ABSTRACT

DEUSKAR, GAURISH CHANDRASHEKHAR. Packet Aggregation based Backpressure Scheduling in Multi-hop Wireless Networks. (Under the direction of Dr. Rudra Dutta.)

Multihop wireless networks face the fundamental problem of interference in a local neighborhood. Given this and the nature of the 802.11 DCF mechanism at the MAC layer with the current transport layer protocols like UDP and TCP, wireless multihop networks face problems of limited capacity and extreme unfairness across different traffic flows. Research studies of joint congestion control and backpressure based scheduling mechanisms have shown to improve the fairness across traffic flows and the aggregate throughput. Also the other major property of multihop wireless networks is that of the huge amount of MAC and PHY overheads. With this problem coupled with the fact that a huge percentage of internet traffic consists of small packet sizes, we get the opportunity to aggregate packets in order to reduce the MAC and PHY overheads thereby getting an increase in the throughput. In this thesis, we aim at performing packet aggregation at the IP layer, meaning that multiple IP packets are bundled into one MAC layer SDU. We do this in such a way so as to conform with the back-pressure based scheduling principles. Our aim is to obtain an increase in the aggregate throughput of multihop wireless networks without causing degradation in the fairness across flows and also to observe the effect on the average network delay and the buffer occupancy at intermediate nodes. We simulate our design in OPNET with various network topologies and also implement it in the Linux kernel running on a real testbed. Our results show that our scheme of packet aggregation based backpressure scheduling increases the aggregate network throughput with respect to the technique consisting of only backpressure based scheduling. Also the fairness of our scheme degrades only by a slight degree in some cases while is better in other cases with respect to its counterpart. We also observe an overall decrease in the average network delay and the average buffer occupancy on intermediate nodes.

Packet Aggregation based Backpressure Scheduling in Multi-hop Wireless Networks

by
Gaurish Chandrashekhkar Deuskar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Dr. George Rouskas

Dr. Injong Rhee

Dr. Rudra Dutta
Chair of Advisory Committee

DEDICATION

To my family and the memory of my late grandfather...

BIOGRAPHY

The author was born in a small town in India in 1986. After receiving bachelors degree in computer engineering from Mumbai university in 2007, the author enrolled in Masters of science in computer science program in North Carolina state university in 2007.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Rudra Dutta for advising me in this study. I am grateful to him for his valuable ideas and insights in various directions. I also thank him for helping me in domains other than my research like giving me more disk space for my simulations.

I would also like to thank Dr. George Rouskas and Dr. Injong Rhee for agreeing to be on my committee.

I would also like to thank my room mate Parth Pathak for his help in solving my doubts in this thesis.

Last but not the least, mere words are insufficient to thank my family members. It is only due to their hard efforts, encouragement and abysmal love that I got an opportunity to have a fruitful Masters education. I am forever indebted to them.

TABLE OF CONTENTS

List of Tables	vii
List of Figures	viii
Chapter 1 Introduction	1
1.1 Background and challenges	1
1.2 Motivation	2
1.3 Organization	3
Chapter 2 Related work	4
2.1 Work on backpressure scheduling and congestion control	4
2.2 Work on packet aggregation	6
Chapter 3 Packet aggregation with PDQ scheduling	8
3.1 Terms	8
3.1.1 Per Destination Queues (PDQ)	8
3.1.2 Urgency weight	8
3.1.3 Urgency Weight state (UW state)	8
3.1.4 MAC priorities	9
3.1.5 Packet aggregate	9
3.2 Methodology	10
3.2.1 Basic idea	10
3.2.2 Aggregation process	10
3.2.3 De-aggregation process	11
3.3 Example	11
3.4 Theoretical analysis	15
3.4.1 Terms	15
3.4.2 Modelling average throughput of a flow	17
3.4.3 Modelling the buffer process	19
Chapter 4 Experimental methodology: OPNET simulation	21
4.1 Requirements	21
4.2 OPNET basics	21
4.2.1 Node model	21
4.2.2 Process Model	22
4.3 UW header	23
4.4 Changes in the wlan process model	24
4.4.1 Event: Packet arrival from higher layer	24
4.4.2 Event: Packet transmission	24
4.4.3 Event: Packet arrival from physical layer	25

Chapter 5 Experimental methodology: Linux kernel implementation	29
5.1 Linux network stack overview	29
5.1.1 The <i>sk_buff</i> structure	29
5.1.2 Packet transmission overview	29
5.1.3 Packet reception overview	31
5.2 Netfilter architecture	33
5.3 Madwifi driver	33
5.4 Implementation details	34
5.4.1 Per-destination queues and netfilter-specific code	34
5.4.2 Scheduler	35
5.4.3 Promiscuous handler	35
5.4.4 Aggregation	36
Chapter 6 Experimental set-up and numerical results	38
6.1 Metrics of interest	38
6.2 OPNET simulations	39
6.2.1 Parameters	40
6.3 Linux testbed results	66
Chapter 7 Summary and future work	67
References	68

LIST OF TABLES

Table 5.1	Values of 802.11e parameters for different priority levels	34
Table 6.1	OPNET simulation parameters	43

LIST OF FIGURES

Figure 1.1	Throughput of different packet sizes	2
Figure 3.1	Format of a packet aggregate	9
Figure 3.2	PDQ scheduling without aggregation	13
Figure 3.3	PDQ scheduling with aggregation	14
Figure 4.1	OPNET node model wlan_wkstn_adv	22
Figure 4.2	OPNET process model wlan_proc	23
Figure 4.3	UW header format	24
Figure 4.4	Enqueue flowchart	26
Figure 4.5	Dequeue flowchart	27
Figure 4.6	Dequeue flowchart with Aggregation	28
Figure 5.1	Linux kernel packet transmission	30
Figure 5.2	Linux kernel packet reception	31
Figure 5.3	Netfilter hooks	32
Figure 6.1	Grid topology (49 nodes)	41
Figure 6.2	Clustered topology (36 nodes)	41
Figure 6.3	Random topology 1 (25 nodes)	42
Figure 6.4	Random topology 2 (16 nodes)	42
Figure 6.5	Aggregate Throughput (Random topology 1, 11 Mbps channel rate, exponential distribution)	44
Figure 6.6	Aggregate Throughput (Random topology 1, 1 Mbps channel rate, exponential distribution)	44
Figure 6.7	Network utility (Random topology 1, 11 Mbps channel rate, exponential distribution)	45
Figure 6.8	Network utility (Random topology 1, 1 Mbps channel rate, exponential distribution)	45
Figure 6.9	Fairness Index (Random topology 1, 11 Mbps channel rate, exponential distribution)	46
Figure 6.10	Fairness Index (Random topology 1, 1 Mbps channel rate, exponential distribution)	46
Figure 6.11	Average buffer occupancy (Random topology 1, 11Mbps channel rate, 4 flows, exponential distribution)	47
Figure 6.12	Average buffer occupancy (Random topology 1, 11Mbps channel rate, 8 flows, exponential distribution)	47
Figure 6.13	Average buffer occupancy (Random topology 1, 11Mbps channel rate, 12 flows, exponential distribution)	48
Figure 6.14	Average buffer occupancy (Random topology 1, 11Mbps channel rate, 16 flows, exponential distribution)	48

Figure 6.15	Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 8 flows, exponential distribution)	49
Figure 6.16	Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 12 flows, exponential distribution)	49
Figure 6.17	Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 16 flows, exponential distribution)	50
Figure 6.18	Aggregate throughput simulated vs theory (Random topology 1, 11 Mbps channel rate, exponential distribution)	50
Figure 6.19	Average network delay (Random topology 1, 11 Mbps channel rate, exponential distribution)	51
Figure 6.20	Random topology 1, 11 Mbps channel rate, exponential distribution, 50-60 percent medium utilization	51
Figure 6.21	Random topology 1, 1 Mbps channel rate, exponential distribution, 50-60 percent medium utilization	51
Figure 6.22	Aggregate throughput (Cluster topology, 11 Mbps channel rate, exponential distribution)	52
Figure 6.23	Network utility (Cluster topology, 11 Mbps channel rate, exponential distribution)	52
Figure 6.24	Fairness index (Cluster topology, 11 Mbps channel rate, exponential distribution)	53
Figure 6.25	Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 4 flows, exponential distribution)	53
Figure 6.26	Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 8 flows, exponential distribution)	54
Figure 6.27	Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 12 flows, exponential distribution)	54
Figure 6.28	Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 16 flows, exponential distribution)	55
Figure 6.29	Aggregation opportunities (Cluster topology, 11 Mbps channel rate, exponential distribution)	55
Figure 6.30	Average network delay (Cluster topology, 11 Mbps channel rate, exponential distribution)	56
Figure 6.31	Aggregate throughput (Grid topology, 11 Mbps channel rate, exponential distribution)	56
Figure 6.32	Network utility (Grid topology, 11 Mbps channel rate, exponential distribution)	57
Figure 6.33	Fairness Index (Grid topology, 11 Mbps channel rate, exponential distribution)	57
Figure 6.34	Average network delay (Grid topology, 11 Mbps channel rate, exponential distribution)	58
Figure 6.35	Average buffer occupancy (Grid topology, 11 Mbps channel rate, exponential distribution)	58
Figure 6.36	Aggregation opportunities (Grid topology, 11 Mbps channel rate, exponential distribution)	59

Figure 6.37	Aggregate throughput (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	59
Figure 6.38	Network utility (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	60
Figure 6.39	Fairness index (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	60
Figure 6.40	Average network delay (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	61
Figure 6.41	Average buffer occupancy (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	61
Figure 6.42	Aggregation opportunities (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)	62
Figure 6.43	Aggregate throughput (Random topology 2, 11 Mbps channel rate, exponential distribution)	62
Figure 6.44	Network utility (Random topology 2, 11 Mbps channel rate, exponential distribution)	63
Figure 6.45	Fairness Index (Random topology 2, 11 Mbps channel rate, exponential distribution)	63
Figure 6.46	Average buffer occupancy per node (Random topology 2, 11 Mbps channel rate, exponential distribution)	64
Figure 6.47	Average network delay (Random topology 2, 11 Mbps channel rate, exponential distribution)	64
Figure 6.48	Aggregate throughputs (Random topology 2, 4 flows, Different channel rates)	65
Figure 6.49	Random topology 2, different channel rates, exponential distribution . . .	65
Figure 6.50	4 node Linux testbed topology	66
Figure 6.51	Aggregate throughput, Linux testbed, 11 Mbps channel rate	66

Chapter 1

Introduction

1.1 Background and challenges

Wireless multi-hop or wireless mesh networks [5, 7, 31] are attractive for last-mile technology owing to the fact of ease of deployment, reconfiguration and lack of wired infrastructure. However these benefits are offset by the constraints of limited capacity and unfairness across different traffic flows. Wireless multi-hop networks face the fundamental problem of interference in a local neighbourhood. Due to this, at a given time only one link needs to be scheduled for a successful transmission between two nodes in the local interference neighbourhood. In order to maintain fairness and at the same time obtain good capacity from the network, the standard 802.11 tries to be fair at the MAC layer by using the MACA method. While this works for single hop networks consisting typically of one access point and many clients; in the context of multi-hop networks, this method introduces capacity degradation and unfairness across different traffic flows and hence fails dramatically. Joint congestion control and back pressure based scheduling techniques have been proposed to alleviate these problems [6, 30]. Another problem with wireless networks is that of the overhead associated for transmitting a data frame. Due to neighbourhood interference, the overhead percentage of frame transmission of smaller payload is significantly greater than that of a frame with a larger payload. This property of wireless networks coupled with the characteristic of packet switched networks of having a majority of traffic consisting of smaller packet sizes have led many works to study packet aggregation which consists of reducing the MAC layer overhead by combining several MAC layer SDUs in one single frame. This thesis deals with the study of introducing packet aggregation in conformance with the back pressure scheduling techniques.

Figure 1.1: Throughput of different packet sizes

1.2 Motivation

Surveys on the wide area traffic patterns have shown that majority of the wide area internet traffic has small packet sizes [16, 29]. Also, in the last mile wireless networks, a considerable amount of traffic is formed of small packet sizes. [26] and [22] show that about 70 to 80 percent of the overall application level traffic has packet sizes less than 100 bytes. This is because applications like chat, web requests, ssh traffic, Kerberos, ICMP, DNS, BOOTP contribute to small packet sizes. The problem with small packet sizes is the overhead associated with them due to the PLCP, MAC header, trailers and contention time which almost increases exponentially with the number of wireless nodes in a collision domain. [14] shows that for a TCP acknowledgement of size 40 bytes, about 90 percent overhead is associated when 802.11b with 11M channel rate is used. With the higher percentage of small packet sizes and with the multi-hop scenario, it is evident that the overall network utilization would be much less owing to the huge overheads. Thus the idea of packet aggregation at the IP level is considered (multiple IP packets are aggregated into one single IP packet which then becomes the MAC SDU).

Figure 1.1 shows the throughputs with and without packet aggregation with respect to different packet sizes. In the simple experiment, we implemented packet aggregation in the linux kernel. Aggregation is implemented at the IP layer meaning that multiple IP packets

are packed into one single MAC SDU. Iperf was run between 2 machines with one wireless card on each of those machines set at 54Mbps channel rate, 11g MAC. It can be seen that for small packet sizes, the flow with aggregation receives more throughput than the flow with no aggregation. We can thus say that aggregation causes an increase in the service rate achieved by a flow.

The central principle of studies of cross layer approach to scheduling like [6] and [30] is to introduce differentiated service rate at the MAC layer based on the amount of backpressure of per-destination queues. They achieve this by the help of differentiated access to the medium. We investigate in our study whether we can combine such backpressure based PDQ mechanisms with packet aggregation technique to get at least some benefit out of aggregation in terms of the aggregate throughput while preserving the important benefit of PDQs which is fairness and congestion control. In this thesis, we use packet aggregation as a tool to introduce MAC layer service differentiation: queues with more back pressure have more opportunity for packet aggregation thereby increasing their service rate. Using our design of PDQ+Aggregation which will be described in later sections, we see that for small packet sizes we increase the aggregate throughput of the network while preserving fairness amongst flows. Also, in some cases, we actually get better fairness along with good aggregate throughput.

1.3 Organization

The thesis is organized as follows. Chapter 2 describes the related work in the domain which includes a review of joint backpressure based scheduling and congestion control. It also consists of a short survey on the packet aggregation techniques in wireless networks. In chapter 3 we describe our scheme of packet aggregation based backpressure scheduling. We also do a simple analysis of our scheme and derive expressions for the throughput of a flow and congestion on the nodes with and without aggregation. Chapter 4 covers the design of our scheme in OPNET. It discusses the changes in the *wlan_proc* process model of OPNET at a functional and design level. Next in chapter 5, we describe the implementation of our design in the Linux kernel testbed along with a basic description of the Linux kernel network stack. In chapter 6, we present our results after stating the OPNET parameters used in the simulations. In chapter 7, we summarize our work, present future directions and conclude.

Chapter 2

Related work

2.1 Work on backpressure scheduling and congestion control

The problem of scheduling involves determining which packet from a node is to be transmitted the next and also which link among a set of interfering links is to be scheduled the next. The problem of congestion control involves maintaining the source rate control meaning at each time step whether a packet should be added to a flow at the source node. The aim of both of these problems is to get good throughput and fairness across different flows in the network. Much of the previous research has considered these two problems separately. Kelly, Mauloo and Tan [15] addressed the issue of congestion control and have not looked at how data is scheduled. They tried to solve the congestion control problem as a distributed optimization problem, by formulating it as the classic Network Utility Modelling problem in the context of wireless networks defined below; where the utility functions U_f assigned to flows reflect the desire of the network as a whole to minimize congestion.

$$\text{Maximize: } \sum_f U_f(x_f)$$

subject to:

$$\sum_{f \in S_e} x_f \leq c_e$$

where S_e are the set of flows passing through server e , x_f is the current injection rate into flow f and c_e is the capacity of each server.

In the context of scheduling, studies [27] and [28] have explicitly considered the problem of scheduling. Their scheduling algorithm tries to move data from large queues to small queues.

[6] and [30] have considered the problem of MAC scheduling and congestion control jointly and have provided practical solutions to the problem. These studies both employ the notion

of per destination queues (PDQs) at each node in the network. These queues contain the packets destined to their respective destination. They also have the notion of urgency weights associated with it. The urgency weight of a PDQ is equal to the backlog of the queue minus the backlog of the PDQ of the next hop towards the destination. Also both of these studies employ a MAC providing differentiated service meaning that it is possible to provide different levels of MAC layer throughputs across different traffic flows. The MAC scheduling happens thusly:

1. Each node maintains a “urgency weight context” of the neighbourhood which contains the urgency weight information of the PDQs of the nodes in the neighbourhood.
2. When the next packet is to be transmitted, it is chosen from the PDQ with the highest urgency weight and is the head of line packet
3. The packet is transmitted with a MAC layer priority depending upon the urgency weight of its PDQ and the “urgency weight context”

[6] chooses the MAC layer priority of the packet by changing the value of CWmin which is specified by the 802.11 standard. Those packets which are from higher urgency weight PDQs take a low value of CWmin thereby increasing their chance of medium access quickly in the neighbourhood whereas those packets which come from lower urgency weight PDQs take a high value of CWmin thereby decreasing their chance of quickly accessing the medium. With respect to congestion control, this work uses a utility based congestion control. Each flow is assigned a log based utility where the utility or the “benefit” a flow provides to the system or the network as a whole is equal to the log of the current throughput received by that flow. The congestion control mechanism operates at the source node where the flow originates and is performed accordingly:

A packet from the source node is added to the PDQ only if it satisfies the following equation.

$$U'(x) > \beta * q \quad (2.1)$$

where β is a small constant and ‘q’ is the size of the respective PDQ. While [6] have simulated in OPNET [1]; [30] have implemented PDQ scheduling and congestion control in the Linux kernel. This work does the differentiated MAC layer scheduling using the IEEE 802.11e [21] mechanism. They have 4 levels of service at the MAC layer implemented in the modified madwifi driver. A packet to be transmitted is mapped to the appropriate service level based on its urgency weight.

The source rate control is done according to the AIMD (additive increase, multiplicative decrease which is used to control the congestion window in TCP) manner thusly:

Algorithm 1 AIMD source rate control

```
if  $qlen \geq QUEUE\_THRESH$  then  
     $rate \leftarrow rate/\beta$   
else  
     $rate \leftarrow rate + \alpha$   
end if
```

where $QUEUE_THRESH$, α and β are constants and $rate$ is the rate of the flow.

Solutions of [6] and [30] have shown to solve the problem of starvation of flows, provide better fairness and increase the network utility.

2.2 Work on packet aggregation

Aggregation can be performed at different layers: The MAC layer where it is called frame aggregation, the IP layer and even at the transport layer. Packets that are aggregated at a given layer share common headers of the layers below it. [11, 14, 18, 23, 24, 32], etc. belong to the category where packets are aggregated at the IP layer. Thus multiple IP packets go into the same MAC frame. On the other hand, aggregation which is performed at the MAC layer which is also called as frame aggregation has been experimented in [13, 17].

Aggregation can also be classified as hop by hop and end to end. In the end to end case, packets are aggregated at the source node itself and the aggregated packets then travel as it is to the destination. [11, 23] are examples of such a case. This introduces less aggregation opportunities at the intermediate nodes. [14] is an example of packet aggregation for the hop to hop case. Aggregation can also be classified as forced delay based or no-delay based. In the forced delay based aggregation packets are forced to wait for some time so that new packets are available for aggregation thereby increasing the aggregation opportunities. This kind of aggregation should be designed carefully as it might cause unnecessary delays which may be bad for applications like VoIP. [11, 14, 23] introduce forced delay based aggregation. The choice of optimal packet size for aggregation is also one of the important design parameters for getting good throughput in the face of noise. [23] uses the WCETT [20] metric of the path for choosing the maximum packet size of the aggregated packet that is transferred across that path. WCETT of a path from a source to destination is defined as the weighted sum of the sum of the ETTs (Expected transmission time) of links on that path and the largest ETT on that path.

Aggregation has been widely used in the field of wireless sensor networks. Mostly aggregation is used in this context to reduce the energy consumption of the sensor nodes and hence to increase the lifetime of the network. [9] has done a survey on the aggregation schemes in wireless sensor networks. [10] uses packet aggregation in a sensor network specifically not to

reduce the energy consumption but to alleviate the congestion in the network. In that network, the authors use special type of nodes which are called “aggregator” nodes which are dedicated to perform aggregation. In contrast to this, the work in this report does not have special “aggregator” node but instead any node can perform aggregation if it is eligible to do so.

In this thesis, aggregation is done on a per destination basis on each hop independently. In section 3.3, we give an example which describes our strategy of packet aggregation wherein the PDQ having the highest urgency weight in the neighborhood gets chance to aggregate packets upto the point where it remains the highest urgency weight PDQ.

A research that has particularly considered the idea of back pressure scheduling with aggregation is [19]. In their work, blocks are the units of aggregation. A block consists of multiple frames transmitted in a txop (transmission opportunity) and also multiple such txops. The MAC layer ARQ is disabled and there are no MAC layer ACKs for the frames. Thus BLOCK can be said to operate with its aggregation at the MAC layer. In contrast to this, we do not disable MAC layer ACKs and our aggregation scheme operates at the IP layer. Also BLOCK is an attempt to introduce a new type of transport protocol whereas we try to propose a solution which studies the effects of aggregation on back pressure based scheduling which we implement between the IP and the MAC layer and hence we do not presuppose any transport layer protocol.

Chapter 3

Packet aggregation with PDQ scheduling

3.1 Terms

In this section we will gloss over the important terms and their meanings used in the thesis.

3.1.1 Per Destination Queues (PDQ)

Each node maintains a separate queue for every possible destination in the network. The traffic that the node generates or forwards to a destination goes into the the PDQ for that destination.

3.1.2 Urgency weight

Every PDQ has an urgency weight associated with it. The urgency weight is equal to the backlog of the PDQ subtracted by the backlog of the PDQ on the next hop node towards the destination. The urgency weight information is used by the works [6] and [30] for back-pressure scheduling.

3.1.3 Urgency Weight state (UW state)

In works [6] and [30], each node in the network maintains the urgency weight information of the neighboring nodes. In this section, we formally define what kind of information is maintained by each node and call it as the urgency weight state or UW state. Each node maintains the following:

1. For each PDQ, the destination, next hop ID, backlog and urgency weight

Figure 3.1: Format of a packet aggregate

2. The PDQ ID, node ID and urgency weight of the PDQ having maximum urgency weight in the neighborhood
3. The PDQ ID, node ID and urgency weight of the PDQ having minimum urgency weight in the neighborhood

With the help of this information, a node determines the MAC priority of the next packet to be transmitted. How to maintain the UW state is explained in subsequent chapters 4 and 5.

3.1.4 MAC priorities

It is assumed that the MAC layer is able to provide different levels of service. In our experiments, 4 MAC priorities are considered: 0, 1, 2, and 3. 3 is the highest priority while 0 is the lowest. With a CSMA/CA kind of MAC which is considered in our experiments, when a packet is transmitted with priority 3, it has higher probability of accessing the medium while a packet with priority 0 has a lower probability of accessing the medium. With a regular 802.11 MAC, priority 3 means choosing the backoff with a low value of CWmin. This is used in OPNET simulations. With the 802.11e MAC, priority 3 means having lower values of AIFS, CWmin and CWmax. This is used in the linux kernel experiments. On a node, the next packet to be transmitted is chosen as the head of line (HOL) packet from the PDQ having the highest urgency weight. This is the intra-node scheduling. The MAC priority with which the packet is transmitted is chosen linearly from the range 0-3 by mapping the urgency weight of the PDQ to which the packet belongs to the range

$\{minimum\ urgency\ weight\ in\ the\ neighborhood, maximum\ urgency\ weight\ of\ the\ neighborhood\}$

It follows that if the PDQ from which the packet is to be transmitted has the highest urgency weight in the neighborhood, then the packet will be transmitted with priority 3 whereas if the PDQ has the lowest urgency weight in the neighborhood, then the packet will be transmitted with priority 0.

3.1.5 Packet aggregate

In our technique, aggregation is performed at the IP layer. Multiple IP packets are bundled together along with an aggregation header in one MAC service data unit. This forms a packet

aggregate. Its format is shown in figure 3.1. What contains in the aggregation header is explained in chapters 4 and 5.

3.2 Methodology

3.2.1 Basic idea

The idea of backpressure scheduling is to give more service rate to queues having higher urgency weight. With the help of aggregation, multiple packets from the same PDQ (per destination queue) with the highest urgency weight are aggregated together. Due to this, the service rate for that PDQ is increased. Hence aggregation with the help of backpressure based PDQ information will help achieve 2 objectives:

1. Increase in the service rate for high urgency weight PDQs
2. Due to the reduction in the overhead, a quicker access to the medium on an average for the HOL packets of all the PDQs.

The details of packet aggregation with PDQ scheduler are explained in subsequent sections.

3.2.2 Aggregation process

The aggregation process takes on a PDQ. With the basic PDQ scheduler at any given node, in a loop it de-queues the head of line packets from the PDQ with the highest urgency weight, maps the packet to a MAC priority level depending upon the values of the minimum and maximum urgency weights in the neighborhood and sends the packet. Now if a PDQ on a given node has the highest urgency weight in the neighborhood, then with a high probability the first few HOL packets from that PDQ will be mapped to the highest MAC priority level and then transmitted individually. Thus for each of these packets, the underlying MAC overhead of MAC header, PLCP header and contention will continue to exist. With the help of aggregation, these first few HOL packets are aggregated together at the IP level, meaning that these first few IP packets will be packed together into one single MAC SDU which entails that only the first packet from this set will experience the MAC overhead whereas the remaining others will experience none. Thus these first few IP packets from the highest urgency weight PDQ will form a train of packets. This means that the total time taken to transmit these set of packets from the given node to its next hop will reduce which means that the service rate for these set of packets and in turn for the PDQ with the highest urgency weight in the neighborhood has increased.

3.2.3 De-aggregation process

The de-aggregation process is quite simple. Whenever a node finds that an incoming packet is a packet aggregate, it peeps into the aggregation header of that packet and de-aggregates all the packets contained in it and re-injects each of those in the IP stack.

Algorithms 2 and 3 describe the process of MAC priority determination of a packet and the idea of our aggregation respectively.

Algorithm 2 MAC layer priority determination

```
max ← Maximum urgency weight in neighborhood
min ← Minimum urgency weight in neighborhood
urg ← Urgency weight of the PDQ
numlevels ← Number of MAC priority levels
maclevel := (urg - min) * maxlevels / (max - min)
return maclevel
```

Algorithm 3 PDQ + Aggregation (dequeue operation)

```
size ← 0
maxsize ← Maximum size of aggregated packet


← PDQ with the highest urgency weight on the node

repeat
  Dequeue HOL packet from p
  Add the packet to the packet aggregate
  size := size + sizeof(HOL packet)
until size ≤ maxsize AND p has highest urgency weight in the neighborhood
  Determine MAC priority of the packet aggregate using algorithm 2 and transmit
```

3.3 Example

In this section we explain our methodology with an example. The basic concept of our methodology is to aggregate those packets which are to be transmitted by the PDQ with the highest urgency weight into one single IP packet. Figure ?? shows the case when no aggregation is used with PDQ scheduling. Here we consider four PDQs in a neighborhood (the cloud in the figure ??). Each of these PDQs belong to four separate nodes. The urgency weights of the PDQs at an

instant are shown at the top of the PDQs. They are 0, 100, 600 and 300 respectively. Now each node maintains the UW state which is the values of urgency weights of each of the PDQs on the node along with the information about the maximum and minimum urgency weights of PDQ in the neighborhood. With the help of this information the MAC level priority of the next packet to be transmitted is determined in a way as described in sub-section 3.1.4. Also, in this example it is assumed that each PDQ carries packets of size 50 bytes. With the current scenario, PDQ 3 will transmit its first 6 packets with MAC priority 3 whereas PDQ 4 will transmit its first 2 packets with a MAC priority of 2. As a neighborhood is considered, only one packet can occupy the medium at any time instant. Ideally, over that period of time, since PDQ 3 is the one with the highest urgency weight in the neighborhood for the transmission of the first six packets, the first six packets should be transmitted before any other packet back-to-back. Then the first two packets from PDQ 4 should be transmitted. However, owing to a CSMA/CA kind of MAC, the order in which the packets across different nodes get access to the medium can change and figure 3.2 shows one possible schedule sequence. Packets P1 and P2 from PDQ 3 obtain access to the medium first, followed by packets Q1 and Q2 from PDQ 4, subsequently followed by packets P3 through P6 from PDQ 3. The ideal case of access would have been packets P1 through P6 gaining access back-to-back followed by packets Q1 and Q2. Our methodology of introducing aggregation in this kind of UW based scheduling is depicted in a similar scenario in figure 3.3. Due to the UW state information, node C knows that the PDQ 3 on it will retain the highest urgency weight in the neighborhood even after the transmission of the first six head of line packets. Hence, the first six packets are aggregated, i.e. there are bundled together and then the aggregation header is added to form the packet aggregate. This packet aggregate then becomes the MAC SDU of the MAC frame which is transmitted with MAC priority 3. Thus with the help of aggregation, those packets which should access the medium before others are guaranteed access first. Also, with the help of packet aggregation, the MAC layer overhead like MAC headers, PHY headers, trailers and contention overhead associated with all but one of the aggregated packets is reduced which results in higher throughput.

Figure 3.2: PDQ scheduling without aggregation

Figure 3.3: PDQ scheduling with aggregation

3.4 Theoretical analysis

In this section, we present a simple theoretical analysis of our scheme and obtain the expressions for throughput of each flow and the buffer process on nodes.

3.4.1 Terms

First we introduce some terms required by our analysis:

- q_d^v : This represents the PDQ of destination d at node v
- T : The total time for which the experiment is run
- $n(q_d^v, l)$: the number of packets from the PDQ q_d^v transmitted at level l
- $f_a(q_d^v, i)$: the aggregation function. This value represents the number of additional IP packets that are aggregated with the i^{th} packet aggregate from PDQ q_d^v . This follows that when no aggregation is used, for each packet aggregate i , $f_a(q_d^v, i)$ will be 0
- r : the raw channel rate

On a given node v there are multiple PDQs: $q_1^v, q_2^v, q_3^v, \dots$

Let us consider a PDQ q_d^v . Over the time period t , there will be transmissions of multiple packet aggregates. For each packet aggregate i , there are multiple times associated. They are:

Wait time $t_w(q_d^v, i)$: This is the time the HOL transmission i from PDQ q_d^v waits till transmissions from other PDQs on the same node having higher urgency weights are complete.

Service time $t_s(q_d^v, i)$: This is the time interval from the point the node starts an attempt to transmit the HOL packet from PDQ q_d^v till the point of time when the receiver of the transmission receives the transmission successfully. $t_s(q_d^v, i)$ depends upon the following factors:

1. Priority level of transmission i
2. Number of packets aggregated if any. This will depend upon the aggregation function $f_a()$
3. The *context* of the neighboring nodes, i.e the values of the residual back-offs of transmissions of the neighboring nodes.

Actual data transmission time $t_d(q_d^v, i)$: This is the time required to transmit the *useful* data which is the data from the IP packet. This time is a part of the service time $t_s(q_d^v, i)$

Thus for a given packet aggregate i from PDQ q_d^v , the throughput of transmission i or its service rate $s(i)$ is given by:

$$s(i) = \frac{t_d(i) \times r}{t_s(i) + t_w(i)} \quad (3.1)$$

Thus over time T , if there are X transmissions (packet aggregates) from the PDQ, then the average throughput of PDQ q_d^v is given by:

$$\text{Average throughput} = \sum_{i=1}^X \frac{t_d(i) \times r}{t_s(i) + t_w(i)} \quad (3.2)$$

Now let t_b represent the time to transmit the elementary data. This elementary data is of the smallest length. (for example, consider t_b to be 50 bytes which is the smallest IP packet length) This is just used for analysis.

Hence,

$$t_d(i) = t_b \times (f_a(i) + 1) \quad (3.3)$$

This means that the data transmission time of the i^{th} transmission is equal to the transmission time of the elementary data packet times the number of IP packets packed in one MAC SDU. Hence the service rate of the i^{th} transmission is given by:

$$s(i) = \frac{t_b \times (f_a(i) + 1) \times r}{t_s(i) + t_w(i)} \quad (3.4)$$

Analysis of service time $t_s(q_d^v, i)$: This is the service time of the i^{th} transmission from PDQ q_d^v . t_s can be composed into two times:

1. t_o : Overhead time
2. t_d : Data transmission time

We have already considered the data transmission time t_d in the analysis above. Now let us consider the overhead time t_o . t_o can be composed into 2 times:

1. Fixed (t_{of}): This is the time required to transmit the fixed parts of the MAC frame which includes the MAC, PHY, PLCP headers and trailers.
2. Variable (t_{ov}): This time includes the following:
 - Backoff time (t_{ovb})

- Waiting time due to busy medium (t_{ovw})

Hence for the i^{th} packet aggregate, we have:

$$\begin{aligned}
t_s &= t_o + t_d \\
&= (t_{of} + t_{ov}) + t_d \\
&= (t_{of} + t_{ovb} + t_{ovw}) + t_d
\end{aligned} \tag{3.5}$$

Considering the value of t_d from equation 3.3, the time required to service t_d amount of data is:

- With no aggregation:

$$t_s = (1 + f_a) \times ((t_{of} + t_{ovb} + t_{ovw}) + t_b) \tag{3.6}$$

- With aggregation:

$$t_s = (t_{of} + t_{ovb} + t_{ovw}) + (1 + f_a) \times t_b \tag{3.7}$$

Comparing equations 3.6 and 3.7 we can clearly see that the time required to serve t_d amount of data with aggregation is less than that required to serve the same amount of data with no aggregation. Thus aggregation helps reduce the overhead associated per transmission thereby increasing the throughput.

3.4.2 Modelling average throughput of a flow

Let us consider PDQ q_d^v .

We have to derive the throughput of a flow in the network. For this analysis, PDQ flows are considered, i.e. for a given destination, there is only one flow having its termination point to it. Also we consider that experiments are run over a time period T . Now a given flow passes through several PDQs towards its destination. Hence under stability, the average throughput that a flow gets is equal to the average throughput of any of the PDQs through which the flow passes. Now we will derive the expressions for the average throughputs for the cases of both with and without aggregation.

$$\text{Average throughput} = \frac{\text{Time required to serve useful data}}{T} \times r \tag{3.8}$$

1. Without aggregation:

Let X be the number of transmissions from PDQ q_d^v over the time period T .

$$\text{Average throughput} = \frac{\sum_{i=1}^X t_b(i)}{\sum_{i=1}^X t_s(i) + t_w(i)} \times r \quad (3.9)$$

2. With aggregation:

Let X_a be the number of transmissions from PDQ q_d^v over the time period T .

$$\text{Average throughput} = \frac{\sum_{i=1}^{X_a} (1 + f_a(i)) \times t_b(i)}{\sum_{i=1}^X t_s(i) + t_w(i)} \times r \quad (3.10)$$

Choice of the aggregation function $f_a(i)$: The aggregation function $f_a(i)$ is defined for each PDQ. It determines the number of IP packets that should be aggregated in the i^{th} transmission (i.e the i^{th} packet aggregate). The choice of this function is important as it will affect the throughput of all the flows in a network. The proposed strategy is to have $f_a(i)$ for a given PDQ greater than 0 only if the PDQ has the highest urgency weight in its neighborhood. This means that packets are aggregated only from the PDQ which has the highest urgency weight in the neighborhood. The motivations for this are as follows:

1. Consider the case where a node v has a PDQ with the highest urgency weight in its neighborhood. Also the node v has many neighbors each of whose PDQs have packets to transmit. Now it may happen that the residual MAC back-offs of at least one of the neighbors expire and hence that neighbor gets access to the medium (this may happen due to the probabilistic nature of the CSMA/CA with back-off). Let's call the neighbor which gains an access to the medium as n . In such a case, if packets from the node n 's PDQ are aggregated and if the channel rate r is small, then as there are many IP packets in the MAC payload of the node n 's transmission, this transmission will occupy a large amount of time on the medium. Mathematically, this means that the service time of the i_{th} transmission as given in eq 3.6 will increase. Due to this, the node with the highest urgency weight in the neighborhood v will have to wait for a longer amount of time in order to access the medium, which in turn mathematically means that the service time of the current transmission i of the highest urgency weight PDQ on node v will increase. Due to this, the service rate for its current transmission will decrease as given by 3.4. On the other hand, if packets from the lower urgency weight PDQ of node n were not aggregated, i.e. had $f_a(i)$ been 0, then for the case where the channel rate r is low, the transmission of node v would have taken less time on the medium as a result of which node v would have gotten quicker access to the medium and hence the service rate of node v would have increased.

This situation is similar to that of the priority inversion problem in operating systems. In that case, if there are multiple process with different priority levels sharing a common resource, and if the lower priority process gains access to the resource and subsequently a higher priority process requests access to the same resource and hence gets blocked, the lower priority process, due to its lower service rate will take more time to release the resource. Due to this the higher priority process will get starved for a more period of time as a result of which its service rate will decrease. To avoid this, the lower priority process which has acquired the resource is scheduled with a higher priority so that it can release the resource quickly. In similar terms, with the packet aggregation case example considered above, the node n with the lower urgency weight which has acquired the shared medium before the highest urgency weight node v should release the medium quickly so that node v can get access to the medium quickly, which further means that $f_a(i)$ for higher urgency weight PDQs should be higher and for lower urgency weight PDQs should be lower.

2. The other motivation behind not aggregating packets from lower urgency weight PDQs is that if this is done, it violates the basic principle of backpressure scheduling which states that lower urgency weight PDQs should have lower service rate. Also by aggregating more packets from the lower urgency weight PDQ, packets to the next hop PDQ arrive at a faster rate thereby increasing the congestion on the next hop PDQ.

3.4.3 Modelling the buffer process

The buffer process of a PDQ represents the congestion at the PDQ. Let us consider PDQ q_d^v . Let $b(t)$ represent the buffer process of the PDQ. This is equal to the number of bytes present in the PDQ at time t .

Let $r(t)$ denote the instantaneous incoming data rate at the PDQ. This is in units of number of bytes per second. $r(t)$ is equal to the sum of the data rates from different PDQs. Thus if data comes to the considered PDQ from multiple nodes m (i.e. the node is the next hop to multiple nodes), then $r(t)$ is equal to the summation of the incoming rates. Let,

$$r(t) = \sum_{i=1}^m r(t, i) \quad (3.11)$$

where $r(t, i)$ is the incoming rate to the PDQ q_d^v from node i .

Let $s(t)$ denote the instantaneous service rate of PDQ q_d^v .

Hence the buffer process of the PDQ q_d^v at a later time instant $t + \alpha$ is given by:

$$b(t + \alpha) = \int_t^{t+\alpha} [r(t) - s(t)]y \, dt \quad (3.12)$$

Hence in order to make the queues stable or in other words to maintain the queue sizes within acceptable bounds, the value $(r(t) - s(t))$ should be kept as small as possible.

In the case where multiple incoming flows constitute $r(t)$, in order to maintain less difference between $r(t)$ and $s(t)$, it means that $s(t)$ should be greater than the individual incoming rates $r(t, i)$. To accomplish this, the following should be true:

1. Over the time period T , the source nodes of the transmissions of the incoming flows should have less percentage of high priority MAC transmissions than compared to the outgoing flow of the PDQ
2. The aggregation opportunities (i.e. the aggregation function $f_a()$) should on an average have less values for incoming transmissions to the PDQ whereas the function should on an average have higher values for outgoing transmissions from the PDQ

Chapter 4

Experimental methodology: OPNET simulation

This chapter explains the architecture of PDQ scheduler with packet aggregation in detail in OPNET simulation. Section 4.1 briefly enlists the requirements of the design. Section 4.2 briefly touches upon OPNET basics and section 4.4 will contain the actual description of the changes in the *wlan_proc* process model to simulate the PDQ + packet aggregation architecture.

4.1 Requirements

The PDQ scheduler with packet aggregation needs to meet the following design criteria:

- Each node in the network should have updated information of the UW state.
- There is no centralized node to co-ordinate the above information and hence the information should be maintained at each node in a distributed manner.
- The propagation of information across different nodes will have some overhead associated with it and it should be minimum.

4.2 OPNET basics

4.2.1 Node model

A node model is used to represent a network object and its characteristics. A network object could be a node, client station or a router. In our simulations, the node model *wlan_wkstn_adv* which is shown in figure 4.1 is used. The internal components of a node model consists of the following elements:

- Modules: A module represents internal aspects of a node such as data creation, storage and processing. For example, some of the modules of the node model *wlan_wkstn_adv* are *ip*: internet protocol, *aodv*: routing protocol AODV, *wlan*: the wireless LAN module
- Packet streams: These are used to transfer data packets across different modules within the same node model
- Statistic wires: These are used to transfer statistic information like queue sizes, data rate, packet loss, etc. across the different modules

Figure 4.1: OPNET node model *wlan_wkstn_adv*

4.2.2 Process Model

A process model is used to represent the internal working of a module. A process model consists of a finite state machine with a set of states and lines interconnecting the state. These lines

PDQ ID	Urgency weight	backlog	min PDQ ID	Min PDQ urgency weight
--------	----------------	---------	------------	------------------------

Figure 4.3: UW header format

4.4 Changes in the wlan process model

4.4.1 Event: Packet arrival from higher layer

In the *wlan_proc* process, when a packet arrives from the higher layer, it is added to the queue *hld_list_ptr*. But with our code, it is added to one of the PDQs. The function that enqueues the packet is declared as follows:

```
static void PDQ_enqueue(struct wgpd_state* gs, WlanT_Hld_List_Elem* hld_ptr, OpT_Int64 my)
```

In this event, actions specified in figure 4.4 are performed. First source rate control is performed. In our experiments we use log based utility function for each flow. This utility function was used by paper [6]. A packet is added to the PDQ only when the following condition is satisfied.

$$U'(x) > \beta * QueueSize \quad (4.1)$$

Here β is a small constant whose value is 0.000001 in our experiments.

In [6], the authors had the value of β set to a small constant. *QueueSize* is the size of the PDQ to which the packet is to be enqueued and $U'(x)$ is the first derivative of the utility function. Equation 4.1 means that a packet is added to a PDQ only when the incremental benefit to the system is greater than the product of the source queue size and a small constant.

When a packet is added to a PDQ, the urgency weight and backlog of the PDQ are updated.

4.4.2 Event: Packet transmission

In the *wlan_proc* process, a packet transmission begins in the function:

```
static void wlan_frame_transmit ()
```

In this function, the HOL packet from the queue *hld_list_ptr* is removed and transmitted. However, in our framework, the head of line packet from the PDQ with the highest urgency weight is removed and transmitted. The transmission process is depicted in figure 4.5. This

is the case when there is no packet aggregation involved. However, when there is aggregation involved, the transmission process as depicted in figure 4.6 takes place. The function that performs the dequeue operation has the following prototype. It handles both the cases with and without aggregation:

```
static WlanT_Hld_List_Elem* gcd_PDQ_dequeue(struct gcd_wgpd_state* gs)
```

Without aggregation, whenever the HOL packet from a PDQ is chosen to be transmitted, it is assigned a MAC level priority based upon the UW state on this node and the neighboring nodes, is added the UW header as shown in figure 4.3 and simply transmitted. The urgency weight, backlog and the throughput of the PDQ are then updated. Then packets from the source list of the PDQ are brought in the PDQ until the source rate control allows or the source list is non-empty. With aggregation, the above process remains mostly the same except this time, when a HOL packet is dequeued from a PDQ, it is checked whether the PDQ has the highest urgency weight in the neighborhood. If so, it is check whether by dequeuing the next HOL packet from the PDQ, the PDQ would still have the highest urgency weight in the neighborhood. If this is true, then the next packet is removed. Packets are removed from the PDQ as long as the PDQ retains highest urgency weight in the neighborhood or until sum of the size of all the dequeued packets is less than the maximum MAC SDU size. In the experiments, we set the maximum MAC SDU size to be 1400 bytes. All the dequeued packets are the IP packets which are bundled up together in one packet aggregate. The packet aggregate has the format shown in figure 3.1. In the OPNET simulation the aggregation header is nothing but a one byte field indicating the number of packets in the packet aggregate. The packet aggregate is then added a properly filled UW header and transmitted.

4.4.3 Event: Packet arrival from physical layer

A received MAC layer frame is processed in the function *wlan_physical_layer_data_arrival()*. In this function, before processing the frame it is checked whether the frame is destined to this station. If so, it is processed. If not, it is simply discarded. Before this checking happens, our code first removes the UW header from the frame and based on its contents updates the UW state. Hence at this point, the UW header is processed even if the frame is not destined to this station. This helps a station to get to know the UW state of its neighboring stations. Also at this point, if the frame contains a packet aggregate, the aggregation header is removed. From this function, the function *wlan_data_process()* is called which does the reassembly of the PHY segments (if one frame is transmitted into more than one segment), forms a packet out of those and passes it over to the higher i.e. IP layer. Now with aggregation, this function converts the packet aggregate into individual packets and hands each packet over to the IP layer separately.

Figure 4.4: Enqueue flowchart

Figure 4.5: Dequeue flowchart

Figure 4.6: Dequeue flowchart with Aggregation

Chapter 5

Experimental methodology: Linux kernel implementation

5.1 Linux network stack overview

This section will briefly cover the important parts of the Linux networking stack which are relevant to our implementation. Detailed information of the stack is given in [12].

5.1.1 The *sk_buff* structure

Linux provides an efficient way of representation of network data in the kernel with the help of a socket buffer structure called as *struct sk_buff*. This structure contains the headers of different layers as well as the actual data. Every packet in the stack is associated with one instance of this structure. The implementation of this structure is such that as the packet traverses across different layers of the stack, the data in it holds its position. Only the pointers pointing to the relevant headers are modified to reflect the change in the packet. This avoids the costly copy operation resulting in efficient implementation. The kernel provides APIs for allocating, deallocating, copying, cloning, adding more space to the data portion and removing space from the data portion of this structure. The fields of interest of this structure are: the length of this structure, the pointer to the IP header of this structure by which the destination address, TOS value, etc. can be modified.

5.1.2 Packet transmission overview

Figure 5.1 gives an overview of the various points in the linux kernel network stack that a transmitting packet goes through. At the very beginning is the socket layer where interface to user land processes is provided. A socket buffer is allocated at this layer. The data written to

Figure 5.1: Linux kernel packet transmission

the socket is processed by the TCP/UDP transport layer. Then the transport layer calls the function *ip_queue_xmit()* to hand it over to the IP layer. The IP layer adds the IP headers and performs routing. Also the most important Netfilter architecture is implemented at this layer whose details shall be covered in section 5.2. After IP processing, the API *dev_queue_xmit()* is called which passes the packet to the queueing discipline. The queueing discipline is network interface specific and does traffic conditioning and policing. The queueing discipline then calls the function in the function pointer *dev->hard_start_xmit()* which is provided by the NIC driver. This function is the last function in the transmission process which physically transmits the packet. After successful packet transmission or after a certain number of retries due to failure, the device driver de-allocates the socket buffer for this packet. It is at this point where the journey of the packet at the transmission path ends.

Figure 5.2: Linux kernel packet reception

5.1.3 Packet reception overview

Figure 5.2 shows the path a packet takes in the linux kernel when it has been received from the NIC. When a packet is received by the NIC, it issues an interrupt to the kernel. In the interrupt handler routine, the device driver allocates an instance of *struct sk_buff* and copies the packet data from the card into the *sk_buff* memory area. The driver then passes the *sk_buff* to the *netif_rx()* kernel API and the interrupt handler is terminated. The function *netif_rx()* simply enqueues the *sk_buff* into the proper per-CPU queue and exits. Most of the linux network stack runs in the context of the soft-IRQ *NET_RX_SOFTIRQ*. When the kernel scheduler schedules this soft-IRQ, its handler function *net_rx_action()* is executed. This function removes the queued *sk_buffs* from the per-CPU queues following some execution time limit and then

Figure 5.3: Netfilter hooks

depending upon the protocol field in the MAC layer header of the packet hands it over to the appropriate protocol handler routines. In case the MAC SDU is an IP packet, the function passes the *sk_buff* to the IP layer handler routine which is *ip_rcv()* as shown in the figure. This function is the entry point of the IP layer processing. The IP layer then hands the *sk_buff* over to transport layer protocols UDP/TCP depending upon the protocol field in the IP header by invoking their handling routines *udp_rcv()* and *tcp_rcv()* respectively. As shown in subsection 5.4.3, we can register our own promiscuous handlers in order to receive all the IP packets (the ones that are even not destined to the local machine). The green box *Promiscuous handler* in the figure 5.2 is our registered handler that will be called by the function *net_rx_action()*. Also the other two green boxes shown in the IP layer box are the packet de-aggregation module and module that removes the UW header. These are described in detail in subsequent sections.

5.2 Netfilter architecture

The netfilter architecture [2] provided by Linux allows the kernel developer to insert his own code in the network stack and capture packets, change its fields, drop packets, etc. Netfilter provides support for various protocols but the current implementation only makes use of the IP protocol functionality. The netfilter provides five hooks in the IP code as shown in figure 5.3 which are:

1. `NF_INET_LOCAL_OUT`
2. `NF_INET_LOCAL_IN`
3. `NF_INET_PRE_ROUTING`
4. `NF_INET_POST_ROUTING`
5. `NF_INET_FORWARD`

These hooks enable us to register our functions. The functions registered at hooks are called in the order of priority when a packet passes through the hook. The current implementation make use of the *NF_INET_PRE_ROUTING* and *NF_INET_POST_ROUTING* hooks.

5.3 Madwifi driver

Madwifi [3] is an open source driver for Atheros chipset wireless cards for Linux. For the experiments, we have used the modified madwifi driver by the HOP group [4]. This driver contains the logic of having 4 service levels at the MAC layer in ahdemo mode. The ahdemo mode is a special mode (other modes are: management, ad-hoc) provided by madwifi in which there are no beacons and management frames transmitted by a card. The modified madwifi logic enables choosing a set of values of the *AIFS*, *CWmin* and *CWmax* parameters of the 802.11e standard based on the value of the TOS field in the IP header of the packet. There are 4 MAC priorities implemented in the driver. The values of *AIFS*, *CWmin*, *CWmax* for each of the priorities are given in table 5.1.

Table 5.1: Values of 802.11e parameters for different priority levels

Priority level	AIFS	CW _{min}	CW _{max}
0	3	4	10
1	7	4	10
2	2	3	4
3	1	2	3

5.4 Implementation details

5.4.1 Per-destination queues and netfilter-specific code

PDQs

The PDQs are implemented as linked lists where each item in the list is of the type:

```
struct nf_queue_entry*
```

Each PDQ maintains state information that contains the following:

- Destination ID
- Backlog in bytes
- Backlog in packets
- Backlog in bytes of the next hop node
- Urgency weight

The enqueue and dequeue procedures on these PDQs are same as given in figures 4.4 and 4.5.

Netfilter

The implementation makes use of 2 netfilter hooks:

1. `NF_INET_POST_ROUTING`
2. `NF_INET_PRE_ROUTING`

The *NF_INET_POST_ROUTING* hook is used to capture packets that are generated locally at this node or are forwarded by this node. After capturing a packet, a return value of *NF_QUEUE* is returned which means that the captured packet is to be queued. The queue-handler registered for this purpose is defined as follows:

```
static struct nf_queue_handler nfqh =
{
    .name    = "scheduler",
    .outfn   = &scheduler_enqueue_packet,
};
```

Here the name of the queue handler is *scheduler* and the handling function is *scheduler_enqueue_packet*. Hence when a packet arrives at the post-routing hook, the aforementioned queue handler is called. The queue handler function determines the destination ID of the packet and calls our PDQ function which is declared as follows:

```
void pdq_enqueue(unsigned int id, struct nf_queue_entry *nf);
```

There are two functions registered at hook *NF_INET_PRE_ROUTING*. One of these functions handles the de-aggregation of packets which will be described in sub-section 5.4.4. The other one captures the packets, removes the UW header from the packet and then re-injects the packet in the network stack.

5.4.2 Scheduler

The scheduler is implemented as a kernel thread. The thread waits in a while loop and during each iteration, it dequeues the head of line packet from the PDQ with the highest urgency weight. Now if aggregation is not enabled, the dequeue process takes form as given in figure 4.5. If aggregation is enabled, then it dequeues packets from the highest urgency weight PDQ until that PDQ has the highest urgency weight in the neighborhood or until the size of the packet aggregate reaches the maximum allowed payload size as shown in figure 4.6. Then it forms the UW header and appends it to the packet/ packet aggregate. It then re-injects the packet back in the IP stack. In case all of the PDQs are empty, the scheduler sleeps for 1 msec., then wakes up and continues to work.

5.4.3 Promiscuous handler

The UW header that each packet carries is processed by the *PRE_ROUTING* netfilter hook as mentioned previously. However, in our distributed algorithm, each node needs to be aware of the urgency weight information of the neighboring nodes. For example, if a packet is transmitted from node 1 to node 2 and node 3 is in the neighborhood of node 1, then node 3 should have the urgency weight information on node 1. Hence each node has its wireless card in the promiscuous mode. This enables the IP module in the network stack to receive IP packets that are directed to other hosts. The promiscuous handler is declared in the following way:

```

struct packet_type wgpdpromiscuous =
{
    .type    =    htons(ETH_P_IP),
    .func    =    wgpdpromiscuous_rcv,
};

```

The above declaration registers a protocol handler at the IP layer whose handling function is *wgpdpromiscuous_rcv*. This protocol handler receives all the IP packets that are received at the MAC layer.

The promiscuous handler is registered in the following manner:

```

dev_add_pack(&wgpdpromiscuous);

```

The promiscuous handler removes the UW header of the received packet and updates its UW state according to the information contained within the UW header.

5.4.4 Aggregation

As mentioned previously, aggregation is performed when a packet is de-queued from a PDQ at the *NF_INET_POST_ROUTING* hook. In the kernel implementation, a packet aggregate follows the same format as specified in figure 3.1. The aggregation header is, however unlike the one used in OPNET simulations. The aggregation header here is an IP header which we shall refer to as IP-aggregation header. The protocol code in this header is set to 253 in order to help the stack recognize that this is a packet aggregate. The value 253 is specified as experimental by the IANA and hence we have used it. The total length field in the IP-aggregation header is set to the total length of the packet aggregate. While transmitting, when multiple IP packets need to be packed in one packet aggregate, space needs to be created in the socket buffer. For this, the following kernel API is used:

```

int pskb_expand_head (struct sk_buff * skb, int nhead, int ntail, int gfp_mask);

```

The de-aggregation logic is implemented at the *NF_INET_PRE_ROUTING* hook. Whenever a packet reaches this hook, it is determined whether it is a packet aggregate by checking whether the value of the protocol field in the IP header is 253. If it is, then socket buffers equal to the number of IP packets packet in the packet aggregate are allocated and properly filled. Then the function *netif_rx()* is called on each of this newly created socket buffer so that they get freshly entered into the network stack.

The hook registered for this purpose is defined as follows:

```
static struct nf_hook_ops pre_skb_shared_ops =
{
    .hook          = pre_hook,
    .owner         = THIS_MODULE,
    .pf            = PF_INET,
    .hooknum       = NF_INET_PRE_ROUTING,
    .priority      = NF_IP_PRI_FILTER,
};
```


Chapter 6

Experimental set-up and numerical results

6.1 Metrics of interest

In our experiments, we measure the following entities:

- Aggregate throughput:

This is simply the sum of the throughputs of all the flows in the network over a given period of time. With aggregation, we suppress the overhead of the MAC and PHY headers and hence we expect to obtain better aggregate throughput with the PDQ+aggregation scheme than that without aggregation.

- Aggregate network utility:

In our experiments, we measure the logarithmic network utility which is defined as:

$$\text{Network utility} = \sum_i \log(f_i) \quad (6.1)$$

where:

f : The flows in the network

Network utility measures the benefit to the system. This metric has been used by works [6] and [30].

- Network fairness index:

We use Jain's fairness index [8] defined as:

$$\text{Fairness index} = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2} \quad (6.2)$$

where:

n : Number of flows in the network

x_i : Throughput of each flow

Jain's fairness index determines how fair the rate allocation in the network is. An index close to 1 indicates more fairness whereas the ones closer to 0 denote unfair rate allocation.

With our PDQ+Aggregation policy, we expect better aggregate throughputs and we study how this policy affects network fairness and utility. It is desired to preserve good values of fairness index and network utility.

6.2 OPNET simulations

We used different types of topologies for simulation. They are the grid topology in figure 6.1 consisting of 49 nodes, cluster topology shown in figure 6.2 consisting of 36 nodes and random topologies shown in figures 6.3 and 6.4 consisting of 25 and 16 nodes respectively. For each topology we experiment with different number of flows (4, 8, 12 and 16) and observe the aggregate throughput, network utility, fairness index, average network delay and the average buffer occupancy at intermediate nodes with both the schemes: with and without aggregation. We also record the aggregation opportunities for the first two topologies to quantify the benefit of aggregation.

It is observed that each of the topologies, PDQ + Aggregation scheme achieves better throughput and network utility. Also, the fairness index of the network is sometimes slightly worse and sometimes slightly better than its counterpart. Also, the average network delay also gets low. This is owing to the fact that with aggregation, there are multiple IP level packets for which the MAC and PHY overheads are avoided. Also, with respect to congestion, it is seen that for PDQ + Aggregation scheme, intermediate nodes have less buffer sizes on an average. All of the results shown are derived from scenes where the medium utilization is about 80 to 90 percent, meaning that on an average, a node attempts an access to the medium for 80 to 90 percent of the time. We would also like to observe the results in cases where the medium utilization is relatively low. Hence we repeat experiments on random topology 1 where the rate at which data is pumped by the source nodes is less causing the average system utilization

to lie somewhere between 50 to 60 percent. Figures 6.20 and 6.21 show the results. Under low medium utilization, the nodes will have comparatively less aggregation opportunities, but still it is observed that PDQ+Aggregation schemes gives better aggregation throughputs while preserving the fairness index.

Figure 6.18 compares the aggregate throughputs from theoretical expressions of equations 3.9 and 3.10 and the simulated ones. From the graph, we can see that the theoretical expressions overestimate the actual simulated results. Also, the degree of overestimate increases with the number of flows. One of the possibilities of this overestimation is due to the fact that we do not take into account the overheads caused by the UW header and the aggregation header in theoretical calculations. The other possibility is that we do not consider the degradation due to retransmissions.

In another set of experiments, we study the behaviour of the aggregation scheme with respect to different channel rates. We experiment four flows on random topology of figure 6.4. The flows are from nodes 2 to 0, 8 to 3, 9 to 4 and 13 to 5. The channel rates used are 11b (1, 2, 5.5 and 11Mbps) and 11g(54Mbps). The packet size of the application is 50 bytes. For each of these channel rates, we study aggregate throughput, network utility and fairness index for the aforementioned setting. For these experiments, it is observed that as raw the channel rate increases, the aggregate throughput (figure 6.48) also increases. For an application packet size of 50 bytes, the aggregate throughput steadily increases from about 60 percent for 1Mbps to about 100 percent for 54Mbps. The aggregate network utility for each of the channel rates increases is at least slightly better as shown in figure 6.49a. However, the fairness index for almost every channel rate is worse than the scheme without aggregation(figure 6.49b). Thus, even though aggregation increases the aggregate throughput considerably and network utility by a small amount, this is at the cost of a lower fairness index.

We also study how aggregation performs on real traffic scenarios of the internet. To emulate this, we generate traffic of packet sizes 40, 60, 80, 100, 200, 576 and 1400 bytes randomly with weights 40, 15, 10, 5, 5, 5 and 20 percent respectively. We simulate these traffic patterns with 4, 8, 12 and 16 flows on the grid topology. The results are shown from figures 6.37 through 6.42.

6.2.1 Parameters

In our experiments, we mostly use the default OPNET parameters, some of which are shown in table 6.1. Each flow demand has exponential arrival rate. To observe the effects of our aggregation scheme, the packet size is set to 50 bytes.

Figure 6.1: Grid topology (49 nodes)

Figure 6.2: Clustered topology (36 nodes)

Figure 6.3: Random topology 1 (25 nodes)

Figure 6.4: Random topology 2 (16 nodes)

Table 6.1: OPNET simulation parameters

Parameter	value
Transmit power	0.001W
Packet Reception power threshold	-67
Routing protocol	AODV
Channel rate	11Mbps
RTS threshold	None
Application demand type	Exponential
Average demand rate	30000000 packets/hour
Transport protocol	UDP
Aggregation threshold	1400 bytes
Packet size	50 bytes

Figure 6.5: Aggregate Throughput (Random topology 1, 11 Mbps channel rate, exponential distribution)

Figure 6.6: Aggregate Throughput (Random topology 1, 1 Mbps channel rate, exponential distribution)

Figure 6.7: Network utility (Random topology 1, 11 Mbps channel rate, exponential distribution)

Figure 6.8: Network utility (Random topology 1, 1 Mbps channel rate, exponential distribution)

Figure 6.9: Fairness Index (Random topology 1, 11 Mbps channel rate, exponential distribution)

Figure 6.10: Fairness Index (Random topology 1, 1 Mbps channel rate, exponential distribution)

Figure 6.11: Average buffer occupancy (Random topology 1, 11Mbps channel rate, 4 flows, exponential distribution)

Figure 6.12: Average buffer occupancy (Random topology 1, 11Mbps channel rate, 8 flows, exponential distribution)

Figure 6.13: Average buffer occupancy (Random topology 1, 11Mbps channel rate, 12 flows, exponential distribution)

Figure 6.14: Average buffer occupancy (Random topology 1, 11Mbps channel rate, 16 flows, exponential distribution)

Figure 6.15: Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 8 flows, exponential distribution)

Figure 6.16: Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 12 flows, exponential distribution)

Figure 6.17: Average buffer occupancy (Random topology 1, 1 Mbps channel rate, 16 flows, exponential distribution)

Figure 6.18: Aggregate throughput simulated vs theory (Random topology 1, 11 Mbps channel rate, exponential distribution)

Figure 6.19: Average network delay (Random topology 1, 11 Mbps channel rate, exponential distribution)

Figure 6.20: Random topology 1, 11 Mbps channel rate, exponential distribution, 50-60 percent medium utilization

Figure 6.21: Random topology 1, 1 Mbps channel rate, exponential distribution, 50-60 percent medium utilization

Figure 6.22: Aggregate throughput (Cluster topology, 11 Mbps channel rate, exponential distribution)

Figure 6.23: Network utility (Cluster topology, 11 Mbps channel rate, exponential distribution)

Figure 6.24: Fairness index (Cluster topology, 11 Mbps channel rate, exponential distribution)

Figure 6.25: Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 4 flows, exponential distribution)

Figure 6.26: Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 8 flows, exponential distribution)

Figure 6.27: Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 12 flows, exponential distribution)

Figure 6.28: Average buffer occupancy (Cluster topology, 11 Mbps channel rate, 16 flows, exponential distribution)

Figure 6.29: Aggregation opportunities (Cluster topology, 11 Mbps channel rate, exponential distribution)

Figure 6.30: Average network delay (Cluster topology, 11 Mbps channel rate, exponential distribution)

Figure 6.31: Aggregate throughput (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.32: Network utility (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.33: Fairness Index (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.34: Average network delay (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.35: Average buffer occupancy (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.36: Aggregation opportunities (Grid topology, 11 Mbps channel rate, exponential distribution)

Figure 6.37: Aggregate throughput (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.38: Network utility (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.39: Fairness index (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.40: Average network delay (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.41: Average buffer occupancy (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.42: Aggregation opportunities (Grid topology, 11 Mbps channel rate, exponential distribution, internet packet sizes)

Figure 6.43: Aggregate throughput (Random topology 2, 11 Mbps channel rate, exponential distribution)

Figure 6.44: Network utility (Random topology 2, 11 Mbps channel rate, exponential distribution)

Figure 6.45: Fairness Index (Random topology 2, 11 Mbps channel rate, exponential distribution)

Figure 6.46: Average buffer occupancy per node (Random topology 2, 11 Mbps channel rate, exponential distribution)

Figure 6.47: Average network delay (Random topology 2, 11 Mbps channel rate, exponential distribution)

Figure 6.48: Aggregate throughputs (Random topology 2, 4 flows, Different channel rates)

Figure 6.49: Random topology 2, different channel rates, exponential distribution

Figure 6.50: 4 node Linux testbed topology

Figure 6.51: Aggregate throughput, Linux testbed, 11 Mbps channel rate

6.3 Linux testbed results

Experiments on a real test bed were carried out on 4 nodes running Linux. The topology is shown in figure 6.50. Two flows run from node 1 to nodes 3 and 4 through node 2. Figure 6.51 shows the aggregate throughputs with the 2 strategies: PDQ and PDQ+Aggregation. Tests were carried out for 3 different packet sizes: 50, 100 and 200 bytes. As expected, PDQ+Aggregation scheme yields higher throughputs compared to only PDQ strategy. It would be interesting to carry out experiments on a testbed consisting of multiple nodes and then comparing the results.

Chapter 7

Summary and future work

In this thesis, we presented a scheme of packet aggregation based backpressure scheduling in wireless multihop networks. Our goal was to incorporate the tool of packet aggregation in the backpressure based scheduling technique and to gain throughput benefits without sacrificing the fairness index of the network. We simulated our scheme in OPNET and also implemented it on a testbed running Linux kernel. Our results show that our scheme of packet aggregation based backpressure scheduling achieves better aggregate throughput than just the backpressure based scheduling scheme without causing a degradation in the fairness index. Also the other important factors like average network delay and congestion at intermediate nodes are improved with the help of our aggregation scheme. We studied our scheme on different topologies.

We do not explicitly consider the effects of noise in the background. The effects of noise could be detrimental to the aggregation scheme. This is because, the presence of noise increases the bit error probability and hence the probability of packet errors increases with the size of the packet. Thus in that case, a threshold needs to be determined which balances the benefits and drawbacks of aggregation by choosing a proper packet size. This is one of the directions.

Also another interesting direction would be to analyse how this scheme will affect the processing of the MAC chip (NIC card). With aggregation, the MAC chip is given less burden of processing and appending MAC headers for many small IP packets. This reduces the work of the MAC chip thereby making it faster. This type of study has been done by authors of [25] in the context of core routers in wired networks.

REFERENCES

- [1] <http://www.opnet.com>.
- [2] <http://www.netfilter.org/>.
- [3] <http://madwifi-project.org/>.
- [4] <http://hop.cs.umass.edu/>.
- [5] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445 – 487, 2005.
- [6] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. pages 619 –627, april 2008.
- [7] R. Bruno, M. Conti, and E. Gregori. Mesh networks: commodity multihop ad hoc networks. *Communications Magazine, IEEE*, 43(3):123 – 131, march 2005.
- [8] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1 – 14, 1989.
- [9] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 14(2):70–87, 2007.
- [10] Laura Galluccio, Andrew T. Campbell, and Sergio Palazzo. Concert: aggregation-based congestion control for sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 274–275, New York, NY, USA, 2005. ACM.
- [11] S. Ganguly, V. Navda, Kyungtae Kim, A. Kashyap, D. Niculescu, R. Izmailov, Sangjin Hong, and S.R. Das. Performance optimizations for deploying voip services in mesh networks. *Selected Areas in Communications, IEEE Journal on*, 24(11):2147 –2158, nov. 2006.
- [12] Thomas F. Herbert and Thomas F. Herbert. *Linux TCPIP networking for embedded systems*. Charles River Media, 2007.
- [13] A.P. Iyer, G. Deshpande, E. Rozner, A. Bhartia, and Lili Qiu. Fast resilient jumbo frames in wireless lans. pages 1 –9, july 2009.
- [14] Ashish Jain, Marco Gruteser, Mike Neufeld, and Dirk Grunwald. Benefits of packet aggregation in ad-hoc wireless network. Technical report, 2003.
- [15] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

- [16] S. Keshav and Rosen Sharma. Issues and trends in router design. *IEEE Communications Magazine*, 36:144–151, 1998.
- [17] Wonsoo Kim, Hyrum K. Wright, and Scott M. Nettles. Improving the performance of multi-hop wireless networks using frame aggregation and broadcast for tcp acks. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–12, New York, NY, USA, 2008. ACM.
- [18] Dzmitry Kliazovich and Fabrizio Granelli. Packet concatenation at the ip level for performance enhancement in wireless local area networks. *Wirel. Netw.*, 14(4):519–529, 2008.
- [19] Ming Li, Devesh Agrawal, Deepak Ganesan, and Arun Venkataramani. Block-switched networks: A new paradigm for wireless transport.
- [20] Liang Ma and M.K. Denko. A routing metric for load-balancing in wireless mesh networks. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 2, pages 409–414, may 2007.
- [21] Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, Lothar Stibor, Cf poll Contention, and Free Poll. Ieee 802.11e wireless lan for quality of service.
- [22] C. Na, J.K. Chen, and T.S. Rappaport. Measured traffic statistics and throughput of ieee 802.11b public wlan hotspots with three different applications. *Wireless Communications, IEEE Transactions on*, 5(11):3296–3305, november 2006.
- [23] R. Raghavendra, A.P. Jardosh, E.M. Belding, and Haitao Zheng. Ipac: Ip-based adaptive packet concatenation for multihop wireless networks. pages 2147–2153, 29 2006-nov. 1 2006.
- [24] R. Riggio, D. Miorandi, F. De Pellegrini, F. Granelli, and I. Chlamtac. A traffic aggregation and differentiation scheme for enhanced qos in ieee 802.11-based wireless mesh networks. *Comput. Commun.*, 31(7):1290–1300, 2008.
- [25] David Salyers. Jumbogen: Dynamic jumbo frame generation for network performance scalability, univ. of notre dame. *Univ. of Notre Dame*, 2006.
- [26] Diane Tang and Mary Baker. Analysis of a local-area wireless network. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 1–10, New York, NY, USA, 2000. ACM.
- [27] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. pages 2130–2132 vol.4, dec 1990.
- [28] L. Tassiulas and A. Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. *Information Theory, IEEE Transactions on*, 39(2):466–478, mar 1993.
- [29] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.

- [30] A. Warrior, S. Janakiraman, Sangtae Ha, and I. Rhee. Diffq: Practical differential backlog congestion control for wireless networks. pages 262 –270, april 2009.
- [31] Tsai-Wei Wu and Hung-Yun Hsieh. Interworking wireless mesh networks: Performance characterization and perspectives. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 4846 –4851, nov. 2007.
- [32] Hongqiang Zhai and Yuguang Fang. A distributed packet concatenation scheme for sensor and ad hoc networks. pages 1443 –1449 Vol. 3, oct. 2005.