioritization*, where the higher priority nodes always capture the channel earlier than lower priority nodes, Aad et al. [14] propose having non-overlapping backoff periods for each priority, i.e. $DIFS_i = DIFS_{i-1} + CW_{i-1}$, where CW_i is the backoff window for priority level i . We use this version of 802.11e for our evaluation.

5.3.2 Beacon-based schemes

In beacon-based schemes, nodes rely on short bursts of energy to inform their neighbors of their priority. The actual priority is encoded in the length of the beacon (e.g. black burst [88], prioMAC [85]) or the time (e.g. EY-NPMA [11]) at which they are sent. A subset of such schemes also rely on an additional channel to send beacons, (e.g. BTPS [100]). We focus our evaluation on EY-NPMA which uses short beacons sent at pre-defined moments in time to encode priority information. The reasoning for choosing EY-NPMA is twofold. First, encoding the priority in the length of the beacon wastes

²For brevity, we will abuse terminology and refer to "nodes with head of line packet priority i " as "nodes with priority i ".

energy. Second, using a separate channel for beacon transmissions is not possible in most contemporary radios which have only a single transceiver.

We refer the reader to the original EY-NPMA paper [11] for full details on its working. We give a brief description of its working. In EY-NPMA, each node goes through a PR phase followed by a contention resolution (CR) phase. The PR phase is as follows. As soon as the channel becomes idle, a node with a back-logged packet with priority i waits for i *beacon periods* which is defined to be the time to fully transmit a short *beacon*. At the end of i beacon periods, if the channel is idle, the node transmits the beacon and enters contention for the medium. However, if it hears any beacon transmission prior to the i beacon periods, then it recognizes that a higher priority transmission is pending and defers its own transmission. If there exist more than one node with the same highest priority, they will all transmit beacons simultaneously and will enter the CR phase together. At the end of the PR phase, one or more nodes with the highest priority within two hops are left in contention, and they contend for the medium in the CR phase.

5.3.3 Scheduling-based schemes

In scheduling-based schemes, nodes disseminate priority information well before the actual packet transmissions and then co-ordinate their transmissions in such a way that the order of packet transmissions matches the advertised priorities. Distributed priority scheduling [51] and DWOP [52] follow this scheme. They are similar in the way they collect priority information: nodes piggyback their packet priorities onto RTS, CTS, DATA and ACK messages, which are overheard by neighbors. Thus each node builds a database of HOL (head-of-line) packet priorities of all its neighbors within a two-hop communication range. RTS and CTS messages of a node inform its immediate neighbors and potential two-hop interferers respectively about its HOL packet priority. DATA and ACK messages inform the neighbors that the packet has been successfully transmitted so that they may update their priority database. In addition, DATA and ACK messages also inform neighbors about the priority of the next packet in its queue.

In distributed priority scheduling [51], the backoff period of a node is calculated based on its priority relative to other priorities in the priority database. DWOP [52] is similar, except that it is stricter in its enforcement of packet priorities where nodes are not allowed to send an RTS message unless the maximum priority level in their cache

corresponds to their HOL packet. We choose DWOP as our representative scheme due to this priority resolution mechanism as it provides the absolute prioritization property seen in our version of 802.11e (Section 5.3.1) and EY-NPMA. As we shall see later in Section 5.5, absolute prioritization allows a simplified implementation of various fairness policies.

5.4 MAC Modifications for ZigBee and Multihop Environments

We modify 802.11e, EY-NPMA and DWOP MACs to use them for multihop wireless environments under the CC2420 radio. In this section we describe how each protocol is modified. Although the details are to some extent CC2420-specific, we believe that similar issues appear for other CSMA-based radios like commercial off-the-shelf 802.11 radios. Before going into specific details for each MAC, we first give a brief description of the CC2420 transceiver.

5.4.1 CC2420 Transceiver – Overview

The MicaZ [9] sensor node contains the CC2420 [2] transceiver which is a ZigBee [10] compatible radio. It has a link speed of 250Kbps, resulting in a per-byte transmission time of 32us. It supports 8 different transmission power levels ranging from a minimum of -25 dBm ($< 10\mu\text{W}$) to a maximum of 0 dBm (1 mW). It contains two buffers of size 128 bytes each for outgoing and incoming packets respectively. Packets placed in the outgoing buffer need to strictly adhere to the ZigBee packet format, else they are dropped.

The ZigBee [10] standard specifies the minimum PHY layer header of 5 bytes (4 bytes preamble and 1 byte synchronization), a MAC layer header of 9 bytes and 2 bytes of CRC. This implies that the minimum packet size with no payload in CC2420 is 16 bytes. Also, the standard specifies that the maximum packet size (including PHY, MAC headers, CRC and payload) is restricted to 128 bytes.

Packet transmissions may operate in two modes – with link-layer ACK enabled or disabled. When the link-layer ACK is enabled, the intended receiver of a packet, upon packet reception, will automatically send back an ACK packet of 11 bytes, provided the received packet passes the CRC test. The CC2420 also disables overhearing and packets not addressed to itself are dropped in the hardware itself, and never reach the MAC driver.

When the link-layer ACK is disabled, CC2420 does not send ACK for received packets and leaves this responsibility to higher layers.

5.4.2 Large Packet Size Emulation

The maximum packet size of CC2420 is 128 bytes which is highly restrictive compared to commercial 802.11 radios with MTU of 1500 bytes. To overcome this limitation, we maintain a *virtual packet* at the MAC layer in which the transmission of larger packets is emulated. The size of a virtual packet is set to X times 128 bytes. For every data packet of size 128 bytes received from the upper layers, the MAC driver retransmits the same packet X number of times. The receiver counts the number of CRC-valid copies of a packet it receives. If it receives X copies of the same packet, it will send ACK. Otherwise it will silently discard all copies of the packet. The sender, after sending X copies, waits for ACK. If it does not receive ACK, it will signal a failure to the upper layer, and try again. This emulates closely the behavior of a packet of size $128 \times X$ bytes. In our experiment, we set X to 3 for an emulated packet size of 384 bytes.

5.4.3 Overhearing

CC2420 disables overhearing when the link layer ACK is enabled. However, overhearing is an important requirement for several of the fairness policies which we shall discuss in detail in Section 5.5 where overheard information is used to decide packet priorities. Hence, we modified the default MAC driver for CC2420 to send MAC-level ACKs, and disabled the automatic link-level ACK feature of CC2420. We verified that this does not cause any appreciable loss in channel efficiency.

5.4.4 802.11e

IEEE 802.11e pauses backoff timers when the channel becomes busy and resumes them again when the channel becomes idle. This is difficult to implement in CC2420. When transmitting a packet, the MAC driver first loads the packet into the CC240 transmit buffer. Once the buffer is loaded, transmission is initiated by an explicit command (STXONCCA command) to the hardware. After that, the radio checks the noise level on the channel (CCA or Clear Channel Assessment) for 4 byte-times or 128us. If the CCA detects that the channel is idle, the packet is put on the channel byte by byte. If not, the radio reports

an error and takes an appropriate error handling action. Hence the radio does not perform the CCA continuously in the transmit mode. This makes it difficult to pause and resume backoff timers based on the activity on the channel.

To get over this limitation, we do not follow the 802.11e standard of pausing backoff timers. In our implementation of 802.11e, which is called PMAC, when a node has a packet to transmit, it takes a random backoff. During the backoff period it does not check the channel. At the end of the backoff period, it will issue the STXONCCA command to the PHY. If it receives an error because the channel is busy, the node will take another random backoff. Otherwise the PHY will begin transmission.

PMAC uses non-overlapping backoff windows to support absolute prioritization. Despite this, it is still possible to have priority inversions if the backoff window is paused. To see why, consider two nodes A and B , with A having a higher priority than B . A will naturally use a shorter backoff and will access the channel multiple times. However, during each channel access, there is some small channel idle time during A 's backoff period. During this period, B 's backoff timer keeps counting down since the timer resumes during channel idle times and pauses during channel busy times. Hence eventually B will access the channel with a shorter backoff than A causing a priority inversion. Thus by not pausing backoff timers and forcing B to take a new backoff every time, PMAC completely avoids priority inversions.

A second issue with 802.11e is the use of exponentially increasing backoff timers. In the standard, if a node's backoff timer expires, it checks the medium and if it is still busy, it doubles the backoff window and takes another random backoff within this window. However, PMAC needs to support several priority levels and supporting multiple exponential increases in the backoff window leads to large backoffs for each priority level. This greatly increases overhead for each priority level. As we shall see in Section 5.4.7, even while using a fixed window without exponential backoffs, PMAC still incurs high overhead since every priority level requires one backoff window. Hence we choose to use fixed backoff windows for PMAC.

5.4.5 EY-NPMA

EY-NPMA was originally intended for a single-hop WLAN environment where every node can sense the transmissions of its neighbors, and hence can enter the PR and CR phases in synchrony. To enable this behavior for multi-hop environments we require

time-synchronization among neighboring nodes. We refer to this modified protocol as *Siren* to differentiate it from EY-NPMA for the rest of this section. In *Siren*, time is slotted.

Note that the overhead incurred in the PR phase of EY-NPMA is the time taken for a beacon transmission times the number of priority levels supported. Clearly, a smaller beacon size incurs less overhead. Early radios (e.g. CC1000 [1]) support bit streaming which can be used to construct extremely small beacons. However the new generation of packetizing radios (e.g. CC2420 [2], most commercial 802.11 radios) does not support bit streaming and allows only valid packets with standard compatible headers to be transmitted. This limitation significantly increases the overhead of the priority resolution phase if the minimum packet size is large. *Siren* overcomes this limitation by the use of *beacon sensing*. The basic idea is that a node does not need to completely receive a beacon to sense a higher priority node in its vicinity; it only needs to detect the presence of the beacon on the channel through CSMA. We now discuss this concept in more detail.

Beacon Sensing

Figure 5.1: The working of *Siren* for nodes *A*, *B*, *C* and *D* with priorities 1,1,2 and 3 respectively is shown above. Both *A* and *B* send beacons after one CCA time. *C* and *D* sense the beacons after two CCA times, and hence they postpone their transmissions to the next time slot. Both *A* and *B* enter the CR phase and contend for the channel. *A* picks a shorter backoff and successfully transmits a packet while *B* senses this packet and defers its transmission to the next slot. In the third time slot, *C* transmits a beacon after two CCA times which is sensed by *D* after three CCA times. Hence *D* defers its transmission while *C* transmits its packet. Finally *D* transmits its beacon and packet in the fourth time slot.

Figure 5.2: The expanded view of one time slot which clearly shows the PR and CR phases.

A node with a priority i waits for i CCA times which is the minimum time to sense a (beacon) transmission from a neighbor. At the end of the i -th CCA time, if the channel is idle, the node transmits the beacon and enters the CR phase. However, if it senses any beacon before the i CCA times, then it defers its transmission to the next time slot. If there exist multiple nodes with the same priority, they all transmit beacons simultaneously and enter the CR phase together. As illustrated in Figure 5.1, the overhead for the PR phase in Siren is the CCA time multiplied by the number of priority levels plus one beacon transmission time. Since the CCA time for most radios is much smaller than the minimum valid packet transmission time, beacon sensing results in a significant overhead saving. The PR and CR phases of Siren in each timeslot is shown in Figure 5.2

The minimum packet size limitation of 16 bytes corresponds to a beacon transmission time of $16 \times 32 = 512\mu s$. By means of experiments, we found out that CC2420 requires sensing the medium for 10 byte-times = $320\mu s$ to sense any activity on the channel. Hence we use a beacon sensing time of $320\mu s$ for our implementation. Unless otherwise specified, we use 6 priority levels for Siren incurring an overhead of $320 \times 6 + 512 \mu s = 2.4ms$ per packet transmission for priority resolution.

Beacon Transmission Power

In Siren, a node uses beacons to inform its neighbors about its priority. However, hidden terminals located at two hops away from the node, and at one hop away from its intended receiver may not sense the beacon and may interfere with packet reception at the

Figure 5.3: Figure shows how node *A* uses a higher beacon transmission power to block out potential interfering nodes for receiver *B*. *C* is a one-hop node whose transmission to *G* would interfere with both *A*'s transmission and *B*'s reception. *F* and *E* are two-hop nodes whose transmission would affect *B*'s reception. *I*'s transmission to *D* would not affect *A* or *B*, however *D*'s acknowledgment to *I* would interfere with *B*'s reception. Node *H*'s transmission would not affect *A* or *B*, but it will get blocked out. This is an instance of the exposed terminal due to Siren's beacon transmission.

receiver [86, 39], as illustrated in Figure 5.3 This problem is not handled in EY-NPMA.

To fix this problem, we adopt a higher transmission power for beacon transmissions compared to data transmissions to preserve the correctness of PR across two hops. The transmission power for beacons should be enough to reach all potential interfering nodes for *A*'s intended receiver. This approach has been used in [50, 71, 98] as well. In our study, we set the beacon transmission power of all nodes to be roughly three times that of their data transmission power for a simplified implementation. This is decided based on the following reasoning. Assuming free-space propagation, where the transmitted signal drops by square root of distance, quadrupling transmission power for beacons corresponds to doubling distance, hence ensuring that the beacon is received by all two-hop interferers. However, since beacon packets need to be only sensed, not received, we find through experiment that simply using three times the data transmission power works for most cases.

Using the same beacon transmission power for all nodes in the network is a conser-

vative approach which may cause the beacon transmissions of some nodes to reach farther away from interfering nodes and shut down non-interfering nodes from transmission leading to loss of network capacity. This problem is called the *exposed terminal* problem. An example is shown in Figure 5.3. Node H will be shut down by A 's beacon, even though its transmission will not interfere with A 's transmission to B . Recent work has established that it is possible to detect the interference relations between nodes on a run-time basis [103]. In addition, transmission power can also be tuned so that transmissions reach only a set of interfering nodes [62]. Hence it is possible to tune the beacon transmission power on a per-node basis, so that it reaches the set of interfering nodes and no farther. This will not eliminate all cases of exposed terminals, but will definitely result in improvement in network capacity. However, we do not explore this option in this thesis, and leave it for future work. Our experimental results presented in Section 5.6 are obtained without this tuning. We use -5 dBm (316 μ W) transmit power for data packets, and 0 dBm (1 mW) for beacons.

Time Synchronization

Siren does not require global clock synchronization. The protocol works as long as the sender is synchronized with all potential interferers (both one and two-hop nodes). Hence conventional distributed clock synchronization algorithms which depend on hop-by-hop clock synchronization [37, 69] are appropriate for Siren. We use FTSP [69] for our testbed.

We use the FTSP [69] algorithm for time synchronization.³ In our multi-hop testbed comprising of 30 MicaZ sensor nodes, the FTSP algorithm takes 4 minutes (with time synch messages broadcast every 10 seconds) for all nodes to synchronize. All data transmissions begin 4 minutes after the start of the experiment. Since time synchronization messages are sent as broadcast they are prone to loss due to collisions with data packets. This makes it difficult to maintain time synchronization under high network load. To prevent this problem, synchronization messages are sent with a priority level 0.

³The current implementation of FTSP in TinyOS contains a bug where invalid timestamps are sent in time synchronization messages under high network load. We could not find the source of this error, however we observed that such invalid timestamps usually had a value less than 3s. We thus dropped all timestamp messages which showed a sending time less than 3s, and started the time synchronization module after 5s of the beginning of the experiment.

Figure 5.4: DWOP problem scenario in lossy networks. Node *A* reserves the channel with a high priority packet using an RTS/CTS exchange. The small size of RTS/CTS allows nodes far away from *A* (e.g. *C* and *D*), to overhear *A*'s priority and enter *A*'s priority in their priority cache. The consequent data packet from *A* informs neighboring nodes that *A*'s transmission is over and also advertises the priority of the *next* HOL packet. However, due to the higher packet loss rate caused by the large data packet size, *A*'s transmission to *B* is not overheard by *C*. Hence *C* never knows that *A* has completed transmission. This causes *C* to defer its own transmissions needlessly.

Contention Resolution in Siren

The nodes that survive the PR phase compete on an equal basis for channel accesses in the CR phase. Siren uses CSMA/CA with fixed window backoffs for CR. Nodes take a random backoff and sense the channel at the end of the backoff period. If a node senses a carrier, then it simply goes to the next slot where it runs the PR phase again. This scheme provides equal chances to all competing nodes in the CR phase. We choose this CR mechanism because it is simple to implement.

In extending EY-NPMA and implementing Siren, we have made several simplifying design choices as discussed above. While there is room for more optimization, we confirm that Siren performs fairly well compared to other protocols. So we believe that the current simple implementation would still prove to be meaningful for a comparison study.

5.4.6 DWOP

In DWOP, nodes exchange priority information by overhearing RTS/CTS/DATA/ACK exchanges onto which the packet priorities are piggybacked. The default MAC in the CC2420 is a CSMA MAC without RTS/CTS support. We added support for RTS/CTS in the CC2420 MAC. RTS and CTS are sent as minimum size packets (16 bytes) plus 2 bytes for the packet priority.

One issue with DWOP is that in multihop wireless environments, it is possible that priority information often become stale or lost. This happens due to several reasons. First, packet priorities may change dynamically even at the time-scale of medium access (a few hundred microseconds to a few milliseconds), requiring nodes to perform an RTS/CTS exchange every time the packet priority changes (the CTS is necessary to inform nodes two hops away). Second, priority information could be lost due to collisions and interference. Stale or lost priority information may lead to deadlock. Such a scenario for DWOP [52] is illustrated in Figure 5.4. To deal with the deadlock caused by lost or stale information, we introduce a simple timeout feature which works as follows. A node with a packet to transmit checks the priority table. If its priority is lower than the highest priority in the table, it takes a backoff for a random period of time and initializes a counter. At the end of the backoff period, it will check the table again. This cycle is repeated N times, incrementing the counter at every check. If the counter reaches N with the highest priority remaining unchanged in the table, the node is allowed to go ahead and transmit the packet. We verified that $N = 20$ works the best in our experiments.

5.4.7 Relative Overheads

Figure 5.5: Normalized priority resolution overhead for Siren, EY-NPMA and 802.11e with non-overlapped backoff periods with different data packet sizes.

In this section we quantify the overhead for each MAC by comparing the time taken for PR normalized by the total time for transmission of one data packet (with different data packet sizes and 10 priority levels). We plot the result in Figure 5.5. ZigBee radios have

a link speed of 250Kbps (resulting in a per-byte transmission time or “byte-time” of $32\mu\text{s}$, CCA time of $320\mu\text{s}$ and minimum packet transmission time of $512\mu\text{s}$. For PMAC (802.11e), the overhead for priority resolution is the backoff period times the number of priority levels. We assume the conventional backoff period of 32 byte-times; 1.024ms for ZigBee. Siren’s beacon sensing saves about 20% overhead compared to a naive implementation of beaconing (marked as EY-NPMA in the figure) in ZigBee radios. PMAC incurs significantly more overhead compared to Siren. This is because in PMAC, the PR and CR schemes are interlocked with each other, and hence each additional priority level incurs the overhead of one additional backoff period while in Siren, adding an additional priority level requires only an additional CCA time. An important attribute of DWOP is that it supports infinite priority levels – the overhead is always the time taken for the RTS and CTS exchange.

5.5 Implementing Fairness Policies

In this section, we describe how different fairness policies are implemented on top of PMAC, Siren and DWOP. Each implementation is mostly agnostic to the kind of MAC used, except in a few cases which we highlight in this section. The only requirement is that the MAC should support absolute prioritization, which is supported by all of our MAC implementations discussed in Section 5.4. We implement the following fairness and QoS policies: static priority, EDF, proportional rate allocation, and proportional fairness. Below, we present our implementation; since the implementation of static priority is straightforward, we omit its description.

5.5.1 Proportional Rate Allocation and Time Fairness

Consider the problem of proportional throughput allocation in a WLAN [40]. Let there be N nodes in a WLAN which require service differentiation to the effect that their obtained throughput values are in the ratio $r_1 : r_2 : \dots r_N$. A related problem is maintaining *temporal fairness* (where all nodes receive equal channel time) in multi-rate WLANs. Suppose N stations access the channel with different transmission rates $R_1, R_2, \dots R_N$ (due to varying channel quality) in a WLAN where all nodes can hear each other. IEEE 802.11b has been shown to be highly “unfair” in such an environment [43]. A solution is to assign equal time-shares among competing nodes on the channel. Existing work [44, 42, 92, 101] implements this solution by directly modifying MAC. The temporal fairness problem can

also be framed as a proportional rate allocation problem in the following manner: if the N nodes access the channel so that their obtained throughputs are proportional to their transmission rates, i.e. $R_1 : R_2 : \dots R_N$, then the time spent by each node on the channel is equalized.

We propose the following algorithm for achieving *approximate proportional rate allocation*. Let there be P priority levels available. A station i , $1 \leq i \leq N$, sets the priority p_i of its outgoing packets by executing a maximum of $P - 1$ Bernoulli trials with probability $\phi(i) = \frac{r_i}{\sum_{i=1}^N r_i}$. If it wins in trial m , $1 \leq m \leq (P - 1)$, it sets $p_i = m$. If it runs $P - 1$ trials without success, then it sets $p_i = P$.

Consider a single time slot. If only a single node wins in the first Bernoulli trial, then it sends a beacon with priority 1, and grabs the channel and transmit its data packet. This occurs with probability $\sum_{i=1}^N \phi(i)(1 - \phi(i))^{N-1}$. Now, if more than one node wins the first Bernoulli trial, then they simultaneously transmit beacons and enter the CR phase. This gives an equal opportunity to such nodes (as all are in one hop), so the medium access probabilities among such nodes do not follow the required ratios. However, note that the probabilities $\phi(i)$ are normalized to 1. Hence, on average, the number of nodes winning the first (or any) Bernoulli trial is close to 1. Hence the probability of medium access for a node i approximates the Bernoulli probability $\phi(i)$. Since the Bernoulli probabilities are normalized based on the required proportional rate allocation, the medium access probability of nodes winning the first Bernoulli trial approximates the required ratios.

Now consider the case that no node wins the first Bernoulli trial – this happens with probability $\prod_{i=1}^N (1 - \phi(i))$. Such nodes try the Bernoulli trial again and by the same logic as above their access pattern also approximates the required ratios. This process is repeated for $P - 1$ trials. All nodes which do not win in any of the $P - 1$ Bernoulli trials transmit beacons with priority P . Clearly the medium access pattern between this set of nodes will not follow the required ratio, but will be equally distributed. However, the probability of all nodes not winning their $P - 1$ Bernoulli trials is given by $(\prod_{i=1}^N (1 - \phi(i)))^{P-1}$. This probability quickly falls down to 0 with increase in P . Hence the more priority levels becomes available to the system, the closer the approximation to the desired ratios gets.

Implementation Specifics: PMAC and Siren are implemented with 6 priority levels ($P = 6$). However, DWOP supports infinite priority levels. This has an implication as to how close PMAC and Siren can approximate the desired rate allocation ratios when compared to DWOP. While DWOP will always show closer approximation to the desired

rate allocation, we shall see in Section 5.6.5 that 6 priority levels is enough for PMAC and 802.11e to have reasonable accuracy.

5.5.2 EDF Scheduling in Multi-hop Networks

EDF scheduling can be implemented as described by Kanodia et al. [51]. Every packet at its source is stamped with a TTL (time-to-live) which is its desired end-to-end delay bound. At every hop, just before the transmission of this packet, its TTL is decremented by the amount of time spent in the queue. The TTL value is directly mapped to a priority level at each hop. This allows nodes with backlogged packets with a very low TTL to access the channel quickly and flush such packets. More precisely, we assign the priority p_i of an outgoing packet i as follows. Let D_{max} be the maximum deadline value that a real time packet can have and P be the number of priority levels. Suppose that a packet i has a deadline d_i . Then p_i is set to d_i/Δ where $\Delta = D_{max}/P$.

Implementation Specifics: In our EDF experiments in Section 5.6.2, all realtime flows are stamped with an initial TTL of $D_{max} = 500\text{ms}$. For Siren and PMAC implemented with 6 priority levels, this corresponds to $\Delta = 83.33$. Since DWOP supports infinite priority levels, the TTL value is itself used as the priority and nodes order packet transmissions based on the TTL of their interfering neighbors.

5.5.3 DiffQ Congestion Control

The DiffQ algorithm is implemented as described in the previous chapter. We describe it briefly. Each node maintains a per-destination queue. The queue size for a destination is piggybacked in every transmission, hence disseminating this information to its neighbors. Intermediate nodes select the next destination to schedule based on the *queue differential* for all destinations. The destination with the maximum queue differential is scheduled first, and the HOL (head of line) packet from its queue is sent down to the MAC. The MAC then transmits the packet with a priority proportional to the queue differential, higher the queue differential, higher the priority. This form of scheduling creates a back-pressure starting at congested nodes moving toward the source, allowing sources to regulate their transmission just by observing their local queue size.

Implementation Specifics: Due to limited sensor memory, in our experiments we use a per-destination queue size of 3 packets. Hence the queue differential always lies in the

Figure 5.6: Above figure shows the 35 node mote testbed distributed across one floor of the MRC building. Lines between nodes indicate connectivity achieved with data transmission power of -5 dBm.

interval $[-2, 3]$. This allows us to use a one-to-one linear mapping of the queue differential $[-2, 3]$ to the packet priority $[0, 5]$ without any quantization error. Hence all three MACs – Siren, PMAC and DWOP use the mapped packet priority directly for priority resolution.

5.6 Testbed Results

5.6.1 Experimental Methodology

Our sensor network testbed consists of 30 MicaZ [9] sensor nodes distributed in labs and student offices in our computer science building, as shown in Figure 5.6. The lines connecting the nodes indicate wireless links. A single MicaZ node is shown in Figure 5.7.

We implement Siren, PMAC and DWOP as described in Section 5.4. As the base case of a MAC protocol which does not support priority resolution, we consider B-MAC [78], which is the default MAC protocol for MicaZ sensor nodes under the TinyOS environment. Medium access in B-MAC is similar to that in 802.11, with the exception that it does not

Figure 5.7: A single MicaZ node. Each node is equipped with 4K RAM, 128KB flash ROM, a 2.4 GHz ZigBee compatible radio capable of a raw data rate of 250 Kbps, and an 8 MHz CPU.

perform exponential backoffs. Just before packet transmission, a node performs a backoff over a fixed window. At the end of the backoff, the node transmits the packet if the channel is idle, else it takes a backoff again. We will refer to this MAC as *CSMA-DATA/ACK* for the rest of this section. For a fair comparison, we also compare with B-MAC with RTS/CTS as an additional performance baseline. We denote it as *CSMA-RTS/CTS*.

For the multi-hop results, each experiment consists of randomly selected source nodes transmitting packets to randomly chosen destination nodes. We repeat the experiment for a number of runs with different (source, destination) pairs and report the distribution of the metric of concern.

We now look at how each MAC performs for different fairness policies.

5.6.2 EDF Scheduling

We implement EDF scheduling as described in Section 5.5.2. Our experiment is as follows. We generate 10 concurrent flows on the testbed, with sources and destinations for the flows generated randomly. The source node of each flow stamps each outgoing packet with a TTL of 500ms. All flows are transmitted at 3 Kbps. The experiment is repeated 15 times. We report the distribution of the average per-flow delay for all runs in Figure 5.8. Figure 5.9 shows the fraction of the runs for each MAC in which the average per-flow delay

is less than the EDF deadline of 500ms.

Figure 5.8: Distribution of delays in the EDF scheduling experiment.

Siren is able to maintain a delay less than the desired TTL for 80% of the runs, while in the case of DWOP and PMAC, only 30% and 15% of the runs fulfill the EDF deadline. The poor performance of DWOP is due to inconsistent cache information causing frequent deadlocks. This problem comes up quite frequently on our testbed due to long, lossy links common in wireless networks. To investigate this problem further, we look at one of the scenarios closely in Figure 5.11. In the figure, node 3 is the source of a realtime flow getting starved. This was because 3 kept deferring its own transmissions due to RTS and CTS messages from node 32, which is quite some distance away. The small size of RTS and CTS messages (18 bytes) allowed these packets to be received intermittently at 3. However, the consequent DATA packet signaling the end of the high priority transmission is frequently lost at node 3, due to its large size (384 bytes). This leaves 3 deadlocked. The priority information from neighboring node 8 reaches 3 without much problem since the loss rate from 8 to 3 is 5%. Our modification to DWOP to deal with this issue (Section 5.4.6) does not help much as it simply reduces the duration of this deadlock. We also run the original DWOP protocol, referred to in Figure 5.9 as “DWOP - Deadlock”. The long delays for “DWOP - Deadlock” show the degree by which the original DWOP gets deadlocked without a deadlock timeout feature.

Figure 5.9: Number of runs for each MAC where the average per-flow delay is below the EDF deadline.

An interesting observation here is that a beacon based scheme like Siren does not suffer from the same problem in the presence of lossy channel. This is because HOL priorities are advertised through beacons which are short packets, and hence do not incur high loss rate. In addition, it is not necessary for a node to receive the beacon fully and correctly to determine that a neighbor has a high priority packet pending. As long as it detects a busy channel, it will defer access.

At this time, we tested an important component of Siren – the high power beacons (Section 5.4.5). We ran the same experiment with Siren with the beacon transmission power set to be the same as data transmission power, denoted by *Siren - Normal Power Beacons*. Although this seems to slightly reduce the average per-flow delay compared to Siren with high power beacons, the aggregate throughput per run is also reduced. This is because the normal power beacons cannot preserve priority across two hops, causing hidden terminal collisions, reducing the throughput at the destination. PMAC’s differentiated backoffs do not protect against hidden terminals, which causes to its poor performance.

5.6.3 Static Priority Scheduling

We consider an environment where two classes of applications coexist – high priority (HP) and low priority (LP). Our fairness policy is to allow the HP flows access to the

Aggregate Throughput for EDF Runs Within TTL Deadline

Figure 5.10: Aggregate throughput for all runs where the average per-flow delay is below the EDF deadline.

channel as much as possible, deferring accesses of LP flows as long as HP flows are present. This is achieved by setting the priority of all HP flows to be higher (set to 2) than that of all the LP flows (set to 3). Our experiment consists of 5 HP flows which transmit at a constant bit rate of 3 Kbps, and 5 LP flows transmit at a constant bit rate of 5 Kbps. The sources and destinations of these 10 flows are selected randomly. This experiment is repeated 15 times with different source and destination pairs. We report the throughput of each class normalized with the source sending rate in Figures 5.12 and 5.13.

Figure 5.12 shows that both DWOP and Siren are able to provide full access to HP flows with DWOP performing better. However, DWOP does this at the expense of throughput for LP flows. From Figure 5.13, we can see that LP flows in DWOP get about 50% of the throughput achieved in Siren. This is again due to the inconsistent priority database information caused by lossy links. Note that for DWOP, the HP flows do not affect each other, since they are all set to the same high priority level, and hence the HP flows do not get into deadlock. However, the LP flows may experience deadlock if there is a high priority flow within two-hops. The deadlock happens when a HP packet blocking LP packets has actually finished transmission, but the information about finishing the transmission is lost while the channel is free. Since the DATA packet is much larger than RTS and CTS packets and is subject to higher packet loss, its transmission may not

Figure 5.11: Above figure illustrates the DWOP problem in lossy networks. Flows $33 \rightarrow 32 \rightarrow 31$ and $7 \rightarrow 8 \rightarrow 14$ are real-time flows from one of the runs in the EDF scheduling experiment. Node 3 is the source of a flow which was observed to be starved. We monitored node 3 and found that it was continuously deferring its own transmission due to perceived high priority packets in its neighborhood. The figure on the right represents the normalized fraction of time spent in deferring attributed to the constituent flow segments. About 75% of the time 3 defers due to the RTS heard from node 32 ($32 \rightarrow 31$) or the CTS heard from 32 ($33 \rightarrow 32$).

register among neighboring nodes, which leaves them deadlocked, as seen in Figure 5.11 earlier. Figure 5.14 shows the fraction of LP flows in all the runs that get starved (0 throughput).

5.6.4 DiffQ Performance

We now look at how DiffQ performs with different MAC protocols. The DiffQ cross-layer (CLO) framework consisted of three components, source rate control, per-flow queuing and MAC scheduling. Instead of directly comparing the performance of the framework under each of DWOP, Siren and PMAC, we take the approach of incremental performance evaluation of each component.

We start with the baseline, which is a framework without any congestion control or scheduling. We select 8 sources randomly which transmit packets as fast as possible toward 8 randomly selected destinations. We refer to this as *CSMA-DATA/ACK-Full-Rate*. We then add the source rate control and per-flow queuing components to the framework, but leave out the MAC scheduling component. We refer to this result as *CLO-CSMA-XX* – the suffix *XX* denoting whether the underlying MAC uses RTS/CTS or just DATA/ACK.

Figure 5.12: Average normalized per-flow throughput (with 90%) confidence intervals of high-priority flows in the static priority scheduling experiment. Note that the MACs are presented in the increasing order of normalized throughput.

Finally, we test the full framework under Siren, DWOP and PMAC, denoted respectively as *CLO-Siren*, *CLO-DWOP*, and *CLO-PMAC*. The performance metric we report is the aggregate utility ($\sum \log(x)$), where x is the throughput obtained by a flow. We report the distribution of the aggregate utility in Figure 5.15.

Note that a low value of the aggregate utility indicates severe starvation for some flows, although the aggregate throughput may be high as seen in Figure 5.16. The main causes of flow starvation in conventional wireless networks are the inherent MAC-layer unfairness observed in CSMA-based MAC protocols [86] and the lack of effective congestion control which causes queue overflows at intermediate nodes [47, 99]. Both these factors cause the low aggregate utility in the case of (*CSMA-DATA-ACK-Full Rate*). By adding source rate control and per-flow queuing starvation due to queue overflows is mitigated to some extent as can be seen in the case of *CLO-CSMA-DATA/ACK* and *CLO-CSMA-RTS/CTS*. The large improvement seen in *CLO-CSMA-RTS/CTS* indicates that a significant factor of the flow starvation is due to hidden terminal scenarios on our testbed. The disappointingly low utility in *CLO-PMAC* compared to *CLO-CSMA-RTS/CTS* is due to the vulnerability

Figure 5.13: Average normalized per-flow throughput (with 90% confidence intervals) of low priority flows in the static priority scheduling experiment. Note that the MACs are presented in the increasing order of normalized throughput.

of PMAC to hidden terminals.

Finally *CLO-Siren* and *CLO-DWOP* both show the maximum value of the aggregate utility among all the tested cases. This shows that even after congestion control and per-flow queuing, there is still exists some scope of improvement by tuning the MAC scheduling component. It is interesting that both Siren and DWOP provide more or less equivalent performance for the proportional fairness framework. An interesting question here is why CLO-DWOP does not suffer from the deadlock situations seen in the earlier experiments. This is because in CLO, a deadlocked flow immediately begins to be backlogged, thus increasing its queue differential. Eventually, its priority becomes equal to the highest priority, and it is able to flush out its packets. This prevents complete starvation of any particular flow. However in the case of static priority scheduling, the deadlocked flow's priority never changes, and in EDF scheduling, a deadlocked flow's priority (TTL) changes very slowly before its priority becomes large enough to transmit. Clearly, if used in wireless multihop environments where deadlock situations abound, with a fairness policy where the priority of a flow is does not change fast enough in response to deadlock situations, DWOP will perform poorly.

Figure 5.14: Fraction of low priority flows getting starved (0 throughput) in the static priority scheduling experiment.

5.6.5 Proportional Rate Allocation

We test the MAC protocols in approximating the rate allocation by running the following test. We pick 4 and 6 nodes of our testbed in sequence in such a way that they are all within radio range of each other. These nodes then transmit packets to a common sink node by the proportional rate allocation algorithm described in 5.5.1. The sink node records the ratio of the throughputs obtained from each node which we report in Figure 5.17. We use Siren with 6 priority levels for this experiment.

DWOP represents the faithful implementation of our rate allocation algorithm, since it is not constrained by number of priority levels. As a result, its rate allocation is closest to the desired ratios. Siren with just 6 priority levels shows performance close to that of DWOP. The performance of PMAC was similar to that of Siren for this experiment because all nodes are within radio range of each other so there are no hidden terminals.

5.7 Conclusion

We evaluate different MAC protocols for their effectiveness in implementing different commonly cited fairness policies. We find that each MAC has its strengths and

Figure 5.15: Distribution of aggregate utility $\sum \log(x)$ for CLO experiment.

weaknesses. Differential backoff-based schemes are easy to implement (do not need special constructs such as RTS/CTS or time synchronization) and work well for single hop fairness policies such as proportional rate allocation/temporal fairness but are vulnerable to hidden terminals under multihop wireless environments. Beacon-based schemes are robust to hidden terminals and lossy wireless environments, but are difficult to implement due to the requirement of time synchronization. Scheduling-based schemes provide infinite priority levels with low overhead, but are vulnerable to loss of priority information, often leading to deadlocks. Deadlocks most often occur when the fairness policy assigns priorities which do not change with time, e.g. static priority scheduling where some classes of flows always have higher priorities than others. If the fairness policy is such that it automatically compensates for deadlock by increasing the priority of a deadlocked node, then the effects of deadlock are mitigated. Both differential backoff-based schemes and beacon-based schemes support discrete priority levels. This leads to some quantization error when used to enforce some fairness policies, when compared to scheduling-based schemes which have infinite priority levels.

Clearly, Siren and DWOP show the most promise as a MAC for DiffQ in wireless sensor environments. However, both Siren and DWOP are not suitable for current 802.11 radios, the core reason being the lack of flexibility in 802.11 radios compared to sensor radios.

Figure 5.16: Comparison of per-flow throughput of baseline CSMA with CLO-Siren for one specific run. Note that CSMA’s aggregate throughput is higher, but CLO-Siren gets better utility since no flow is starved.

In 802.11 radios, for efficiency reasons, significant amount of functionality is encoded into firmware and is not amenable to change, e.g. CTS and ACK packets are transmitted by the firmware immediately on the reception of RTS and data packets respectively due to packet timing concerns. Hence in order to implement a scheduling protocol like DWOP which not only changes the format of CTS/ACK packets (by piggy-backing priority information) but also whether they should be transmitted at all, one would need to modify the firmware itself. Also, once a packet goes from the MAC driver into the 802.11 firmware, the exact moment it is transmitted into the air is controlled by the firmware, and not the MAC driver. Hence it is not possible to control the packet timing at the granularity required for a beaconing scheme like Siren.

Because of this, for 802.11 radios, we fall back on using a differentiated backoff-based scheme similar to 802.11e/PMAC. We describe the full implementation details of DiffQ with an 802.11e-variant in the next chapter.

Figure 5.17: Proportional rate allocations for DWOP and Siren compared with the ideal rate allocations for 4 and 6 sources.

Chapter 6

DiffQ - Architecture and Implementation

In this chapter, we describe the overall architecture and implementation of the DiffQ congestion control protocol in the Linux networking stack.

6.1 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). DiffQ uses the routing support of IP for single path routing while using source routing for multi-path routing in which case, DiffQ bypasses IP routing.

To implement the functions of DiffQ over IP, we need various mechanisms (1) to capture, process and re-inject all IP packets before or after routing in the kernel, (2) to control the MAC priority of each packet transmitted by MAC, and (3) to provide DiffQ information to the transport layer (e.g., queue size information for source rate control). DiffQ uses the Linux Netfilter module to implement the first mechanism, and uses the TOS field of IP header to specify the priority of each packet. The MadWiFi driver is modified to read this field of each packet to get the priority information of the packet and put the packet into appropriate priority queues. The driver outputs packets into the air interface in the order of priority using the queues.

Figure 6.1: Linux Netfilter hooks; DiffQ uses these hooks to capture packets from IP and reinject processed packets into IP.

6.2 Interface into IP Layer

We utilize the Netfilter mechanisms of Linux to capture IP packets for DiffQ processing. The Netfilter API provides various hooks throughout the networking stack where packets can be captured from the stack. Note that capturing does not incur data copy overhead as it simply copies pointers. Any module interested in packet inspection can tap into these hooks by specifying the type of the hook and the handling function. When a packet arrives at a hook the packet is captured from the stack and passed to the hook handling function. There can be several handling functions at the same hook in which case the packet is passed to each of them sequentially starting with the handling function having the highest priority (the priority is specified while registering the hook). The packets can be modified and returned to the stack, dropped from the stack, or queued for re-injection later. The various positions in the stack where the hooks are available are shown in Figure 6.1.

Each packet in DiffQ belongs to a flow which is uniquely identified by four values - source IP address, source port, destination IP address and destination port. Flows are unidirectional. Hence, a TCP connection sends data on one flow and receives acknowledgment on another flow. Each node can have one or more air interfaces, at least one running in the ad hoc mode (denoted *ath0*) for transmitting to and receiving from other mesh nodes, and another running in the infrastructure mode (denoted *ath1*) for receiving from and transmitting to mobile devices, e.g., laptops and PDA. At each node, packets can be generated from the application running in this node or mobile nodes attached to *ath1*, or

being forwarded by its neighboring nodes.

To handle various types of traffic we use the following hooks. We use `LOCALOUT` to trap packets which are generated from the node. At this hook level we know the destination interface of the packet. The following sequence of actions is taken on the packets trapped at this hook: if the packet is going out to `ath1` then the packet is returned back to the IP stack. If the packet is going out of `ath0` interface then we attach the packet with a DiffQ header discussed later. Then the packet is queued. This makes the network stack pass the packet to a queue handler which enqueues the packet in the DiffQ destination queues.

We use `LOCALIN` to trap packets destined for this node. We check if the packet has a DiffQ header and if the header is present then it is removed. The packet is then returned to the stack. This makes the stack deliver the packet to the transport layer (UDP or TCP). This way, the transport level protocols do not have to be aware of the presence of DiffQ.

We use the `PREROUTING` hook to trap packets received from a network interface. If the packet has a DiffQ header, then this packet is part of a DiffQ flow and hence we enqueue the packet into its destination queue. If the header is not present then the packet is not a DiffQ flow and is returned back to the stack – this enables our system to handle other IP traffic.

6.3 DiffQ header format

Each DiffQ packet is appended with a DiffQ header containing various fields used by the DiffQ algorithm. We attach the header at the end of data so that the IP layer treats the packet just like an IP packet. This allows the transport header not to be modified since we remove the DiffQ header before passing it to transport layer on the receiver side. The IP header is modified to reflect the new size of the packet. Since we change the size of the packet, we re-compute the checksum before injection to the network stack and delivery to the transport.

The DiffQ header has the following fields (illustrated in Figure 6.2): The type field contains the type of the flow that this packet belongs to, e.g. we use this field to recognize whether a flow is a source-routed flow like `MORE` or not. The flowid is the source assigned id for the current flow. Qlen is the length of the packet queue of the destination of the flow. This field is updated by each transmitting node according to the DiffQ algorithm.

Timestamp is the timestamp at the source node when this packet was transmitted. Sequence number is used for testing purpose to trace packets.

Figure 6.2: DiffQ Architecture and Packet Header Format - DiffQ sits on top of IP and provides congestion control services to upper layer transport modules. It also controls the MAC priority of packets for scheduling and performs source rate controls for the transport flows (for support of TCP, it disables TCP's congestion control).

6.4 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

Table 6.1: 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

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.

6.5 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 Chapter 4. DiffQ uses a modified MADWiFi-NG driver [8] 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 6.1) 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.

6.6 Integration with MORE

DiffQ provides a congestion control service to opportunistic routing applications like MORE [19] as shown in Figure 6.3. The MORE application consists of two components, a source and a forwarder. The MORE source generates new linear encoded packets at the rate dictated by the DiffQ source rate control algorithm. Each MORE encoded packet has a MORE header which contains the list of prospective forwarders. This list is used by DiffQ to calculate the queue differential as described in Section 4.2.

Each MORE packet is then enqueued in the per-destination queues in the DiffQ scheduler inside the Linux kernel. The DiffQ scheduler services the destination with the

Figure 6.3: DiffQ integration with MORE - DiffQ provides congestion control service to MORE which sits in the application layer.

maximum queue differential and pushes the HOL MORE packet for that destination down to the MAC, stamped with the appropriate priority. Finally the MAC transmits the MORE packet as an ordinary broadcast packet with the assigned priority.

The MORE forwarder listens on an assigned port for MORE packets. Since MORE packets are assigned a broadcast destination address, MORE packets transmitted by a node will be received by the MORE forwarder on all the neighbors of the node. If a particular neighbor is specified as a forwarder in the MORE header, it will accept the packet, else it will be dropped. The MORE forwarder generates new packets for a flow based on MORE's *credit factor* algorithm. Such packets are not rate controlled, and hence go directly to the DiffQ scheduler.

Chapter 7

DiffQ - Experimental Results

In this section we perform a comprehensive evaluation of DiffQ in single-path and opportunistic multi-path scenarios.

7.1 WiSeNet Testbed

The WiSeNet testbed consists of 46 wireless nodes distributed over three floors in a building of about 100,000 sq ft space. This is one of the largest testbeds both in terms of number of nodes and coverage area. One floor of this testbed with 20 nodes is shown in Figure 7.1. Nodes are PCs with 128MB RAM and 266 MHz processor. Figure 7.2 show one such node. Each node is equipped with two Atheros-based 802.11 a/b/g wireless interfaces (AR 5213/5112) connected to omni-directional 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 [4] of the OLSR [49] routing protocol which uses ETX [24] to generate routes.

7.2 Experimental Setup

We compare the following single path congestion control algorithms:

- **TCP:** We use TCP-SACK [70] as the baseline.
- **TCP-FeW:** TCP-FeW [72] is a variant of TCP proposed for wireless networks. It addresses the congestion window overestimation problem of TCP-Reno style algorithms

Figure 7.1: One floor of our testbed with 20 nodes. The full testbed has 46 nodes over three floors.

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 [30] 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.
- **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.

Figure 7.2: Soekris mesh nodes used on WiSeNet.

The main strength of DiffQ lies in its ability to support flows using source-based multi-path routing including opportunistic routing and intra-flow network coding where packets of the same flow are encoded together. To demonstrate its effectiveness we use the following algorithms.

- **MORE:** MORE [19] combines network coding with opportunistic routing. We obtained its source code from the authors and ported it over native Linux. It is originally implemented on top of Click [6].
- **DiffQ-MORE:** We apply DiffQ congestion control to MORE as described in Section 4.2. In this implementation the MORE source transmits encoded packets at the DiffQ regulated source rate.

MORE is implemented in the application layer, and MORE packets are delivered to the application at every hop for coding. This particular implementation strategy incurs significantly more performance overheads (context switching, system calls and coding) than TCP implemented in the native Linux kernel. Consequently, the performance of MORE is lower than TCP as shown in Table 7.1. In this thesis, we do not study how much performance enhancement MORE makes over single-path routing as the comparison with TCP is unfair due to the implementation-specific overheads cited above. Instead, we focus on evaluating congestion control performance when MORE is combined with DiffQ. If MORE is implemented in the kernel, it would have achieved much better performance. Since our purpose is to implement congestion control for opportunistic routing like MORE, the current user-space implementation of MORE is sufficient to serve our purpose.

Table 7.1: Throughput of MORE and TCP

	1-hop	2-hop	3-hop	4-hop
MORE (Mbps)	1.87	1.25	0.94	0.62
TCP (Mbps)	4.25	2.29	1.45	1.25

Our experiment is setup as follows. We construct each experimental scenario by fixing the number of concurrently running flows of a protocol and then randomly selecting the source and destination nodes of each flow. For a given number of flows, we construct 20 scenarios with different configurations of source and destination pairs. For a given scenario, we repeat the experiment with different protocols. We categorize the performance statistics of the experiment runs in terms of the number of flows used in a scenario and by a choice of protocol used for congestion control. For each run, all sources transmit packets toward their destination at the rate decided by the underlying congestion control algorithm. The packet size for all the above algorithms is set to 1400 bytes. MORE and DiffQ-MORE use a batch size of 32 packets. 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 competing flows and average per-flow throughput r_i , $1 \leq r \leq N$),

- **Fairness index per run** [7] 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 within a run.

- **Log utility per run** is given by $\sum_{i=1}^N \log(r_i)$. It measures the degree of proportional fairness [28]. 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.

7.3 Multi-path congestion control

The original MORE source [19] broadcasts as fast as it can toward the destination. We saw earlier in Chapter 3, how this can rapidly lead to congestion collapse, with MORE starving almost 50% of its flows. DiffQ-MORE, however restricts the source rate based on its multi-path backpressure algorithm. The effect of congestion control can be readily seen in Figure 7.3. The experimental setup is the same as in Chapter 3 – 20 runs of 1,4,8,16 and 32 flows generated between random sources and destinations on the testbed.

Figure 7.3: CDF of per-flow average throughput for DiffQ-MORE.

When there is little or no congestion in the network, e.g. for the case with 1,4 flows,

MORE and DiffQ-MORE show similar performance. In fact, the single flow performance of MORE is slightly better than that of DiffQ-MORE. This can be seen in Figure 7.5 which shows the average per-run aggregate throughput. We attribute this to the fact that DiffQ is conservative in congestion control, it uses the *minimum* queue differential among all its MORE forwarders to decide its outgoing packet priority. This results in some reduction in capacity. As the number of flows in the system increase, MORE experiences a dramatic reduction in both average throughput as well as fairness, seen in Figures 7.4, 7.5 and 7.6. DiffQ is however able to maintain consistent performance, starving only 18% of the flows in the 32 flow scenario.

Figure 7.4: Average per-run fairness index of MORE and DiffQ-MORE

Figure 7.7 shows the normalized throughput difference of DiffQ-MORE and MORE for the 32 flows scenario – for each flow we calculate the difference of throughputs of DiffQ-MORE and MORE and divide it with the throughput of MORE. The x-axis shows the ETX of that path. A positive value indicates a higher throughput for DiffQ-MORE. As the path ETX increases, the throughput difference also increases. This is because MORE selects several forwarders for long lossy paths, and it is precisely such paths which face high contention and congestion. DiffQ-MORE effectively controls congestion among all forwarders for such paths, and hence delivers more throughput.

Figure 7.5: Average per-run aggregate throughput of MORE and DiffQ-MORE

7.4 Single-path congestion control

We run the same setup as in Chapter 3 with various single-path algorithms. Figure 7.8 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 3.2 for comparison). We also consider the 3-flow unfair scenario for TCP that we presented in Figure 3.4. Figure 7.9 shows the performance over DiffQ for the same scenario. The routing path for flow 1 is $20 \rightarrow 13 \rightarrow 12 \rightarrow 7 \rightarrow 6 \rightarrow 5$, flow 2 is $4 \rightarrow 8 \rightarrow 10$ and flow 3 is $18 \rightarrow 19$, on nodes in the Figure 7.1. 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.

Figure 7.10 compares the average per-run aggregate throughput of various reliable transfer algorithms under different numbers of concurrent flows. We find that the average 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. On the other hand, the fairness of DiffQ-TCP is much higher. Figures 7.11 and 7.12 compare the fairness index and log utility of these protocols respectively. The fairness index is about two times

Figure 7.6: Average per-run log utility of MORE and DiffQ-MORE

higher for DiffQ-TCP and the log utility is about 20% better. These results are manifested in their median throughput. Table 7.2 shows the ratio of average median per-flow throughput of DiffQ-TCP over that of TCP. When 32 concurrent flows are run on the testbed, the average median per-flow throughput of DiffQ-TCP is 8 times that of TCP.

Table 7.2: 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 7.13 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

Figure 7.7: Relative throughput difference between DiffQ-MORE and MORE with respect to path ETX.

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 [34]. 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 coordinations 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 Figures 7.14, 7.16 and 7.15. We also plot the performance of DiffQ-UDP for comparison. 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. Thus, use of differential backlog using prioritized channel access improves fairness as well as efficiency. This can be visualized from Figure 7.17 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 while TFRC-ECN can

Figure 7.8: The CDF of per-flow average throughput of DiffQ-TCP. It shows that DiffQ-TCP do not incur much flow starvation.

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.

7.5 Contribution of rate control, scheduling and MAC prioritization

At this point, a pertinent question about DiffQ would be: 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

Figure 7.9: The instantaneous throughput of three DiffQ-TCP flows for the same scenario described in Figure 3.4.

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 7.18 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.

Figure 7.10: Average per-run aggregate throughput for single path reliable algorithms

Figure 7.11: Average per-run fairness index for single path reliable algorithms

Figure 7.12: Average per-run logarithmic utility for single path reliable algorithms

Figure 7.13: Scatter plot of per-flow throughputs for the 32 flow scenarios.

Figure 7.14: Average per-run Jain's fairness index for single path congestion control algorithms (unreliable transport)

Figure 7.15: Average per-run log utility for single path congestion control algorithms (unreliable transport)

Figure 7.16: Average per-run aggregate throughput for single path congestion control algorithms (unreliable transport)

Figure 7.17: Distribution of throughput of each flow when run together.

Figure 7.18: Number of queue overflows in a 2 minute experiment with variants of DiffQ-UDP.

Chapter 8

Conclusion and Future Work

This thesis presents DiffQ – a new congestion control protocol for wireless multi-hop networks. DiffQ is flexible and scalable enough to be applied for congestion control of network flows using single and multi-path routing over many diverse paths in wireless networks. It does not put any restriction on the traffic patterns of flows and can be applied to general ad hoc networks. The protocol can mitigate the fairness problem commonly observed from existing congestion control algorithms like TCP under wireless multi-hop networks. We implemented DiffQ in the Linux kernel for support of UDP and TCP as well as opportunistic routing such as MORE. Given the recent interest in opportunistic routing in wireless networks, we feel that our work is timely. To the best of our knowledge, this work is the first systems work that implements backpressure-based hop-by-hop cross-layer optimization in IEEE 802.11 mesh networks.

Future work with DiffQ may lie in several directions:

- DiffQ, when combined with MORE can serve as a scalable congestion control service for multicast applications.
- DiffQ can support transport of real-time applications in the presence of best-effort traffic. Real-time packets can be scheduled based on EDF scheduling, as described in Section 5.5. Concurrent best-effort traffic can be assigned priorities lower than that for real-time packets, thus protecting real-time transmissions from concurrent best-effort transmissions.
- From the MAC evaluation in Chapter 5, it was clear that using 802.11e in multi-hop environments leaves DiffQ vulnerable to hidden terminals. With the use of software

defined radio, protocols like DWOP and Siren may be implemented and used as MACs for DiffQ.

Bibliography

- [1] CC1000 Low Power FSK transceiver, Chipcon Corporation.
- [2] CC2420 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver.
- [3] E. Inc. Universal Software Radio Peripheral. <http://ettus.com>.
- [4] <http://olsr.org/>.
- [5] <http://pdos.csail.mit.edu/roofnet/>.
- [6] <http://www.read.cs.ucla.edu/click/>.
- [7] Jain, R. the art of computer systems performance analysis, first ed. wiley, 1991.
- [8] MadWifi <http://madwifi.org/>.
- [9] XBOW MICAz Mote Specifications, <http://www.xbow.com/>.
- [10] ZigBee Alliance, IEEE 802.15.4, ZigBee standard, <http://www.zigbee.org/>.
- [11] Radio Equipment and Systems (RES); High Performance Radio Local Area Network (HIPERLAN); Functional Specifications, ETSI, 1995.
- [12] The impact of multihop wireless channel on tcp performance. *IEEE Transactions on Mobile Computing*, 4(2):209–221, 2005. Student Member-Zhenghua Fu and Student Member-Haiyun Luo and Student Member-Petros Zerfos and Member-Songwu Lu and Senior Member-Lixia Zhang and Fellow-Mario Gerla.
- [13] IEEE WG, Draft Supplement to Standard for Telecommunications and Information Exchange between Systems-LAN/MAN Specific Requirements- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications:

Medium Access Control (MAC), Enhancements for Quality of Service (QoS), 802.11e Draft 4.1, February, 2003.

- [14] I. Aad and C. Castelluccia. Differentiation mechanisms for IEEE 802.11. In *INFOCOM '01*.
- [15] 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.
- [16] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 31–42, New York, NY, USA, 2005. ACM.
- [17] S. Biswas and R. Morris. Exor: opportunistic multi-hop routing for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 35(4):133–144, 2005.
- [18] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang. Tcp westwood: end-to-end congestion control for wired/wireless networks. *Wirel. Netw.*, 8(5):467–479, 2002.
- [19] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. *SIGCOMM Comput. Commun. Rev.*, 37(4):169–180, 2007.
- [20] K. Chen, K. Nahrstedt, and N. Vaidya. The utility of explicit rate-based flow control in mobile ad hoc networks. In *WCNC'04*.
- [21] L. Chen, S. Low, M. Chiang, and J. Doyle. Optimal cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *INFOCOM '06*.
- [22] 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*.
- [23] S. H. L. Cheng Jin, David X. Wei. Fast tcp: Motivation, architecture, algorithms, performance. In *INFOCOM'04*.

- [24] 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: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.
- [25] B. Crow, I. Widjaja, J. G. Kim, and P. Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications Magazine*, 35(9):116–126, 1997.
- [26] D. DeLucia and K. Obraczka. Multicast feedback suppression using representatives. In *INFOCOM'97*.
- [27] A. Eryilmaz and D. S. Lun. Control for inter-session network coding. In *Proceedings of the Workshop on Network Coding, Theory and Applications (NetCod)*, 2007.
- [28] A. M. F. P. Kelly and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, pages 237–252, 1998.
- [29] S. Floyd. Tcp and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.
- [30] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. *SIGCOMM Comput. Commun. Rev.*, 30(4):43–56, 2000.
- [31] C. Fu and S. Liew. Tcp veno: Tcp enhancement for transmission over wireless access, 2003.
- [32] Z. Fu, B. Greenstein, X. Meng, and S. Lu. Design and implementation of a tcpfriendly transport protocol for adhoc wireless networks. In *ICNP'02*.
- [33] Z. Fu, X. Meng, and S. Lu. A transport protocol for supporting multimedia streaming in mobile ad hoc networks. *IEEE JSAC*, 21(10):1615–1626, 2004.
- [34] Z. Fu, P. Zerfos, k Xu, H. Luo, S. Lu, L. Zhang, and M. Geda. On tcp performance in multihop wireless networks.
- [35] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on tcp throughput and loss. In *INFOCOM'03*.

- [36] V. Gambiroza, B. Sadeghi, and E. W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 287–301, New York, NY, USA, 2004. ACM.
- [37] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM.
- [38] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: single-path routing vs. multi-path routing. In *INFOCOM '04*.
- [39] M. Garetto, J. Shi, and E. W. Knightly. Modeling media access in embedded two-flow topologies of multi-hop wireless networks. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 200–214, New York, NY, USA, 2005. ACM.
- [40] Y. Ge, J. Hou, and S. Choi. An analytic study of tuning system parameters in ieee 802.11e enhanced distributed channel access. *Computer Networks*, 2005.
- [41] M. Gerla, K. Tang, and R. Bagrodia. Tcp performance in wireless multi-hop networks. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 41, Washington, DC, USA, 1999. IEEE Computer Society.
- [42] D. Hadaller, S. Keshav, and T. Brecht. Mv-max: improving wireless infrastructure access for multi-vehicular communication. In *CHANTS '06*.
- [43] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *INFOCOM '03*.
- [44] M. Heusse, F. Rousseau, R. Guillier, and A. Duda. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans. *SIGCOMM Comput. Commun. Rev.*, 35(4):121–132, 2005.
- [45] G. Holland and N. Vaidya. Analysis of tcp performance over mobile ad hoc networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference*

- on *Mobile computing and networking*, pages 219–230, New York, NY, USA, 1999. ACM.
- [46] Y. Huang, M. Ghaderi, D. Towsley, and W. Gong. Tcp performance in coded wireless mesh networks.
 - [47] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147, New York, NY, USA, 2004. ACM.
 - [48] C. C. Imad Aad. Differentiation mechanisms for ieee 802.11. In *INFOCOM'01*.
 - [49] P. Jacquet, P. Mhlehler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol. In *INMIC'01*.
 - [50] E.-S. Jung and N. H. Vaidya. A power control mac protocol for ad hoc networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 36–47, New York, NY, USA, 2002. ACM.
 - [51] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed multi-hop scheduling and medium access with delay and throughput constraints. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 200–209, New York, NY, USA, 2001. ACM.
 - [52] V. Kanodia, A. Sabharwal, B. Sadeghi, and E. Knightly. Ordered packet scheduling in wireless ad hoc networks: mechanisms and performance analysis. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 58–70, New York, NY, USA, 2002. ACM.
 - [53] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.
 - [54] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.
 - [55] P. Key, L. Massoulie, and D. Towsley. Path selection and multipath congestion control. In *INFOCOM'07*.

- [56] A. Khreishah, C.-C. Wang, and N. B. Shroff. Optimization based rate control for communication networks with inter-session network coding. In *INFOCOM '08*.
- [57] F. Klemm, Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi. Improving tcp performance in ad hoc networks using signal strength based link management. *Ad Hoc Networks - Ad Hoc Networking for Pervasive Systems*, 3(2):175–191, 2005.
- [58] A. E. L. Tassiulas. 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.
- [59] S. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks, 2001.
- [60] D. J. Leith and P. Clifford. Using the 802.11e edcf to achieve tcp upload fairness over wlan links. In *WIOPT '05*.
- [61] F. Licandro and G. Schembra. Wireless mesh networks to support video surveillance: architecture, protocol, and implementation issues. *EURASIP J. Wirel. Commun. Netw.*, 2007(1):34–34, 2007.
- [62] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He. Atpc: adaptive transmission power control for wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 223–236, New York, NY, USA, 2006. ACM.
- [63] 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.
- [64] 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.
- [65] X. Lin and N. B. Shroff. Utility maximization for communication networks with multi-path routing. *IEEE Transactions on Automatic Control*, 51(5):766–781, 2006.
- [66] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE J-SAC*, 19(7):1300–1315, 2001.

- [67] 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.
- [68] M. K. Marina and S. R. Das. Ad hoc on-demand multipath distance vector routing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(3):92–93, 2002.
- [69] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM.
- [70] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgement options.
- [71] J. Monks, V. Bharghavan, and W. mei W. Hwu. A power controlled multiple access protocol for wireless packet networks. In *INFOCOM '01*.
- [72] K. Nahm, A. Helmy, and C.-C. J. Kuo. Tcp over multihop 802.11 networks: issues and performance enhancement. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 277–287, New York, NY, USA, 2005. ACM.
- [73] T. Nandagopal, T.-E. Kim, X. Gao, and V. Bharghavan. Achieving mac layer fairness in wireless packet networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 87–98, New York, NY, USA, 2000. ACM.
- [74] A. Nasipuri and S. Das. Demand multipath routing for mobile ad hoc networks, 1999.
- [75] M. J. Neely. Energy optimal control for time varying wireless networks. In *INFOCOM'05*.
- [76] M. J. Neely. Joint congestion control and media access control design for ad hoc wireless networks. In *INFOCOM'05*.
- [77] M. J. Neely. Optimal backpressure routing for wireless networks with multi-receiver diversity. In *CISS'06 (invited paper)*.

- [78] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM.
- [79] L. Popa, C. Raiciu, I. Stoica, and D. Rosenblum. Reducing congestion effects in wireless networks by multipath routing. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 96–105, Washington, DC, USA, 2006. IEEE Computer Society.
- [80] 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*.
- [81] 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.
- [82] 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*.
- [83] I. Rhee, L. Xu, and S. Ha. Cubic for fast long-distance networks, ietf, internet draft, 2007.
- [84] V. Sharma, S. Kalyanaraman, K. Kar, K. K. Ramakrishnan, and V. Subramanian. Mplot: A transport protocol exploiting multipath diversity using erasure codes. In *INFOCOM '08*.
- [85] J.-P. Sheu, C.-H. Liu, S.-L. Wu, and Y.-C. Tseng. A priority mac protocol to support real-time traffic in ad hoc networks. *Wirel. Netw.*, 10(1):61–69, 2004.
- [86] J. Shi, T. Salonidis, and E. W. Knightly. Starvation mitigation through multi-channel coordination in csma multi-hop wireless networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 214–225, New York, NY, USA, 2006. ACM.
- [87] V. A. Siris and G. Stamatakis. Optimal cwmin selection for achieving proportional fairness in multi-rate 802.11e wlans: test-bed implementation and evaluation. In *WiNTECH '06*.

- [88] J. L. Sobrinho and A. S. Krishnakumar. Quality-of-service in ad hoc carrier sense multiple access wireless networks. In *IEEE JSAC Vol.17, No.8, pp. 1353-1368, 1999*.
- [89] Y. Su and T. Gross. Wxcp: Explicit congestion control for wireless multi-hop networks. In *IWQoS'05*.
- [90] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226, New York, NY, USA, 2004. ACM.
- [91] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. K. Ramakrishnan. Lt-tcp: End-to-end framework to improve tcp performance over networks with lossy channels. In *IwQoS'05*.
- [92] I. Tinnirello and S. Choi. Temporal fairness provisioning in multi-rate contention-based 802.11e wlans. In *WOWMOM '05*.
- [93] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell. Coda: congestion detection and avoidance in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 266–279, New York, NY, USA, 2003. ACM.
- [94] 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*.
- [95] F. Wang and Y. Zhang. Improving tcp performance over mobile ad-hoc networks with out-of-order detection and response. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 217–225, New York, NY, USA, 2002. ACM.
- [96] W.-H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Perform. Eval.*, 52(2-3):119–132, 2003.
- [97] A. Warrior, L. Le, and I. Rhee. Cross-layer optimization made practical. In *Broad-nets'07 (Invited Paper)*.

- [98] S. Wu, Y. Tseng, and J. Sheu. Intelligent medium access for mobile ad hoc networks with busy tones and power control. *IEEE Journal on Selected Area in Communications*, 18(9):1647–57, 2000.
- [99] K. Xu, M. Gerla, L. Qi, and Y. Shu. Enhancing tcp fairness in ad hoc wireless networks using neighborhood red. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 16–28, New York, NY, USA, 2003. ACM.
- [100] X. Yang and N. Vaidya. Priority scheduling in wireless ad hoc networks. *Wirel. Netw.*, 12(3):273–286, 2006.
- [101] S. Yoo, J. Choi, J. Hwang, and C. Yoo. Eliminating the performance anomaly of 802.11b. volume 3421, pages 1055–1062, 2005.
- [102] H. Zhai, X. Chen, and Y. Fang. Rate-based transport control for mobile ad hoc networks. In *WCNC'05*.
- [103] G. Zhou, T. He, J. Stankovic, and T. Abdelzaher. Rid: radio interference detection in wireless sensor networks. In *INFOCOM '05*.