2008

Load estimation in IEEE 802.11 wireless networks

Aditya Vasudev Dhananjay
Iowa State University

Load estimation in IEEE 802.11 wireless networks

by

Aditya Vasudev Dhananjay

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Lu Ruan, Major Professor
Soma Chaudhuri
Daji Qiao

Iowa State University

Ames, Iowa

2008

DEDICATION

I would like to dedicate this thesis to my entire family, without whose support I would not have been able to complete this work. I would also like to thank my friends for their loving guidance during the writing of this work. In particular, I would like to thank Anand Ramaswamy and Rohan Murty for being there, when I needed them the most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Lu Ruan for her guidance, patience and support throughout this research and the writing of this thesis. She has given me the freedom and encouragement to work on what interests me, while mentoring me through the process. Her insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Soma Chaudhuri and Dr. Daji Qiao. I would also like to thank Dr. Manimaran Govindarasu, whose words of encouragement motivated me through hard times.

This work would not have been possible without Varun Srinivas, who worked with me on the testbed implementations. His sharp mind and systems building skills made working with him a pleasure.

I would also like to mention our graduate secretary Linda Dutton whose kind words, cheerful smile and endless supply of chocolates made my graduate school experience something I will always cherish.

CHAPTER 1. OVERVIEW OF IEEE 802.11 WIRELESS NETWORK MANAGEMENT

IEEE 802.11 is a set of standards for wireless local area network (WLAN) computer communication, developed by the IEEE LAN/MAN Standards Committee (IEEE 802) [1]. Based on the physical layer technology used, it can be further categorized into 802.11a, 802.11b and 802.11g.

The 802.11a standard uses the same core protocol as the original standard, operates in 5 GHz band, and uses a 52-subcarrier *orthogonal frequency-division multiplexing* (OFDM) with a maximum raw data rate of 54 Mbps, which yields realistic net achievable throughput around 20 Mbps. The 802.11b standard is a direct extension of the *Direct-sequence spread spectrum* (DSSS) modulation technique defined in the original standard. On a carrier of 2.4 GHz, it uses *Complementary code keying* (CCK) as its modulation technique. 802.11b has a maximum raw data rate of 11 Mbps, with throughputs of up to 7.1Mbps in practice. The 802.11g physical layer is designed to operate using OFDM at a carrier frequency of 2.4 GHz. The maximum data rate is 54 Mbps, while in practice, it is possible to reach throughputs of up to 20 Mbps.

IEEE 802.11 networks have rapidly gained popularity, largely due to their low cost and ease of deployment [2][3]. They are now widely deployed in homes, corporate offices and schools, to offer wireless network connectivity and internet access [4][5].

Like large deployments of any kind of network, resource management is an important consideration in 802.11 networks [6][7][8][9][10][11][12][13][14][15][16][17]. In order for these networks to utilize the available spectrum efficiently, we need to adapt its behavior in response to the *load*. The work presented in this thesis is a set of techniques that are used for **load estimation**, such that it can be used for network management and performance optimization.

Apart from traditional management of the network's operational parameters, load estimation also has applications in domains such as cooperative scheduling [2].

The remainder of this chapter is organized as follows. In section 1.1, we introduce the modes of operation of IEEE 802.11 wireless networks. In section 1.2, we describe the two protocols that form the 802.11 MAC layer. In section 1.3, we describe the motivation behind our work, by emphasizing the importance of load estimation in network management. We briefly explain how large 802.11 networks are deployed, and describe a few techniques for network management. In section 1.4, we outline the rest of the thesis.

1.1 IEEE 802.11 Operational Modes

The IEEE 802.11 standard defines two modes of operation:

- *Infrastructure Mode:* This is the most commonly deployed mode of operation of 802.11 networks. They are characterized by the use of *access points*. Typically, these access points are connected together using a wired backbone network [18]. Access points are nothing but bridges that operate between the wired and wireless sides of the network. Wireless stations (such as laptops and PDAs) access the network, with the access point as the intermediate hop. The work presented in this thesis is focused on infrastructure networks.

- *Ad-HocMode* In this mode, stations are allowed to communicate directly with one another. Networks can be set up and torn down without the need of any backbone or infrastructure. These networks are typically active for short periods of time, and torn down when the stations no longer need to communicate with each other.

1.2 IEEE 802.11 MAC Protocol

The IEEE 802.11 standard [1] defines two medium access control (MAC) protocols, namely the Distributed Coordination Function (DCF) and the Point Coordination Function (PCF). These are described below:

1.2.1 Distributed Coordination Function (DCF)

This is a contention based protocol, where stations compete against each other for the chance to transmit their frames over the shared wireless channel. The DCF is primarily based on CSMA/CA (Carrier sense multiple access, with collision avoidance). When a station wants to transmit a frame, it senses the channel to check if it is busy or idle. This channel sensing is done in two ways - *physical carrier sensing* and *virtual carrier sensing*. If the channel is busy (as indicated by either the physical or virtual carrier sensing), it waits for the channel to become idle. Once the channel remains idle for a period of $DIFS$, the station runs the backoff procedure. It chooses a slot between 0 and CW, and begins counting down to 0. The station which chose the smallest slot is the one which counts down to 0 the earliest. This station wins the contention and transmits its frame.

The receipt of a positive acknowledgment is the only way a station knows if its transmission was successful or not. If the station fails to receive an acknowledgment (ACK), it assumes that the frame transmission has failed due to a collision. A collision is caused when two or more stations chose the same smallest slot for transmission. Upon failure to receive the ACK, the station then proceeds to retry the transmission.

The initial value of CW is CW_{min} , which is usually 31. Upon every frame transmission failure (failure to receive the positive acknowledgment), it doubles the CW size, up to a maximum of CW_{max} , which is usually 1023. The new CW size is used for the next retransmission. Upon successful frame transmission, it immediately reduces the CW size to CW_{min} . This is known as the Binary Exponential Backoff (BEB) algorithm. The motivation behind such a backoff algorithm is the following: When a frame transmission fails, the station assumes that the failure was caused by a collision. The occurrence of collisions is closely related to the amount of congestion in the network. The CW size is doubled, in order to decrease the probability of subsequent slot choices coinciding, and hence collisions taking place.

1.2.1.1 Exponential Increase Exponential Decrease (EIED)

One of the weaknesses of the legacy 802.11 DCF is caused by the fact that stations reset their CW size to CW_{min} upon a successful frame transmission. For example, assume that a lot of collisions have previously taken place, and hence the stations use high CW sizes. If the stations reduce their CW sizes to CW_{min} upon a successful transmission, it increases the probability of more collisions taking place in the immediate future. This also leads to unfairness in channel access. To overcome these weaknesses, the authors in [19] propose a MAC modification called EIED, where a successful frame transmission leads to stations reducing their CW size by half, instead of resetting it to CW_{min}. However, if the CW size is already CW_{min}, then it is left unchanged. This solution leads to higher channel throughput, and greater fairness.

1.2.2 Point Coordination Function (PCF)

The PCF enables the transmission of frames that have a time bound for transmission. This protocol is centrally controlled by an entity known as the *point coordinator*, which typically resides in the access point. It controls medium access, by determining which station is allowed to access the medium at any point of time. Time is broken into alternating periods of *contention free period* and the *contention period*. The point coordinator maintains a list of stations, which it needs to poll during the contention free period. For example, the point coordinator may first poll station A, and specify a certain time interval for which A has monopoly over the channel; no other station is allowed to access it. By implementing a round-robin scheduling algorithm, the point coordinator can give all stations an opportunity to transmit without contention.

This feature is particularly useful when stations have time-constrained data frames to transmit; for example voice and video data. Stations that do not run the PCF wait for the contention period, where they compete to access the channel, using the same mechanism as the DCF. However, the PCF is rarely implemented, while the DCF is implemented by all 802.11 compatible devices.

1.3 Motivation: Management of Large 802.11 Wireless Networks

IEEE 802.11 based wireless networks often have large deployments, such as in campus and corporate installations. Such networks are almost always run in the *infrastructure* mode. Since each access point has only a finite radio range and bandwidth capacity, these networks are composed of a large number of access points, which are connected together using a wired backbone. The radio range of each access point constitutes its individual *cell*. In order to provide ubiquitous coverage, the access points are placed in such a way that their cells are partially overlapping, such that a mobile station is always in the cell of at least once access point, through which it can access the network. There are three principal techniques (association control, channel selection and power control) used to manage and optimize operational parameters used by the access points, such that the bandwidth utilization in the network is maximized [6][7][8][9][10][12][13][14][15][16].

1.3.1 Association/Admission Control

Recent studies [20][3][9][13][14][15][21][22] have shown that in many networks, the *load* is unevenly distributed amongst access points. Based on the IEEE 802.11 standard, it is the duty of the wireless station to choose which access point it wants to associate with. During this process, the station *scans* all available channels to see which access points it is in the range of, along with their respective capabilities. From among the access points that satisfy the required capabilities, the station associates with that access point from which it receives the highest signal strength. This strength is given by the *received signal strength indicator*, or *RSSI* [5].

Due to this behavior, it is common to see one access point A having a very high load, while the other neighboring access points going virtually unused. For example, if there are many users with their laptops in a conference room, there is a high probability that all the laptops will associate with the same access point. This leads to very high contention at the overloaded access point, which in turn leads to sub-optimal resource utilization. However, if a few of the stations associated with A had in fact associated with some of the neighboring access points,

the overall utilization would have been higher, due to reduced contention at the individual access points.

One method of overriding the default association behavior of the stations is to modify the code on device drivers of the stations. For example in [10], the client driver is modified to recognize additional fields in the beacon frame, which aid in the association decisions being made. There also exist other techniques (like for example, [8]) whereby the default association behavior of the stations can be overridden, without making changes to the client code. In this approach, the access points are connected with a *controller*, which resides in the wireless backplane. The controller maintains a list (which is constantly updated) about which stations are in the range of which access points, along with the respective RSSIs. When the controller decides that a station A should associate with an access point B, then B is instructed to respond to A's probe request messages, while other access points ignore the probe requests. In this method, only one access point is *visible* to the station, and it associates there.

Sometimes, the goal of association/admission control is to provide *MaxMin* fairness, as opposed to blindly maximizing the network throughput. An allocation of bandwidth by means of client-AP associations is MaxMin fair, if it is impossible to increase the available bandwidth of a user A without decreasing the available bandwidth of a user B with equal or less bandwidth. Examples of such work appear in [18] and [13]. Since association control is such an invaluable tool in network management, much work [23][24][25][26][27] has focused on how fast handoffs can be performed in a variety of scenarios. This enables association control to be executed on a fine-grained time scale.

In order for any load-aware association control algorithm[1] to function properly, it is obvious that the load estimates generated by the individual access points must be meaningful. Using weak techniques for load estimation can lead to association control decisions that may exacerbate the problem of high contention.

[1]There exist many approaches, where the clients control the associations. However, we focus on algorithms which are controlled by the access points and a network controller. These are the algorithms where no changes to the code on client machines (to make association control decisions) are needed.

1.3.2 Frequency / Channel Selection

Considering 802.11b/g, there are 11 channels (numbered 1 through 11), which are spaced 5MHz apart. In order for 2 channels to be completely non-overlapping, they have to be at least 25 MHz apart [10][8]. Therefore, 802.11b/g has 3 non-overlapping channels, namely 1, 6 and 11 [28]. Ideally, we would like to have neighboring cells to be on orthogonal channels, so that there is no mutual interference [4][29]. However, since network deployments are dense, this is not always possible.

In areas of the network where the load is very high, we would like to operate the access points on as *far-apart* channels as possible, whereas it is alright for access points in lightly loaded areas to operate on partially overlapping channels. The channels used by the individual access points can be changed on-the-fly, in response to changing network usage [4][30][31][32][14][5]. Clearly, channel assignment for access points depends on the ability of the access point to generate meaningful load estimates.

1.3.3 Power Control

As noted in [28], typical 2.4 GHz and 5 GHz transmit power, which is regulated by the Federal Communications Commission (FCC), is around +17 to +30 dBm (50 mW to 1 W). Typical radio receiver demodulation maximum sensitivity is around 90 dBm for base data rate modulations and all data packet headers for 802.11a/b/g. Antenna gain maximums are limited by the FCC. If all access points are operated in the default setting, they all use the same transmit power. This means that the *cell-size* of each access point is more or less time invariant.

As noted in [33][34], the IEEE 802.11 standard currently merely provides a mechanism for association and disassociation. No information regarding access point load is provided; as a result, the only viable policy for access point selection is to pick the access point with the best signal to noise (SNR) ratio.

It is a very common problem, where an access point A is heavily loaded, while a neighboring access point B is more or less unused [33]. Suppose we decrease the transmit power of access

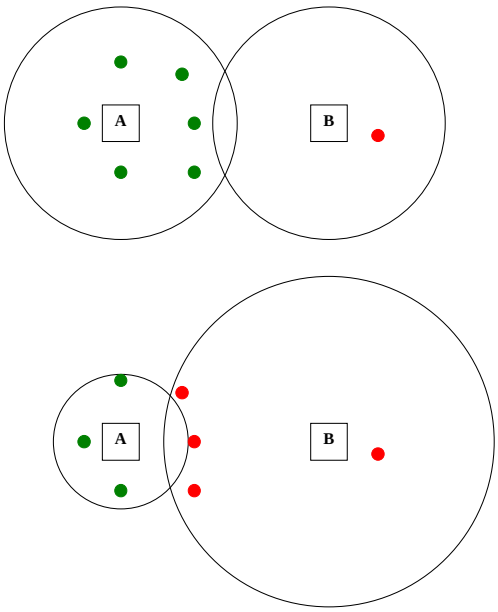

Figure 1.1 Example for Power Control

point A, and increase that of access point B. This means that the *cell* of A becomes smaller, while that of B becomes larger. Based on the default association behavior of the 802.11 stations, this will automatically force a few stations that are associated with A to reassociate with B, thereby achieving some degree of load balancing. This technique forms the foundation of what is known as *cell breathing* [35]. Implementations and adaptations of this technique are available in [8][10][17][29].

For example, consider figure 1.1. Initially, most of the stations are associated with access point A, while B has a low load. For simplicity, assume that each station has the same bandwidth requirement. Initially, both access points transmit with the same power. Upon decreasing the transmit power of A and increasing that of B, the figure shows how reassociations take place. This load balancing has the effect of reducing the contention at A, while increasing the bandwidth utilization at both access points A and B. Clearly, power control for the access points depends on the ability of the access point to generate meaningful load estimates.

1.4 Outline of the Thesis

In chapter 2, we take a look at existing load estimation metrics and estimation techniques, and study a few simple case scenarios where they fail to provide meaningful load estimates. In chapter 3, we present *PigWin*, which is a load estimation technique that can be used to generate more meaningful load estimates in the context of network management. Through simulations in NS-2 [36], we demonstrate how the load estimate varies in response to changing network usage patterns. We also argue why *PigWin* generated load estimates are more useful in the context of network management functions such as association control and channel control across access points in large infrastructure networks. However, the weakness of *PigWin* is that it requires modification to the 802.11 client code, and it measures only the load in the uplink direction.

In chapter 4, we provide another set of techniques to estimate the load. These techniques do not need any modification to the 802.11 client code. Furthermore, the *load* in the uplink and downlink directions is independently measured using different metrics and techniques. We demonstrate their behavior to changing network usage, via simulations in NS-2. We also explain why these load estimates are better suited for network management functions.

In chapter 5, we describe our work on an actual testbed of 802.11 nodes. This work is built upon the load estimation techniques presented in chapter 4. Using these load estimates, we have implemented an association control algorithm that maximizes the total available throughput in the network. We describe our load estimation techniques, the system architecture and finally, the performance.

In chapter 6, we conclude this thesis and briefly describe future directions of work.

CHAPTER 2. RELATED WORK IN LOAD ESTIMATION

There are two main approaches to load estimation in wireless networks, depending on the type of application that uses the load estimate. The *offline* approach is used by applications that do not need a real-time load estimate. For example, a network administrator might want to analyze the network usage patterns over one month, to determine *hot-spots*; the admin can then deploy additional access points in the area to overcome persistent congestion. The *online* approach is used for real-time applications such as association control. In this case, depending on the *current* network usage, the associations of stations to access points can be adjusted to improve network performance.

As mentioned in [37], several WLAN workload studies [21][22][38][39][40] performed *offline* analysis on the trace of a building-wide or a campus-wide wireless network, including syslogs, SNMP, and `tcpdump`. The major goal of these efforts is to study the user behavior, traffic characteristics, usage patterns and network performance in a long term, and then to apply the analysis results into the research of new wireless application development, future WLAN deployment and management.

In this chapter, we take a look at existing load estimation metrics and *online* estimation techniques, and study a few simple case scenarios to understand their behavior. In the literature, there is not much work focused on load estimation in itself. However, there are many papers that define *load* in some way, and then use this load estimate as an input to their network management subsystem. In this chapter, we take a closer look at how the related work defines load, and through simple examples, explain why they do not provide meaningful load estimates. In order to demonstrate their inapplicability in network management applications, we use *association control* as an example network management function.

2.1 Load as the Number of Stations

Some older papers [6][7] equate load to the *number of stations in the Basic Service Set (BSS)*. This definition may have been meaningful if we assume that each station has the same traffic pattern, and hence the same bandwidth requirement. However in reality, there is no correlation between the *work done* by two access points in terms of the number of stations associated with each of them, because some stations might be actively transmitting, while others remain dormant [22][41]. Even among the active stations, they might differ in their bandwidth requirements.

For example in Fig. 2.1, there are two access points A and B. Access point A has stations v_1, v_2, v_3, v_4 and v_5 associated with it. Station v_1 is sparingly using the network, while v_2, v_3, v_4 and v_5 are inactive. Access point B has three stations w_1, w_2 and w_3 associated with it - all of which are actively contending for the channel. Going by this definition of *load*, we incorrectly conclude that access point A is handling a higher load than B. Consequently, any load balancing or association control algorithm is going to force reassociations/new associations to access point B, thereby further exacerbating the problem.

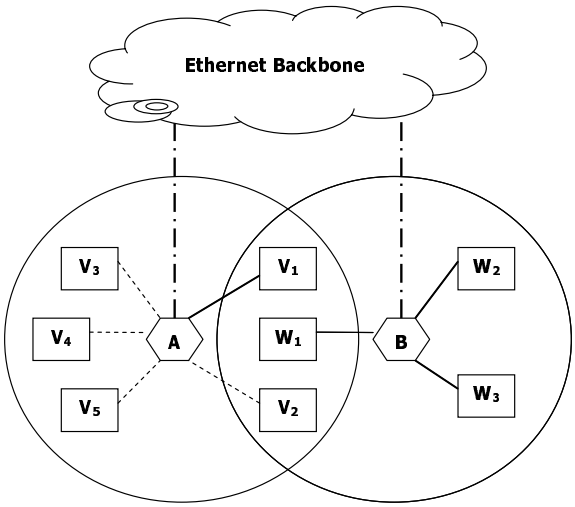

Figure 2.1 Example Topology 1

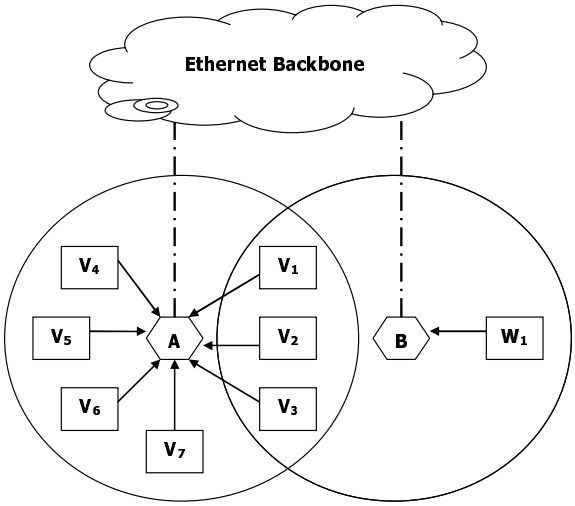

Figure 2.2 Example Topology 2

2.2 Load as the Number of Frames

Newer papers [10][12][37][16] define *load* as the *number of frames that the access point successfully handles per unit time*. Variants include defining load as the *bandwidth demand* made by the stations on the access point. For simplicity, we assume that all frames are the same size.

Consider Fig. 2.2, where there are two access points A and B, operating on orthogonal frequencies. Access point A has many stations associated with it, all of which are actively contending for the channel. Consequently, many collisions occur at A, which leads to decreased throughput. Access point B has just one station w_1 associated with it, which is transmitting its data without any contention. Hence, w_1 is capable of successfully sending a large number of frames per unit time to B. By this definition of *load*, B has a higher load than A.

If an association control algorithm uses this load estimate, a new station x that can associate with either A or B, will be incorrectly forced to associate with A. Clearly, this will lead to further contention, congestion and collisions taking place in the cell of access point A.

Another weakness of this definition of *load* is caused by the multi-rate capabilities of 802.11 a/b/g/n [42][4][43]. Different stations may talk to the access point at different data rates, be-

Index	Rate (Mbps)	SINR (dB)
1	54 Mbps	24.6
2	48 Mbps	24
3	36 Mbps	18.8
4	24 Mbps	17
5	18 Mbps	10.8
6	12 Mbps	9
7	9 Mbps	7.8
8	6 Mbps	6

Table 2.1 SINR Requirements for Different Data Rates

cause they each perceive the channel quality differently. When the channel noise or interference is high, stations jump to a lower data rate, because the modulation at lower rates is more robust. The variations in channel quality could also be attributed to the varying distances of the stations from the access point [29]. The varying distances have the effect of causing variations in the RSSI at the receiver. Therefore for a given RSSI, there is an *optimal* data rate to use. This is illustrated in table 2.1, which has been obtained from [4] and[17].

The time taken for a frame transmission depends on the data rate. Thus, the data rate used by the stations also has an effect on the load at the access point. Consequently, a load of say, x frames per second might indicate congestion if the data rate is 1Mbps, but the same load might imply low channel utilization if the data rate is 54Mbps.

2.3 Load as the Percentage of Busy Time

Another definition of *load* is the *percentage of time that the channel is busy* [8]. This load estimation is done by the access point, by simple physical layer carrier sensing. If the received signal strength is greater than a certain threshold, then the channel is deemed to be busy, and idle otherwise. This threshold is known as the *clear channel assessment* threshold, or CCA.

This metric improves on defining load as the *number of frames handled per unit time*, because collisions also affect the load estimate. We know that when a collision takes place at the receiver, the $RSSI$ at the receiver is definitely greater than the CCA. However, even this

metric/technique has its weaknesses.

From Fig. 2.2, the busy time for access point B is greater, because a lot of time is wasted in BSS A by backoff delays and large *contention window (CW)* sizes (explained in section 1.2.1). As a result, the load estimation algorithm will incorrectly tell us that B has a higher load than A. In [37], the authors present a passive load estimation technique called *Wintry*, which decomposes time and identifies busy periods by estimating the backoff time values.

In summary, the traditional definitions of *load* (explained in sections 2.1, 2.2 and 2.3) fail to capture an intuitive notion of the actual conditions in the BSS. Consequently, any load estimation technique based on these metrics of load will yield inaccurate results. Informally, we would like to define *load* of an access point to capture an intuitive notion of the *amount of contention* in its cell.

2.4 ProbeGap: Load Expressed in Terms of the Available Bandwidth

2.4.1 Introduction

ProbeGap [44] is a technique that is used to estimate the capacity and *available bandwidth* of network paths. Previous work has focused on networks where links have a well defined bandwidth, and packets/frames follow a first in first out (FIFO) ordering. However, this model breaks down in many ways in 802.11 wireless networks, as explained below:

- Uncertain Bandwidth: We know that the maximum bandwidth of a link is 11Mbps for 802.11b, and 54Mbps for 802.11a/g. However, this is the *raw* bandwidth, which is significantly greater than the available bandwidth for the stations. The reasons are the following:
  - There is an overhead for every frame, because the *physical layer convergence protocol* (PLCP) and *physical medium dependent* (PMD) headers are transmitted at the base rate of 1Mbps [1]. Only the payload portion the frame is allowed to be transmitted at the higher rate [28].

  - There is a fixed minimum idle time (DIFS) that is required between frame transmissions, followed by a variable wait time used for contention resolution.
  - Since there are overheads on a per frame basis, the available useful throughput is a function of the frame size.
  - The usable data rate between the stations and the access point is a function of the noise level, which in turn depends on the distance between them. For example, if the station is 1m away from the access point, then communication between them can take place at the full data rate. However, if they are far away, the bit error rate (BER) increases, which in turn increases the packet error rate (PER). Rate control features such as *automatic rate fallback (ARF)* determine the best data rate to use. Lower data rates are associated with more robust modulation schemes, leading to the BER (and hence PER) being decreased. The data rate used determines the available bandwidth, and it depends on transient network conditions. Hence, 802.11 links do not have a well defined bandwidth.

- Non FIFO Ordering: Clearly, contention based MAC protocols do not offer FIFO packet ordering.

2.4.2 Main Idea of ProbeGap

The idea is to estimate the fraction of time that a link is idle by probing for *gaps* in the busy periods, and then multiplying by the capacity to obtain an estimate of the available bandwidth. However, this capacity estimation is susceptible to errors because of rate adaptation [42].

ProbeGap estimates the idle time fraction by gathering samples of one-way delay (OWD) over the link, as explained in [44]. The authors argue that the distribution of OWD samples show two distinct regions, one corresponding to an idle channel, and the other corresponding to a busy channel. The knee in the CDF of the OWD samples is used to calculate the idle time fraction. Note that the idle time is not calculated by simple physical layer carrier sensing, because based on their approach, hidden stations are a source of error. [1]

[1] Parts of the discussion in this section have been quoted directly from [44]

The sender sends a series of Poisson-spaced probes, each with a 20-byte payload containing a local timestamp. The receiver subtracts the senders timestamp from the received time to compute the OWD. The OWD is then *normalized* by subtracting out the smallest OWD from the set of measurements, so that the smallest normalized OWD in the set is zero. The sender and receiver clocks need not be synchronized; they just need to maintain a constant offset.[2] The intuition is that if a probe finds the link to be free, it would experience a small OWD. On the other hand, if it needs to wait for packets in transmission or ahead of it in the queue, it would experience a larger OWD. In other words, large OWD values imply that the channel idle fraction is smaller. This interpretation of *load* captures the effects of both uplink and downlink load components together.

Now, let us consider how *ProbeGap* can be adapted to the case where the access points estimate their idle time, expressed as a function of the OWD. Subsequently, the available bandwidth can be estimated by multiplying the idle time with the channel capacity.

On the receiver side, timestamping is done by the device driver itself, when the packet is received. That way, the overheads of packet processing in the protocol stack are minimized. When the load estimation needs to be done, the following steps are taken, such that no code modifications are needed on the device driver of client stations:

- A probe message is inserted into the transmit queue of access point A, and the timestamp is noted.
- Access point A sends the probe message to access point B in its range, and piggybacks this time stamp on the probe frame. The DCF is used for channel arbitration during this process.
- Access point B receives the probe messages and calculates OWD. It then estimates the idle time for A, as a function of the OWD. It then calculates the available bandwidth by multiplying the idle time with the channel capacity.

[2]Compensating for clock skew is a very well studied problem in distributed systems, and there exist many solutions for the same.

- Access point B tells A the available bandwidth at A through a response frame over either the wired or wireless channel.

It is possible to also perform the OWD estimation slightly differently (and more conveniently), as explained below:

- A probe message is inserted into the transmit queue of access point A, and the timestamp is noted.
- An access point A sends the probe message to access point B in its range, using the standard 802.11 DCF MAC protocol.
- Access point B receives the probe message and sends an ACK to A.
- When the ACK is received, the time difference is calculated as the OWD. We note that the OWD calculated is *correct*, because B sends the ACK exactly $SIFS$ after it received the probe.

2.4.3 Weaknesses

We know that both A and B need to be on the same channel in order to communicate. If A had to jump channels to talk with B, then the contention in A's cell is not measured, and this is clearly not the behavior that we want. In real networks, it is rare to have two access points on the same channel, that are in direct radio range of each other. Hence, this method of active probing may not always work.

The *normalization* process has its own associated sources of error. For example, the OWD is not a function of the *network contention* alone. There are also variability in operating system delay, which is not guaranteed to be eliminated through the normalization process.

Moreover, it is very common for the network to have heterogeneous devices, in terms of the processing power and memory of the constituent systems. To overcome this condition, the normalization should be done on a per-node-pair basis to reduce variations caused by node heterogeneity.

The normalization error can be minimized by sending a large number of probe packets. However, if we want a more accurate load estimate, then the overhead is that we need to transmit a large number of probes. This has the potential to waste network bandwidth, which in itself, is a very precious resource.

2.5 Potential Bandwidth Estimation

2.5.1 Introduction

This approach has been presented in [34].[3] The authors identify *potential bandwidth* as the metric based on which hosts should make association decisions, and define it as the (MAC-layer) bandwidth that the client is likely to receive if it were to associate with a particular AP.

2.5.2 Main Idea

This approach is non intrusive, and relies solely on passive measurements. The approach does not require a station to associate with an access point in order to estimate the bandwidth it would receive from the access point. This allows the station to simultaneously evaluate the potential bandwidth to multiple access points in its range. However, the limitation is that it can estimate the bandwidth of only those access points that operate on the same channel that the station is operating on, because the measurements are made by passive sniffing.

The authors propose a methodology for the estimation of potential upstream and downstream bandwidth between a client and an access point based on measurements of delay incurred by 802.11 Beacon frames from the access point.

Based on the IEEE 802.11 standard, *beacon* frames are broadcast by access points periodically, and this interval is usually 1024 μs. The purpose of the beacon frames is for the access points to announce their identity as well as for the synchronization of the entire network. The delay between the time when a beacon frame is scheduled for transmission and its eventual

[3]Parts of the discussion in this section have been quoted directly from this paper.

transmission captures the load of the AP and the contention inside the network, conditions that the client would face if affiliated with that AP.

The authors present a mechanism, where the potential bandwidth can be estimated by observing the beacon delays. This is the bandwidth a client will receive from the AP, should it associate with that AP.

The important point to note is that when an access point wants to transmit a beacon frame, it must contend with all other stations in its radio range, which operate on the same channel. Therefore, if the beacon frame gets delayed by a large amount, it indicates that many stations were also trying to access the channel at that time, indicating high contention (*load*) in the uplink direction.

The authors also assume that beacon frames are not prioritized over other frames, as implemented in the access points used in their experiments. As a result, if the downlink load is high, it means that the transmit queue of the access point is long. This implies that the scheduled beacon transmission also has to wait longer before it can be sent.

Since beacon delays are computed solely based on timestamps provided by the access point, synchronization issues do not arise.

Based on what we have seen, the beacon delays are indicators of both the uplink and downlink load. *But how do the authors differentiate between the two?* For pure uplink load estimates, the authors propose the use active measurements. The IEEE 802.11 standard allows a station in an unassociated state to send data frames to an access point. The station sends data frames to an access point in the unassociated state and records the time elapsed between the instant when a frame is scheduled for transmission and the time when the end-host receives an ACK message. This delay is an accurate estimate of the contention delay, in the uplink direction.

2.5.3 Weaknesses

The drawback is that device driver modification is required on the client machines. Since there are already millions of 802.11 cards already in the market, this might not be possible.

However on the brighter side, a station with the modified device driver is fully compatible with legacy networks.

Another issue is if the access point implements prioritization of beacon frames, then the downlink load has no effect on the beacon delay. This technique will then capture only the uplink load, which generally does not capture a notion of the actual network conditions.

CHAPTER 3. PIGWIN: LOAD ESTIMATION

3.1 Introduction

PigWin [45] estimates the load by giving a measure of *how easy or difficult it is for a frame to be successfully transmitted*. We define *load* in such a way that it captures the *amount of contention* taking place in the BSS. As explained in section 1.2.1, if the stations are transmitting frames with very few collisions, then a majority of the frames will use CW_{min} as their CW size. This implies that the contention in the BSS is low. However, if there are a lot of collisions, then a sizable percentage of frames are going to be successfully transmitted using CW sizes greater than CW_{min}. If the access point *knows* the CW size used for every recent and successful uplink frame transmission, it can estimate the current load.

Hence, by aggregating recent history of what percentage of frames have been successfully transmitted using each of the permissible CW sizes, it is possible for the access point to define the current *ease of transmission* in the BSS. We note that transmission failures can be caused both by collisions, as well as by channel noise. One of the classical problems in wireless networks is to distinguish between the two. Presently, we do not make this distinction, but intend doing so in the future.

3.2 PigWin: The Protocol

3.2.1 MAC Modification on Clients

We know that in infrastructure mode WLANs, all frame transmissions take place with the access point as the intermediate hop. Consequently, the access point can receive and process any uplink frames sent by the stations.

In order to support *PigWin*, every station piggybacks on to every frame the *CW* size used for that transmission attempt. If the frame transmission fails, then the piggybacked information is lost; and this is exactly the behavior we want. We want the access point to know the *CW* size used only for *successful* frame transmissions. There are 6 permissible *CW* sizes, and hence the amount of data required to be piggybacked onto every frame is just 3 bits, which is padded up to 1 byte. This overhead is very small and hence, there is no network performance drop. As the contention ***win***dow size is ***pig***gybacked on to every uplink frame, this mechanism has been named *PigWin*.

3.2.2 Access Point Estimating Load

It is the duty of the access point (AP) to maintain recent history of the *CW* sizes used for uplink frame transmissions. The AP stores this information in a *circular queue*, with each node in the queue holding two values - a time stamp and a *CW* size. For every frame that is received, the AP records the piggybacked *CW* size (as explained in section 3.2.1) and the timestamp (when the frame was received) in the queue. If the queue is full, then the oldest entry is purged, to make way for a new entry corresponding to the latest received frame. In this manner, it retains only recent history of *CW* sizes used.

There is one parameter that influences the load estimate, which is the time difference between the current time and the recorded time stamp that determines if an entry is *recent enough* to be considered while estimating the load. For example, a *PigWin* policy can be set that enforces that entries older than 10 seconds are not *recent enough*. This value is denoted by β, and its units are seconds. When the AP needs to estimate its load, it does the following: First, it makes one pass through the circular queue and deletes all entries that are older than β seconds.

Each remaining entry in the queue corresponds to one frame that the access point has received in the *recent* past. From these entries, the AP determines (by simple counting) a set λ of values, where each λ_i is the number of entries in the queue, with *CW* size i. Since there are six permissible *CW* sizes, the AP calculates six λ_i values, for $i = \{31, 63, 127, 255, 511,$

1023}. The AP then calculates the *ease of transmission* (denoted by ϕ), which is the reciprocal of the average CW size used in the recent past.

$$\frac{1}{\phi} = \frac{\sum_i \lambda_i \times i}{\sum_i \lambda_i} \tag{3.1}$$

Large ϕ values imply that stations are using small CW sizes, and the amount of transmission failures taking place is low. Small ϕ values imply that the stations are finding it more difficult to transmit their frames. We define *load* to be the reciprocal of ϕ, where ϕ is the *ease of transmission*. This definition of *load* captures the notion of *difficulty in transmission*, and is measured by the average CW size of successful frame transmissions in the recent past. Consequently, *load* has no units. When frames are being transmitted, the minimum permissible value of *load* is CW_{min}, while the theoretical maximum is CW_{max}.

However, if there were no frames transmitted in the previous β seconds, then both the numerator and denominator of equation 3.1 will be zero. In this case, the reported load is CW_{min}, and a flag is set to differentiate this case from previously mentioned case where the load is also CW_{min}.

3.2.3 Effect of β on the Load Estimate

A small value of β implies that the piggybacked CW information becomes invalid earlier; history maintained by the access point is *more recent*. Conversely, a large value of β means that history is maintained longer. It is a well known fact that most users' internet usage patterns are highly bursty; active for short periods of time, and inactive for most of the time. As a result, if a small β value is used for load estimation during a network access burst, the reported load will be very high; even though the network was possibly inactive for a long time before the burst, and probably will become inactive soon after. Small β values reflect transient network conditions. Hence, the load estimate can fluctuate greatly.

On the other hand, large β values tend to keep stale values, which might not have any bearing to the current network status. There is no *optimal* value for β; however an acceptable value depends on the semantic of the application that uses the load estimated by *PigWin*.

3.3 Simulation Techniques and Results

NS [36] or the Network Simulator (also popularly called NS-2, in reference to its current generation) is a discrete event network simulator, targeted at research. NS can simulate many types of protocols, from across the protocol stack. We decided to use NS-2 for our simulations because it can accurately model the shared wireless channel and it can be used to efficiently simulate the network for a variety of topologies and simulation parameters [46].

We simulate an IEEE 802.11b network, operating in the infrastructure mode. The MAC layer is configured to operate with the RTS-CTS exchange for every frame transmission. The simulation parameters are shown in Table 4.1. Our simulation topology consists of one infrastructure mode BSS (one access point A, along with nine wireless stations v_1 to v_9). The access point A is connected to a station B via a backbone ethernet. Our primary goal is to observe the reported load by *PigWin* in response to changing network usage.

We simulate three different scenarios, which differ in the traffic patterns of the nine stations (described in sections 3.3.1, 3.3.2 and 3.3.3). For each scenario, we show how the reported load varies with time, for different values of β. The simulations are run for both the legacy 802.11 MAC and EIED. The load is measured every 100ms, starting at 5s and ending at 100s.

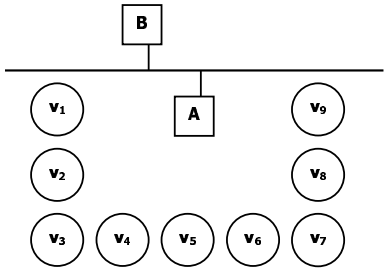

Figure 3.1 Network Topology for Simulation Scenario 1, 2 and 3

Parameter	**Value**	**Parameter**	**Value**
UDP Header	8 bytes	TCP Header	20 bytes
IP Header	20 bytes	MAC Header	36 bytes
PHY Header	24 bytes	Channel Bitrate	11 Mbps
CBR Payload	1000 bytes	CBR Data Rate	0.6 Mbps
SIFS	10 μs	PIFS	30 μs
DIFS	50 μs	EIFS	364 μs
Slot Time	20 μs	RTS size	44 bytes
CTS Size	38 bytes	ACK size	38 bytes

Table 3.1 Simulation Parameters

3.3.1 Scenario 1

This scenario aims to simulate the case when the *load* on the access point is gradually increased until it reaches saturation. Eight stations (v_1 through v_8) transmit constant bit rate (CBR) traffic to B, via individual UDP connections. However, all the transmissions are initiated 10 seconds apart. Station v_1 starts transmitting at time 5s, v_2 begins at 15s and so on, and v_8 begins transmitting at 75s. Station v_9 remains silent during the entire course of the simulation. The simulation terminates at 100s.

From Fig. 3.2, we observe the *PigWin* reported load over time, for the legacy 802.11 MAC. As the contention in the network increases, the reported load also increases, due to the increased *difficulty in transmission*. The large increase in load at 65s to 70s is due to the fact that network saturation occurs around this point. For $\beta = 1$, the reported load responds very quickly to any changes in the network usage. Also, due to the short history maintained, the load fluctuates in response to transient network conditions. For larger β values, we observe less fluctuation in load, simply because the CW sizes are averaged over longer periods of time. While large β values make the load estimate respond slower to changes in network usage, the load is more stable.

From Fig. 3.3, we observe the *PigWin* reported load over time, for the EIED protocol. The load estimate follows a pattern very similar to that seen in Fig. 3.2. However, due to the different CW sizes used by the legacy 802.11 MAC and EIED, we see that the load estimate

for EIED is higher. This is because at periods of high contention, EIED tends to use larger CW sizes to reduce the probability of collisions taking place, when compared to the legacy 802.11 MAC. This comparison between the legacy 802.11 MAC and EIED is illustrated in Fig. 3.4.

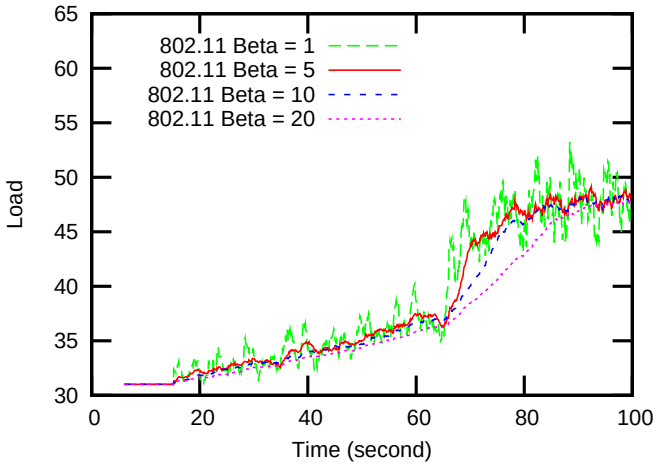

Figure 3.2 Scenario 1 - Legacy 802.11 MAC

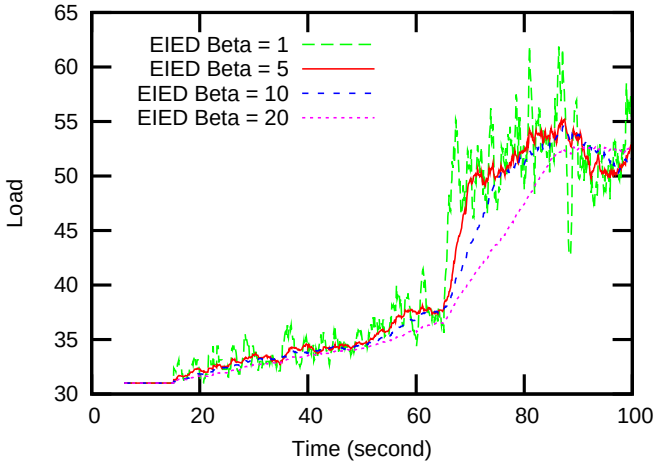

Figure 3.3 Scenario 1 - EIED

3.3.2 Scenario 2

This scenario aims to simulate a bursty network usage pattern. All nine stations transmit constant bit rate (CBR) traffic to B, via individual UDP connections. Station v_9 starts transmitting at 5s and stops at 100s. At 15s, v_1 and v_2 start transmitting, and they both stop

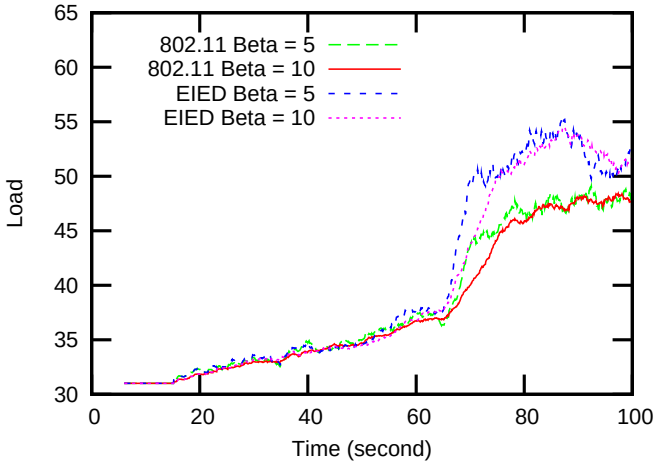

Figure 3.4 Scenario 1 - Comparison between Legacy 802.11 MAC and EIED

transmitting at 25s. At 35s, v_1, v_2, v_3, and v_4 start transmitting, and they stop transmitting at 45s. At 55s, v_1, v_2, v_3, v_4, v_5 and v_6 start transmitting, and they stop transmitting at 65s. At 75s, stations v_1 to v_8 start transmitting, and they stop at 85s. The simulation terminates at 100s.

From Fig. 3.5 (for Legacy 802.11), we observe that as the contention increases, the reported load also increases. Small values of β make the reported load respond faster to changes in *difficulty in transmission*. It is evident that the load for larger β values *trails* those of smaller β values; this is due to the fact that the history is maintained longer for larger β values. As seen earlier (for scenario 1), smaller β values tend to cause more fluctuation in the reported load, when compared to larger β values.

The results for EIED (in Fig. 3.6) are very similar to those for the legacy 802.11 MAC. However at saturation conditions, EIED uses higher CW sizes when compared to the legacy 802.11 MAC. As a result, the load reported in EIED is higher. This comparison is illustrated in Fig. 3.7.

3.3.3 Scenario 3

The third scenario aims to simulate a heavily used channel, but with no contention, and hence high *ease of transmission*. Station v_1 sends a continuous burst of FTP packets to the

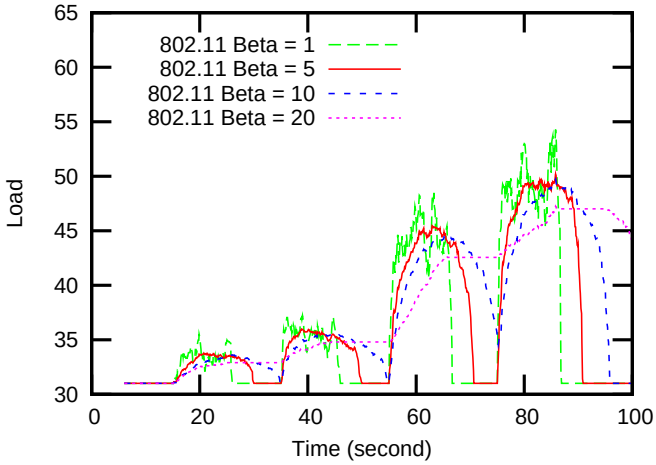

Figure 3.5 Scenario 2 - Legacy 802.11 MAC

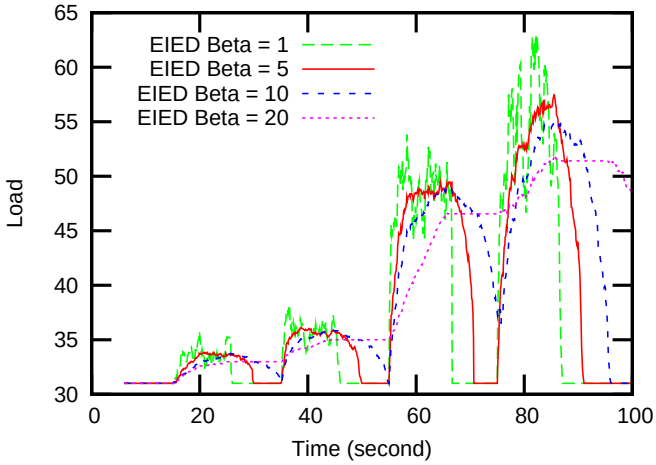

Figure 3.6 Scenario 2 - EIED

station B via the access point A. The FTP runs over a TCP connection set up between v_1 and B. The FTP connection is established at 5s and is torn down at 100s. The remaining eight stations remain inactive during the entire period of the simulation.

As seen in Fig. 3.8, the load is constant during the entire duration of the simulation. Also, we note that this constant load is the minimum permissible load, because it corresponds to the highest possible *ease of transmission*. Since there is no contention, collisions do not take place, and the CW size used is always equal to CW_{min}. It does not matter whether the legacy 802.11 MAC or the EIED is used. Additionally, the β value is of no consequence, because the CW size used does not change during the entire simulation.

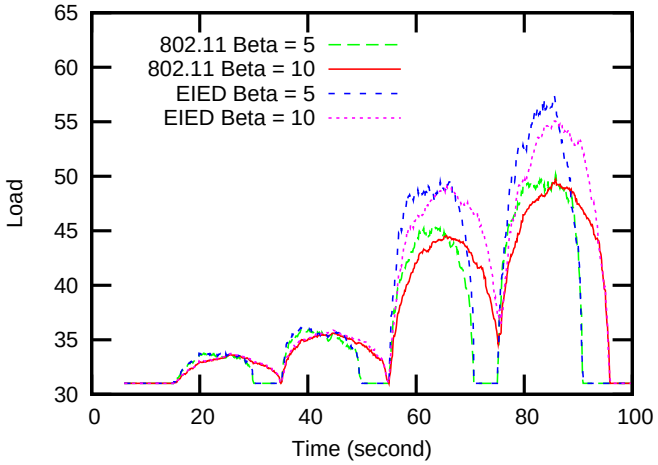

Figure 3.7 Scenario 2 - Comparison between Legacy 802.11 MAC and EIED

Through this scenario, we demonstrate that the load is independent of the number of stations in the BSS. The result would not have changed if the number of idle stations changed. Also, the load is independent of the number of frames transmitted to the access point. Clearly, the number of frames handled by the access point is very high, yet the reported load is low. It is this property of *load* that makes *PigWin* a good choice for load estimation, for use with applications such as load balancing and association control. This is our rationale behind defining *load* as the *difficulty in transmission* in the BSS.

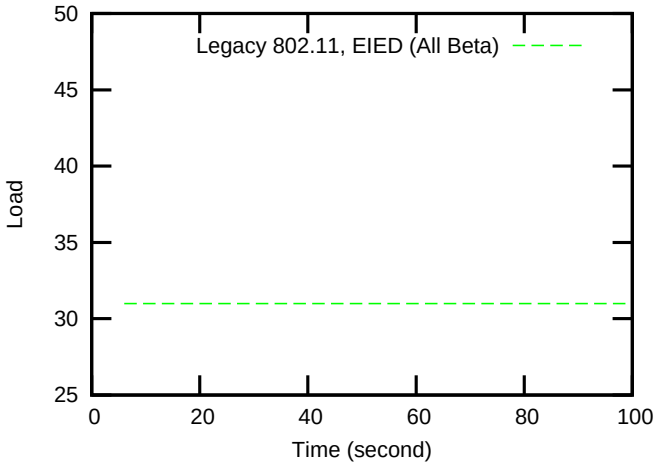

Figure 3.8 Scenario 3 - Legacy 802.11 MAC and EIED. (All β values)

3.4 Conclusion

Traditional interpretations of the term *load* have resulted in load measurement metrics, which fail to give an accurate idea of the actual BSS conditions. The redefinition of the term *load* as the *reciprocal of ease of transmission* (alternatively as the *difficulty in transmission*) gives us a more intuitive notion of the prevailing channel conditions.

The actual load estimation in *PigWin* is performed by leveraging off the behavior of the binary exponential backoff (BEB) algorithm of the DCF. Additionally, *PigWin* uses a parameter β, which can be adjusted to suit the semantic of the application which uses the generated load estimate. This leads to the potential use of *PigWin* in a variety of tasks such as load balancing, association control, network monitoring and planned ESS expansion.

It is easy to understand why the aforementioned tasks perform better with a *PigWin* generated load estimate, when compared to any of the traditional techniques mentioned in chapter 2. *PigWin* can also be applied to MAC protocol variants, such as EIED.

In the future, we plan to design a load metric that takes into account the QoS priorities of the data frames being sent. To this end, we plan to extend *PigWin* to work with the IEEE 802.11e standard [47]. In this case, the access point will generate up to 4 load estimates at any instance of time; one for each priority of data.

Since stations cannot distinguish between channel losses and collisions, the CW growth patterns are identical, irrespective of the cause of the frame loss. In the future, we want to factor in the signal to interference and noise ratio ($SINR$) for the received packet and estimate the packet error rate (PER), so we can estimate collision based losses more accurately. Furthermore, rate adaptation can affect the load estimate, since the data rate is adapted to minimize the PER. In the future, we aim to equip *PigWin* with the additional intelligence to handle these additional issues.

Finally, we are working on methods to perform load estimation without requiring any code changes on the device drivers on client machines. The work from this direction is presented in chapter 4.

CHAPTER 4. CLIENT UNAWARE LOAD ESTIMATION

4.1 Introduction

In this chapter, we propose a load estimation technique that makes two novel contributions [48]. First, we redefine *load* by splitting it into two components - uplink and downlink, which are measured using different metrics and independently of the other. Second, we present simple protocols to efficiently measure this load. These protocols require minor code changes on the access points, but the client machines remain untouched. Our techniques are applicable to the legacy 802.11 DCF as well as variations such as Exponential Increase, Exponential Decrease (EIED). Through mathematical analysis and simulations on a variety of network scenarios using NS-2, we demonstrate the performance of our load estimation techniques. We also explain why our definition of load is more meaningful when compared to traditional load estimation metrics, in the context of applications such as load balancing, association control and network management.

4.2 Load Estimation

As mentioned earlier, we split up *load* into two components - *uplink* and *downlink*, which are estimated separately. Therefore, applications such as load balancing can make more accurate decisions, on how to distribute the stations among the access points, because the load advertised by the access points is more detailed and meaningful.

4.2.1 Uplink Load Estimation

4.2.1.1 Motivation

As explained in chapter 2, traditional methods of load estimation are inaccurate. It is intuitive that a meaningful interpretation of *load* should actually consider the *amount of contention* that is taking place in the BSS, which in other words is a measure of how difficult it is for a station to successfully transmit a frame. Informally, the *amount of contention* depends on the number of stations that are actively contending for the channel. Since we do not want to modify code on client machines, there is no mechanism for the access point to know which stations are attempting to transmit at any instant of time. The access point only knows when it successfully receives a frame on its wireless interface. Additionally, the *amount of contention* should depend on the bandwidth demands that each station is making in the uplink direction. Our mechanism of estimating this *amount of contention* is based on a fundamental property of contention based protocols: *More contention leads to more collisions.*

4.2.1.2 Calculation

Let us consider an epoch of time β during which the load estimation takes place. For now, assume that an access point *knows* what fraction of time in the epoch was spent in idle listening, spent successfully receiving frames and wasted in collisions (In section 4.2.1.3, we explain how these three fractions are obtained). These three fractions are denoted by $idleTime$, $busyTime$ and $collTime$.

As explained in section 4.2.1.1, the load is proportional to the amount of time wasted in collisions for that epoch of time. The load is therefore given by:

$$Load_{Uplink} = \frac{collTime}{idleTime + busyTime + collTime} \tag{4.1}$$

The lower bound on the uplink load is 0 (when no collisions take place) and the theoretical upper bound is 1 (when all the time in the epoch is wasted in collisions).

Let us take a look at how rate adaptation affects the load estimate. We know that when a station operating at a low data rate encounters a collision, more time is wasted in the collision,

because the amount of time spent by the frame *over the air* is greater. Therefore, the reported load on that access point is higher. This behavior is exactly what we want, because it indicates that the amount of bandwidth that a new station would get (if it were to associate with that access point) is lower. Our definition of *load* is intuitive, in that it gives a good comparison between the available access points, and the type of service a station would receive if it were to associate with them.

Another interesting aspect is that many rate control algorithms rely on frame loss ratios as a factor when deciding to switch data rates [43]. The toggling of the data rate in response to collisions, in turn closes the positive feedback loop. For example, suppose many stations are close to access point A, and are all contending for the channel. The increased collisions will lead to stations choosing lower data rates, thereby making the problem of contention even worse. This implies that frames take a longer time *over the air*, and greater amounts of time are wasted in collisions. Our load estimate responds to this condition by indicating a higher load. Intuitively, we say that our load estimation technique captures the *difficulty of transmission* over the wireless channel.

4.2.1.3 Implementation

We now ask the question: How do we obtain $idleTime$, $busyTime$ and $collTime$ during every epoch? Whenever the transceiver of the access point is in the receive mode and the $RSSI$ is lower than the CCA, then the channel is said to be idle. The aggregate of this time is $idleTime$. However, when the $RSSI$ is greater than the CCA, then we really do not know if the current frame transmission is going to be successfully received or not; in other words, either a successful transmission or a collision might be going on. We store this time period as $xTime$. However, whenever a frame is successfully received, the access point knows exactly how much time that frame transmission held the channel busy. The access point keeps track of every frame successfully received in an epoch, and in particular the amount of time taken by each of these frames. This time is nothing but $busyTime$. Obviously, the difference between $xTime$ and $busyTime$ is $collTime$. these values are used to estimate the load, as explained in

section 4.2.1.2

4.2.1.4 Comparison with Traditional Load Estimation Metrics

The most important weakness of defining *load* as the *number of stations associated with the access point* is that inactive stations skew the load estimate. In our protocol, inactive stations do not affect the *amount of contention* in the BSS, and hence do not affect the load estimate. For example in Fig. 2.1, assume that stations v_1, w_1, w_2 and w_3 are all actively transmitting frames, while v_2, v_3, v_4 and v_5 are inactive. In this case, the contention (and hence collisions) at access point B is much higher than that at A, and hence our algorithm will correctly indicate that B has a higher load than A.

Consider the other definition of *load* as the *number of frames handled by the access point per epoch*. For example in Fig. 2.2, assume that all stations are actively trying to transmit their frames. As explained in chapter 2, the number of frames successfully handled by B is more than that of A, incorrectly implying that the load on B is higher. On the other hand, since the contention (and collisions) at A is much higher than at B, our protocol correctly indicates that the load on A is higher. Through these examples, we see that the load estimated by our protocol is proportional to the number of *actively transmitting* stations, and is also proportional to the *bandwidth requirements* of each of these stations. That is why our protocol is more suited for applications such as load balancing.

4.2.2 Downlink Load Estimation

4.2.2.1 Motivation

Similar to the case for uplink load estimation, we want our downlink load estimate to capture a notion of *how much the stations are competing to* ***receive*** *frames from the access point*. This means that we are interested in how the access point's transmit time is going to be divided among the associated stations. Again, we do not want inactive stations (stations that do not receive any frames) to affect our load estimate. We would like the load estimate to be proportional to the number of stations that are currently receiving frames and also be

proportional to the downlink bandwidth demand of each of these stations.

4.2.2.2 Calculation

For an access point to estimate the downlink load in an epoch of time β, it first measures n_{max} μs, which is the amount of time in the epoch that the transceiver of the access point is in the transmit state. Then for each station i that is associated with it, it measures n_i μs, which is the amount of time that the access point has spent transmitting frames destined to station i. Then, it calculates the downlink load as follows:

$$Load_{Downlink} = \prod_{i \in Stations} (1 + \frac{n_i}{n_{max}}) \tag{4.2}$$

The intuition behind this equation is the following: If there is high *internal contention* at the access point, then the amount of time the access point spends serving each of the contending stations is comparable. In other words, the n_i values have a high probability of being close together, for each contending station i. Under such conditions, our equation will result in a high load estimate. On the other hand, if there is very low internal contention at the access points, then the n_i values are going to be skewed. In this case, our equation will result in a lower load estimate.

It is easy to see that the lower bound on this downlink load is 1. This is the case when there are no frames being transmitted by the access point at all. For the upper bound on load, we assume that that there are a large number of stations that are (internally) contending to receive frames from the access point, and hence the time taken by the access point to service each station i is divided approximately equally among all the contending stations. This case represents the highest *internal contention* at the access point. Clearly, the upper bound on load in this case is when the number of stations in the network approaches infinity. In this case, fraction $\frac{n_i}{n_{max}}$ tends to 0. The equation for the theoretical upper bound on load reduces to:

$$Load^{Th-UB}_{Downlink} = \lim_{x \to 0} (1 + x)^{\infty} = e \tag{4.3}$$

However in reality, the 802.11 standard states that there can be no more than 128 stations associated with an access point at any given point. As a result, the practical upper bound of the load is equal to:

$$Load_{Downlink}^{Pr-UB} = (1 + \frac{1}{128})^{128} = 2.7077 \tag{4.4}$$

Let us take a look at how rate adaptation affects our load estimate. Assume that the access point has many stations, which are *internally contending* to receive frames from it. Assume one of the stations x communicates with the access point at a low data date (say, 1Mbps), while the others communicate with it at 54 Mbps. In this case, even though there is high internal contention, the n_i values are going to be skewed, with the n_x value being large. This will lead to a low load estimate, which is not the ideal behavior. However, if we approximate the n_i values by the number of frames received by station i in that epoch of time, then we overcome this condition. This approximation has been explained in section 4.2.2.3.

4.2.2.3 Implementation

There is an alternate simple way to approximate the downlink load estimate, that avoids the complicated procedure mentioned in section 4.2.2.2. First, let us assume that all the frames are of the same size. In this case, we can substitute n_{max} by the total number of frames transmitted by the access point, and each n_i by the number of frames transmitted by the access point to station i in the epoch. In this case, the ratio $\frac{n_i}{n_{max}}$ will not be changed, if all transmissions take place at the same data rate.

Now, let us consider what happens when the frame sizes are not the same. We argue that the load estimate will not change by a large margin. Recent studies on high rate 802.11 channels have shown that large fractions of the transmission time is spent on the low-rate (1 Mbps) PLCP and MAC headers [49]. Only the payload is transmitted at the high rate (for example, 54Mbps). As a result, differences in payload size result in considerably smaller differences in transmission times. As mentioned earlier, this approximation also has the effect of overcoming the error caused by rate adaptation. However, this is just a loose approximation; for very

accurate load estimates, we should fall back to the method described in section 4.2.2.2, and explicitly consider rate adaptation. We intend further studying the effects of rate adaptation in the future.

4.2.2.4 Comparison with Traditional Load Estimation Metrics

For simplicity, we use the optimization mentioned in section 4.2.2.3 for the load estimation. Also assume that n_{max} is equal to 2100. For example, refer to network in Fig. 2.1. Stations v_1, w_1, w_2 and w_3 want to receive a large (infinite) number of downlink frames, while the other stations are inactive. In this case, v_1 receives 2100 frames per second, while w_1, w_2 and w_3 receive 700 frames each. Going by our definition of load, the load on access point A is 2, while the load on access point B is 2.37. This reflects the fact that the amount of *internal contention* at B is more than that of A. If we had defined load to be the *number of stations in the BSS*, we would have incorrectly concluded that the load on A was higher.

For another example, consider the network in Fig. 2.2. Ignoring the direction of arrows in the figure, we see that all stations are trying to receive a large (infinite) number of frames from the wired network. Assuming n_{max} to be equal to 2100, we see that stations v_1, v_2, v_3, v_4, v_5, v_6, and v_7 receive 300 frames each, while station w_1 receives 2100 frames. If we had defined load to be the *number of frames successfully handled by the access point per unit time*, then we would have incorrectly concluded that the load on both access points were equal, because both handle a total of 2100 frames per second. However with our definition of load, we say that the load on B is equal to 2, while the load on A is equal to 2.546, reflecting the fact that the *internal contention* at A is higher.

Through these examples, we see that the load estimated by our protocol is proportional to the number of stations which are actively receiving frames. The load is also proportional to the bandwidth requirements of each of these stations. That is why our protocol generates more meaningful and accurate downlink load estimates in the context of applications, such as load balancing.

Parameter	**Value**	**Parameter**	**Value**
UDP Header	8 bytes	MAC Header	36 bytes
IP Header	20 bytes	Basic Rate	1 Mbps
PHY Header	24 bytes	Data Rate	11 Mbps
CBR Payload	550 bytes	Slot Time	20 μs
SIFS	10 μs	ACK size	38 bytes
DIFS	50 μs	EIFS	364 μs

Table 4.1 Simulation Parameters

4.3 Simulation Techniques and Results

As we did in chapter 3, we simulate an IEEE 802.11b network (basic rate 1Mbps, data rate 11Mbps), operating in the infrastructure mode, using NS-2. However in this case, the MAC layer is configured to operate with the RTS-CTS exchange turned off for every frame transmission. This is because recent studies have shown that for high-rate channels, the RTS-CTS exchange actually hurts network performance [49]. The simulation parameters are shown in Table 4.1. As shown in Fig. 4.1, our simulation topology consists of one infrastructure mode BSS (one access point A, along with fifty wireless stations v_1 to v_{50}). All the stations are associated with A and have a transmit queue size of 50 frames. The access point A is connected to a fixed station B via a high speed backbone ethernet. We set the bandwidth of the ethernet to be very high and the delay very low, because we do not want the ethernet to be the performance bottleneck. We are interested in studying only the MAC layer behavior of the wireless network. Our primary goal is to observe the reported load of our protocol in response to changing network usage. The traffic pattern, simulation parameters and load estimation results for uplink and downlink traffic are discussed in section 4.3.1 and 4.3.2 respectively.

4.3.1 Uplink Traffic

4.3.1.1 Scenario 1

This is the scenario when the traffic in the network is constantly increasing. Each station v_i ($1 \leq i \leq 49$) has set up a UDP connection with the fixed station B, and transmits a one

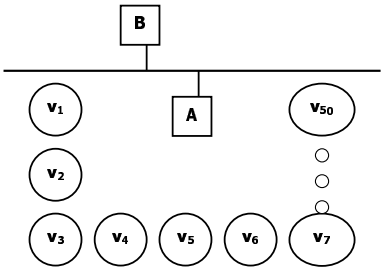

Figure 4.1 Network Topology for All Simulations

way stream of Constant Bit Rate (CBR) frames to B. The payload in each frame is 550 bytes. Including all headers (PHY, PLCP and MAC), the time taken to transmit one frame along with the time taken to receive the corresponding acknowledgment frame is 926 μs. All stations do not start their CBR flows at the same time; instead they start 5s apart. Station v_1 starts its flow at 10s, v_2 starts at 15s, and so on until v_{49} starts at 250s. At 250s, the simulation terminates.

As the number of active stations increases, the contention in the network increases, leading to more collisions. Since we have defined *load* as the fraction of time wasted in collisions, this implies that the load estimate also increases. Due to the BEB algorithm of the 802.11 DCF, we see that as the collisions increase, the CW sizes used also increases. The increased CW sizes leads to lesser collisions taking place, which in turn leads to reduced collisions. In this way, an equilibrium is reached, where the number of collisions taking place is stabilized. This simulation is run for different values of frames per second (Fps), which is nothing but the number of CBR frames that each station generates every second. The epoch size β is set to 3s, and the graph obtained is shown in Fig. 4.2. From this graph, we observe that when saturation occurs, the percentage of time wasted in collisions is around 25%. Therefore, we can say that any load estimate of approximately 0.25 implies that the channel is saturated because of excessively high contention. We also notice that when the stations transmit less than 11FPS, saturation does not take place in our simulations, because the bandwidth demand is less than what is available.

From Fig. 4.3, we examine the same scenario for the EIED variant. The behavior is

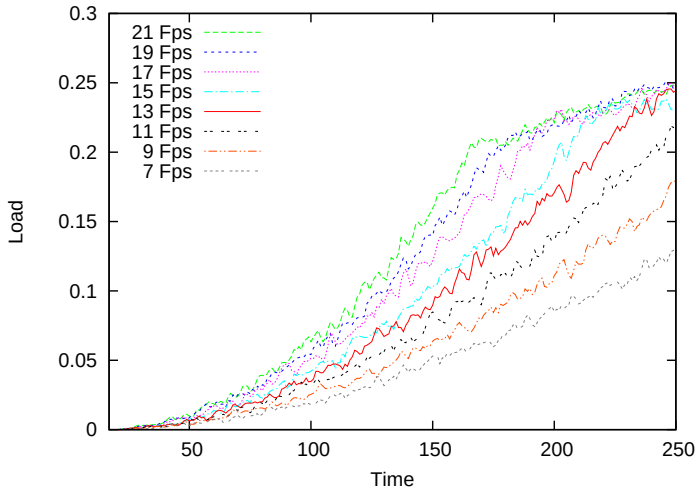

Figure 4.2 Uplink Traffic - Scenario 1 - DCF. β=3s

very similar, with one important difference. As explained in section 1.2.1.1, the conservative reduction of the CW size (as opposed to the one-shot reduction to CW_{min} upon a successful transmission) leads to lesser collisions taking place. The decreased probability of collisions leads to decreased load as shown in the graph, where the load saturates below 0.2, as opposed to below 0.25 in the case of DCF.

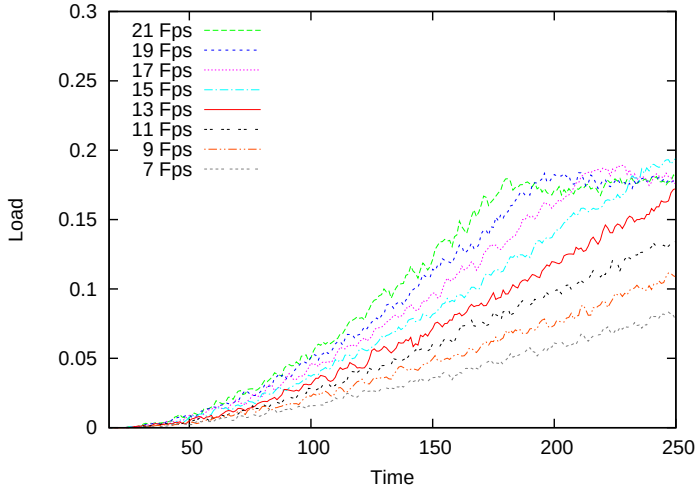

Figure 4.3 Uplink Traffic - Scenario 1 - EIED. β=3s

Next, we examine the effect of the epoch size β on the load estimate when the 802.11 DCF is used. From Fig. 4.4, we observe that a small β value means that the load estimate responds quicker to changes in the generated traffic. With larger β values, the load estimate is averaged out over a longer period of time, and hence tends to respond slower. Consequently, small β values lead to more jagged load graphs, while large β values yield smooth curves. From Fig. 4.5, we observe similar behavior with the EIED MAC variation, but the difference is that the load estimates vary much less, with varying β. This is because the variations in collisions (and hence load) in EIED are more gradual when compared to the DCF.

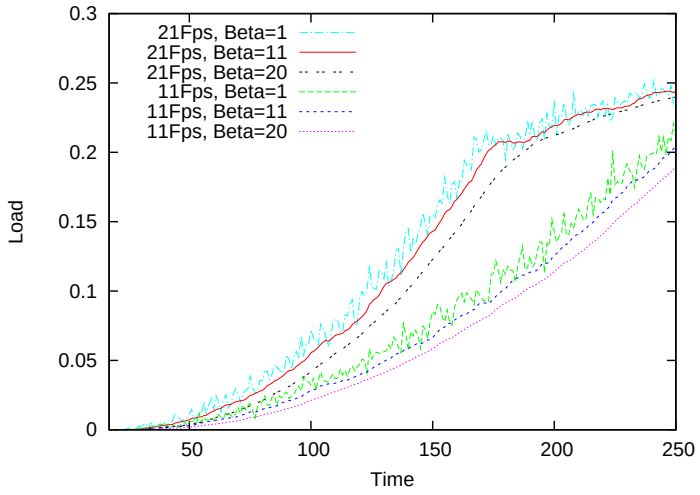

Figure 4.4 Uplink Traffic - Scenario 1 - DCF. Effect of Varying β

4.3.1.2 Scenario 2

This is the scenario when the traffic in the network is highly fluctuating. Each station v_i $(1 \leq i \leq 50)$ has set up a UDP connection with B, and transmits a one way stream of CBR frames to B. As before, the payload in each frame is 550 bytes. Including all headers (PHY, PLCP and MAC), the time taken to transmit one frame along with the time taken to receive the corresponding acknowledgment frame is 926 μs. At time 5s, stations v_1 to v_5 start transmitting, and they stop at time 10s. At 15s, stations v_1 to v_{10} start transmitting, and they stop at time 20s. At 25s, stations v_1 to v_{15} start transmitting, and they stop at time 30s. This

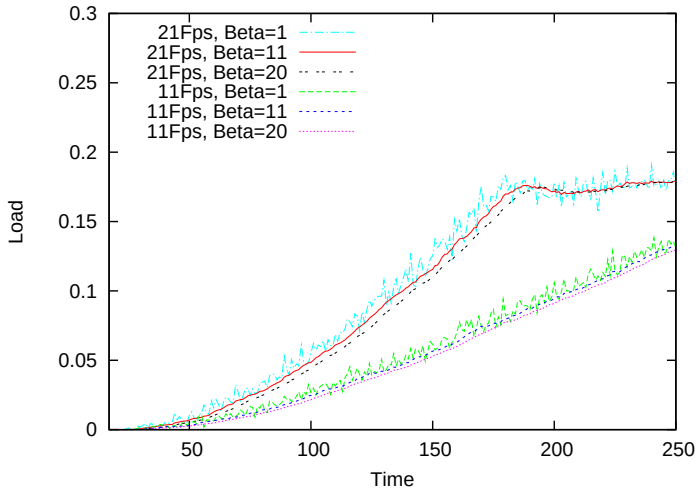

Figure 4.5 Uplink Traffic - Scenario 1 - EIED. Effect of Varying β

pattern goes on, until at 95s, stations v_1 to v_{50} start transmitting, and they stop at time 100s. The simulation terminates at 100s.

From Fig. 4.6, we observe the load for the 802.11 DCF MAC, and epoch size β equal to 3s. When a group of stations start transmitting, the collisions (and hence, load) increases. When the burst ends (stations stop transmitting), the collisions immediately end, but the load does not immediately drop down to 0, because the load values are averaged out over the epoch. As observed earlier, the load increases when the number of contending stations increases and is proportional to the bandwidth requirements of the stations. From Fig. 4.7, we observe the load for the EIED variant, for the same traffic pattern and epoch size of 3s. The behavior is similar to that of the 802.11 DCF, except that the reported load is lower. As explained earlier, this is due to the property of the EIED variant that reduces the probability of collisions taking place, when compared to the 802.11 DCF.

From Fig. 4.8 and 4.9, we observe the effect of the epoch size β on the load estimate for the 802.11 DCF and the EIED variant respectively. As seen earlier, a small β value leads to the load estimate responding to changes in the generated traffic quicker, while a large β value leads to slower load response and a smoother load estimate. Therefore, depending on the semantic of the application using the load estimate, an appropriate β value can be chosen.

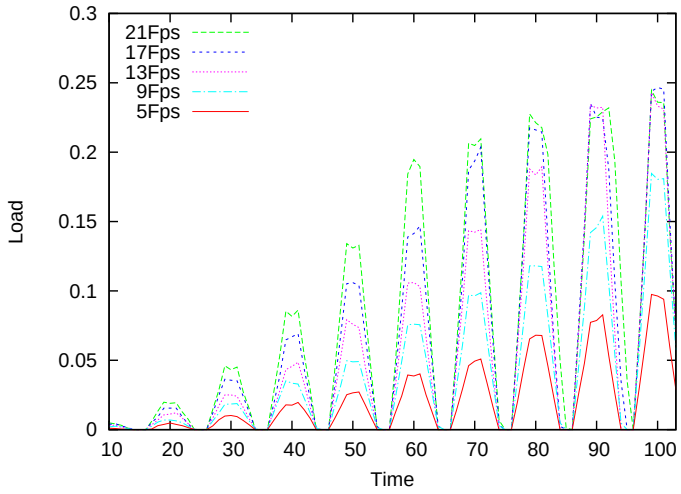

Figure 4.6 Uplink Traffic - Scenario 2 - DCF. β=3s

4.3.2 Downlink Traffic

This is the scenario when the downlink traffic in the network is constantly increasing. Each station v_i $(1 \leq i \leq 50)$ is the sink of an individual UDP connection, with B as the source. B transmits a one way stream of Constant Bit Rate (CBR) frames to each station v_i over the individual connections. The CBR flows are are started 5s apart. The flow to station v_1 starts at 10s, the flow to v_2 starts at 15s, and so on until the flow to v_{50} starts at 255s. At 255s, the simulation terminates.

Using the approximation mentioned in section 4.2.2.3, from Fig. 4.10, we observe how the downlink load varies with time, for an epoch size β of 3s. This load value intuitively captures the notion of the *amount of internal contention* taking place at the access point. As the number of downlink flows increases, the *internal contention* increases, leading to an increase in load. The load increases while the network is not saturated. Upon saturation, the load value stabilizes, and does not significantly increase when the number of flows is further increased. We also observe that the reported load is larger when the downlink bandwidth demand of the individual stations is higher.

Next, we examine the effect of the epoch size β on the load estimate in Fig. 4.11. As expected, small β values respond quicker to traffic variations, while large β values average out

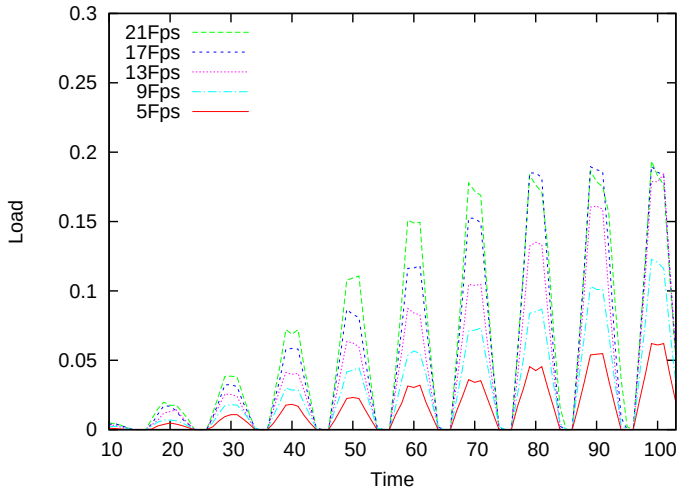

Figure 4.7 Uplink Traffic - Scenario 2 - EIED. β=3s

the load over the longer epoch; consequently, smaller β values detect saturation quicker.

Next, we consider the case when all stations in the network have an infinite number of frames to receive from the access point. In this case, the network is always saturated, and the load value depends on the number of stations that cause this saturation. If there are m stations that are equally involved, then the saturated load is: $(1 + \frac{1}{m})^m$. Using this formula to obtain Fig. 4.12, we plot the saturated load as a function of the number of stations. It is obvious that any load value of greater than 2.0 indicates saturation. As explained in section 4.2.2.2, the upper bound on load is equal to e (2.718), and is reached when the number of stations tends to infinity. At first glance, it might seem strange that the reported load is almost equal, whether the number of stations causing the saturation is 30 (load is 2.674) or 100 (load is 2.704). However, in the context of applications such as load balancing, this is inconsequential. All we need to know is that a large number of stations are causing network saturation, and load balancing should be done to relieve the overloaded access point.

4.4 Conclusion

In this chapter, we have addressed the problem of estimating the load on an access point, for applications such as load balancing and association control. We split up *load* into uplink and

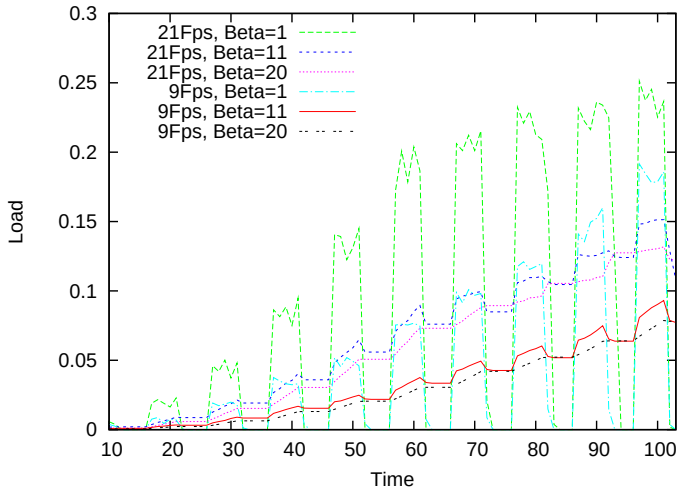

Figure 4.8 Uplink Traffic - Scenario 2 - DCF. Effect of Varying β

downlink components, and measure them independently using different metrics. Our definition of uplink load measures the *amount of contention in the BSS*, while our definition of downlink load measures the *amount of internal contention at the access point.*

We have also argued why our load estimates are more meaningful than traditional load estimates. Through simple numerical analysis and extensive simulations, we have shown how our load estimates should be interpreted and what they tell us about the actual channel conditions. We plan to extend our work by making it QoS aware - being able to estimate the uplink and downlink load independently for each of the 4 priorities of data, as mentioned in the IEEE 802.11e standard [47].

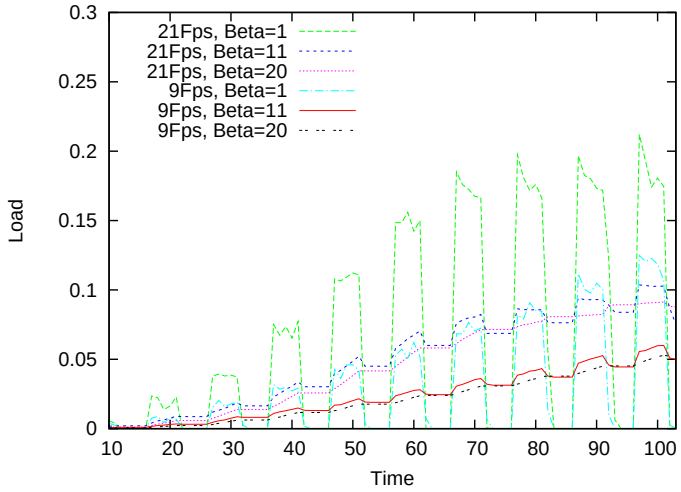

Figure 4.9 Uplink Traffic - Scenario 2 - EIED. Effect of Varying β

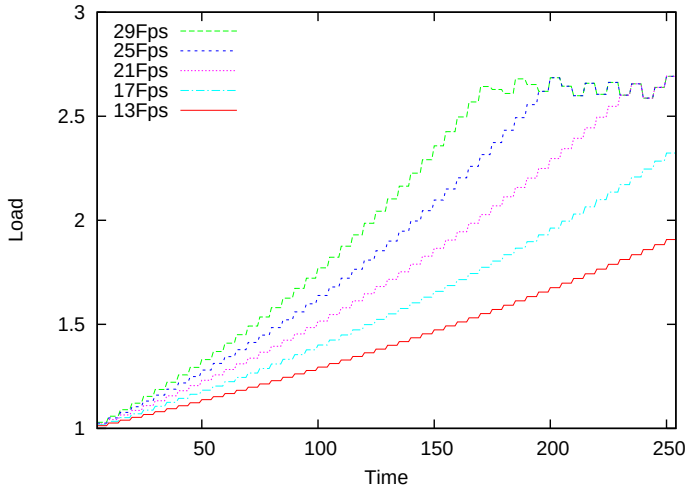

Figure 4.10 Downlink Traffic. $\beta = 3$s

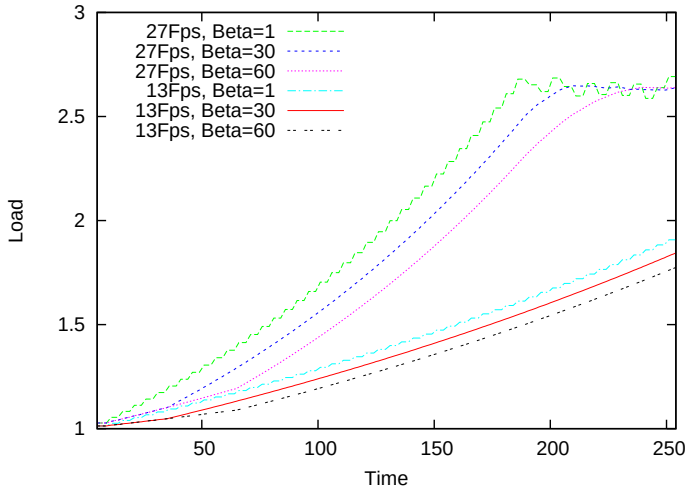

Figure 4.11 Downlink Traffic. Effect of Varying β

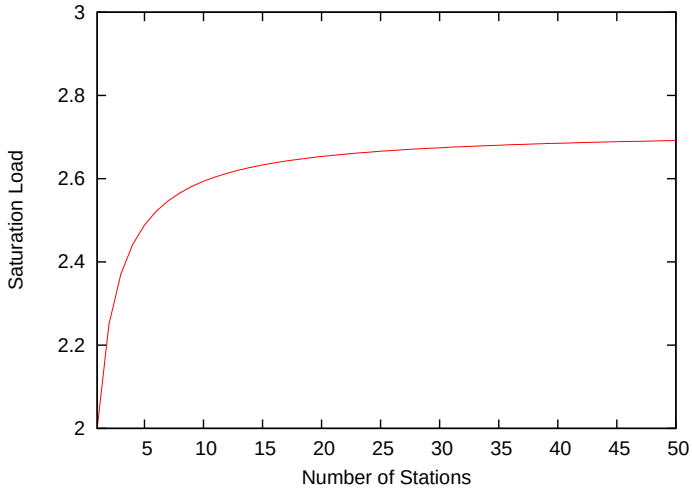

Figure 4.12 Downlink Saturation Load

CHAPTER 5. TESTBED IMPLEMENTATION

5.1 Introduction

The work presented in this chapter is outlined below: In section 5.2, we describe in detail how we built our testbed. We describe its hardware, topology, software architecture, along with explanations of every module that we implemented. In section 5.3, we demonstrate the efficacy of our load estimation technique, using an association control algorithm that we implemented. We also emphasize on a few interesting observations and insights gained, through the testbed.

5.2 System Design

In this section, we describe our testbed topology, and give a systems description of each type of machine used. We also outline the different modules present in each of these types of machines.

5.2.1 Topology and Configuration

There are three types of machines that we use for our experiments:

- Station (client) machines, that represent the end users of the wireless network
- Access Points
- Controller

As shown in figure 5.1, our testbed consists of 11 machines: 1 controller, 3 access points and 7 clients. All machines are actually ordinary computers, with at least 1GHz of processing power, 512MB of RAM and 20GB of hard drive space. All systems run the Fedora 7 distribution

of Linux, with kernel 2.6.21 or higher. The entire testbed is located in 115/116 Atanasoff Hall, Iowa State University.

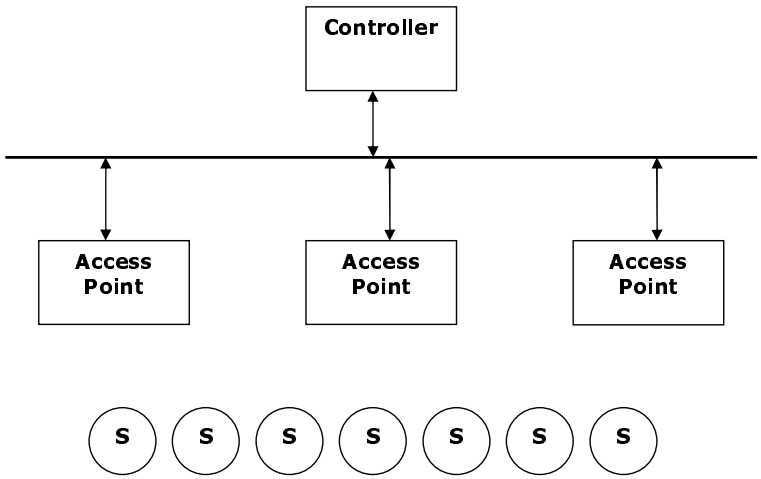

Figure 5.1 Topology of the Testbed

The access points and the controller are all connected to a high speed ethernet, that operates at 100Mbps. The backbone ethernet uses a D-Link ethernet switch *DES 1024D*. The access points and the controller are connected to the switch using *Cat-5* straight cables. The controller has only one network interface, which is the ethernet interface. It has a static Class-C IP address, which is 192.168.1.12.

Each access point has two network interfaces: a wired 100Mbps ethernet interface, and a wireless interface. The wireless interfaces we used in our testbed are Linksys *WMP55AG* PCI wireless cards, which are capable of operating 802.11a/b/g. These cards use Atheros based chipsets and are manufactured by Cisco. The maximum transmit power of these cards is -19dBm. We used the *Multi-mode Atheros Drivers for Wi-Fi* (MadWifi) open source device drivers to run the wireless interface.

We configured 3 machines to behave as access points as follows:

- For simplicity, assume that the wired interface is called `eth0` and the wireless interface is called `ath0`.

- The DCHP client `dhclient` on the machine is killed, because we want static IP addresses assigned.
- The interfaces `eth0` and `ath0` are brought down using `ifconfig`
- Using `brctl` (from the `bridge-utils` package of linux), create a bridge called `br0`.
- Using `wlanconfig` (present in MadWiFi), destroy the `ath0` interface.
- Using `wlanconfig`, create the `ath0` interface, in access point mode.
- Set the ESSID of `ath0` to *Testbed*.
- Using `brctl`, add the interface `eth0` to the bridge `br0`.
- Using `brctl`, add the interface `ath0` to the bridge `br0`.
- Set the channel on `ath0` using `iwconfig`.
- Bring the ethernet interface `eth0` up using the command `ifconfig eth0 0.0.0.0 up`
- Bring the wireless interface `ath0` up using the command `ifconfig ath0 0.0.0.0 up`
- Bring the bridge up, using `ifconfig br0 192.168.1.x up`. This is the IP address of the access point. Note that the IP address isn't assigned to any of its interfaces; instead, the bridge assumes the IP address of the access point.
- Set the base data rate used by the access point using `iwconfig`, if needed.
- The access point 192.168.1.x is up and ready to use.

The seven clients have their wired interfaces turned off, and are not connected to the backbone directly. Configuring a machine to behave as a client is considerably simpler:

- The DCHP client `dhclient` on the machine is killed, because we want static IP addresses assigned.

- The interfaces `eth0` and `ath0` are brought down using `ifconfig`.
- Using `wlanconfig` (present in MadWiFi), destroy the `ath0` interface.
- Using `wlanconfig`, create the `ath0` interface, in station mode.
- Set the ESSID of `ath0` to *Testbed*.
- Bring the wireless interface up, using `ifconfig ath0 192.168.1.y up`. This is the IP address of the station.
- The station is up and ready to use.

5.2.2 Software Modules

The block diagram of our system is shown in figure 5.2. We have built a module known as `Access Point Client` that resides on each access point. Similarly, a module known as `Station Client` resides on each station. The controller machine has many modules, each with a well defined task.

The overview of our modules is as follows: The `access point client` periodically informs the controller about which stations are associated with it, along with the bandwidth used by each of these stations. The `access point server` module in the controller machine accepts this information, and writes it into a SQL database. The `station client` periodically scans all available channels, and makes a list of all access points that are currently in its radio range, along with the respective signal qualities. This list is sent to the `station server` module in the controller machine, which writes it into a database.

The `association engine` module in the controller machine looks at this information present in the database, and makes a handoff decision if necessary. This information is communicated to the station, where it is received by the `station client`. The station client then proceeds to handoff to a new access point, based on the instructions given by the `association engine`.

Now, let us take a more detailed look at how each of these modules work:

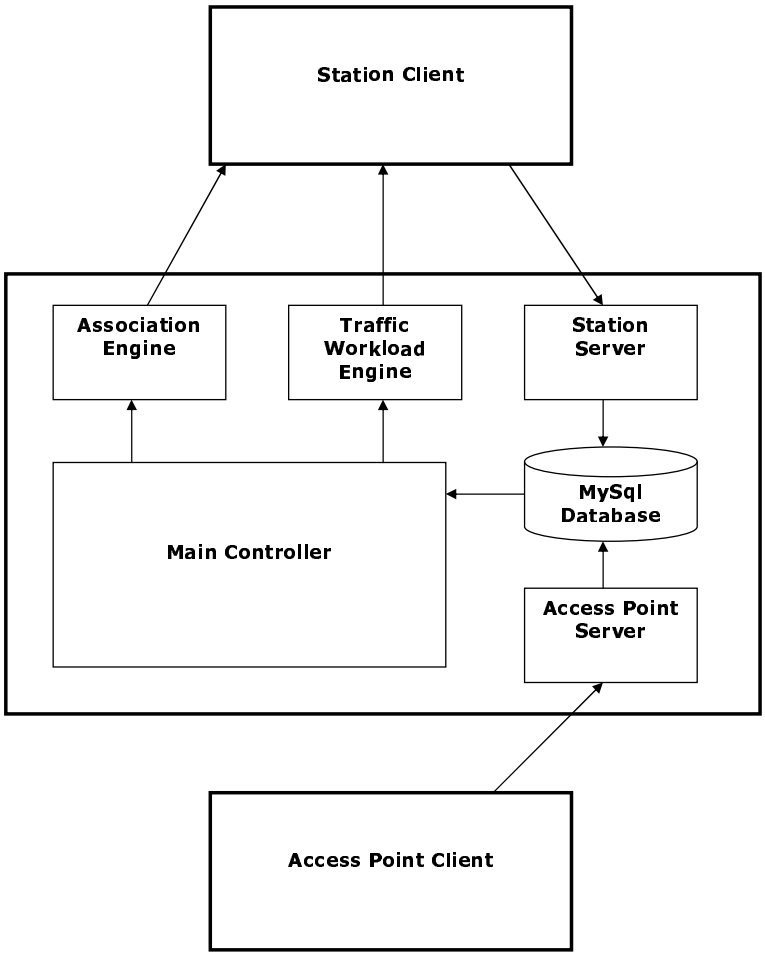

Figure 5.2 Modules of the Testbed System

5.2.2.1 Access Point Client

This module first makes list of stations that are associated with it. The `wlanconfig ath0 list` shell command produces a raw output dump, which needs to be parsed. This module parses this output, and gets the list of associated stations *stationMacList*. We also need to extract the utilized bandwidth by each of these associated stations.

The MadWifi driver has a tool known as `athdebug`, which can be used to get a dump of all frames that are handled by the wireless interface. We have modified the MadWiFi source code, to dump only the *frame headers*, as opposed to the entire frame. Among other fields, the dump contains the source and destination address of every frame. This is done so as to

reduce the size of the dump, and hence, the processing and parsing efficiency. The bandwidth utilized by each associated station is extracted from this dump as follows:

- Clear all `athdebug` dumps.
- Start the `athdebug` dump, sleep for 3 seconds, and then stop the `athdebug` dump.
- For each associated station, parse the dump, to see how much bandwidth has been utilized by that station during the sleep period of 3 seconds.

Additionally, the access point calculates its *unified load*, which is based on a modification of the technique presented in section 4.2.2.2:

$$Load_{Unified} = 100 \times \prod_{i \in Stations} (1 + \frac{n_i}{n_{max}})^{\alpha} \tag{5.1}$$

Experimentally, we found that in our testbed settings, n_{max} is 750 frames. In our testbed, all stations are capable of communicating at 54Mbps with all access points in their radio range. However, if this condition were not to hold good in real-life, then n_{max} would be a function of the data rates being used by the stations. We intend to conduct a deeper study of the effects of rate adaptation in the future.

The equation found in section 4.2.2.2 is used only to calculate the downlink load. However, we have experimentally found that there is a close correlation between the uplink and downlink loads. This is because most higher layer protocols are two-way (like for example, TCP). Therefore, an increase in the load in one direction automatically implies that the load in the reverse direction is also increased. However, there is a subtle difference between the two: an increase of downlink frames does not degrade throughput, whereas an increase in uplink frames degrades throughput due to increased contention and collisions. As a result of this, in the equation, we raise the product term by a factor of α, and we get a unified load estimate, which approximates together the downlink and uplink loads. Experimentally (through association control), we found that if we set α to 2, the unified load estimate efficiently captures both the uplink and downlink loads together.

The access point reports this unified load to the controller. As mentioned earlier, it also reports to the controller, the list of associated stations, along with the bandwidth utilized by each of them.

5.2.2.2 Access Point Server

This module resides in the controller machine. Previosuly, we have taken a look at the duties of the `access point client`. Now, it is the duty of the `access point server` to receive the information sent by the `access point client`, and to write it into the database. The communication model between the `access point client` and `access point server` is based on simple TCP sockets.

5.2.2.3 Station Client

This module resides on the client machines. Its duty is to report to the controller, the list of all access points that are in its range, along with the respective signal qualities. It performs its tasks as follows:

- The `iwlist` shell command is invoked, to scan all channels. This scanning process takes about 1 second, and produces a dump of all access points that are in its radio range.
- This module parses this dump output, and removes information pertaining to all access points that do not have their ESSID set to *Testbed*.
- This output is further parsed, to extract the access point MAC address and the received signal quality from the access point.
- This information is passed on to the `station server`, which resides in the controller machine.

5.2.2.4 Station Server

Much like the `access point server` that receives data from the access point client, this module waits for the station clients to report their scanning data to it. Once this data is

received, it is inserted into the database. The communication between the `station client` and `station server` is also based on simple TCP sockets.

5.2.2.5 Database

The database used is `MySQL 5.0.45`, that is available for free on Linux. We use `libmysql` to interface our C programs with the MySQL database. In our implementation, we use just two tables:

- AP-LOAD: This table is populated by the `access point server` module. Its schema and sample contents are shown in figure 5.3.
  - `AP_MAC:` The MAC address of the access point.
  - `STA_MAC:` The MAC address of a station that is associated with the access point `AP_MAC`. Note that `AP_MAC` together with `STA_MAC` form the primary key of this table.
  - `STA_LOAD:` The number of frames that the access point `AP_MAC` handles for station `STA_MAC` in this aggregation epoch.
  - `NORM_LOAD:` This value is the normalized (unified) load for access point `AP_MAC`, calculated by equation 5.1.
- STATION: This table is populated by the `station server` module. Its schema and sample contents are shown in figure 5.4.
  - `STA_MAC:` The MAC address of a station in the network.
  - `IP_ADDRESS:` It is the IP address of the station, whose MAC address is `STA_MAC`.
  - `AP_MAC:` The MAC address of an access point that is in the radio range of the station, whose MAC address is `STA_MAC`. Note that `STA_MAC` together with `AP_MAC` form the primary key of this table.
  - `QUALITY:` It is the signal quality with which the station (with MAC address `STA_MAC`) receives beacon frames from the access point, whose MAC address is `AP_MAC`.

Field	Type	Null	Key	Default	Extra
AP_MAC	char(18)	YES		NULL	
STA_MAC	char(18)	YES		NULL	
STA_LOAD	int(11)	YES		NULL	
NORM_LOAD	int(11)	YES		NULL	

AP_MAC	STA_MAC	STA_LOAD	NORM_LOAD
00:11:50:55:54:C0	00:11:50:55:4C:88	0	100
00:11:50:55:54:C0	00:11:50:55:4B:4E	0	100
00:11:50:55:54:C0	00:11:50:55:4F:6C	0	100
00:11:50:55:54:C0	00:11:50:55:54:7F	0	100
00:11:50:55:54:C0	00:11:50:55:51:29	0	100
00:11:50:55:4F:7C	00:11:50:55:5D:49	0	100
00:11:50:55:23:E5	00:11:50:55:51:B0	0	100

Figure 5.3 Table `AP-LOAD`

5.2.2.6 Association Engine

The association engine is at the heart of our testbed. Its duty is to manage the station-AP associations, with the goal of maximizing network throughput. Note that this module takes its inputs from the database. The steps taken by (algorithm used by) this engine are enumerated below:

1. Find an overloaded access point A, using table `AP-LOAD`. If none exists, terminate.

2. For this access point A, use table `AP-LOAD` to find a station S that is utilizing between 5% and 50% of its total throughput. If none exists, terminate.

3. Using tables `STATION` and `AP-LOAD`, find a non-overloaded access point B, such that S is in the range of B, and the signal quality between them is good. If such an access point B cannot be found, terminate.

4. Send a command to the station S, telling it to reassociate from access point A to access point B.

Field	Type	Null	Key	Default	Extra
STA_MAC	char(18)	YES		NULL	
IP_ADDRESS	varchar(16)	YES		NULL	
AP_MAC	char(18)	YES		NULL	
QUALITY	int(11)	YES		NULL	

STA_MAC	IP_ADDRESS	AP_MAC	QUALITY
00:11:50:55:4F:6C	192.168.1.2	00:11:50:55:23:E5	38
00:11:50:55:4F:6C	192.168.1.2	00:11:50:55:54:C0	38
00:11:50:55:4F:6C	192.168.1.2	00:11:50:55:4F:7C	68
00:11:50:55:51:29	192.168.1.3	00:11:50:55:4F:7C	69
00:11:50:55:51:29	192.168.1.3	00:11:50:55:23:E5	44
00:11:50:55:51:29	192.168.1.3	00:11:50:55:54:C0	43
00:11:50:55:5D:49	192.168.1.4	00:11:50:55:23:E5	44
00:11:50:55:5D:49	192.168.1.4	00:11:50:55:4F:7C	56
00:11:50:55:5D:49	192.168.1.4	00:11:50:55:54:C0	45
00:11:50:55:4B:4E	192.168.1.5	00:11:50:55:23:E5	43
00:11:50:55:4B:4E	192.168.1.5	00:11:50:55:54:C0	42
00:11:50:55:4B:4E	192.168.1.5	00:11:50:55:4F:7C	47
00:11:50:55:51:B0	192.168.1.6	00:11:50:55:4F:7C	37
00:11:50:55:51:B0	192.168.1.6	00:11:50:55:23:E5	42
00:11:50:55:51:B0	192.168.1.6	00:11:50:55:54:C0	36
00:11:50:55:54:7F	192.168.1.7	00:11:50:55:54:C0	50
00:11:50:55:54:7F	192.168.1.7	00:11:50:55:4F:7C	51
00:11:50:55:54:7F	192.168.1.7	00:11:50:55:23:E5	67
00:11:50:55:4C:88	192.168.1.8	00:11:50:55:54:C0	45
00:11:50:55:4C:88	192.168.1.8	00:11:50:55:4F:7C	44
00:11:50:55:4C:88	192.168.1.8	00:11:50:55:23:E5	62

Figure 5.4 Table `STATION`

5. Terminate. Note that in every run of this algorithm, *at most* one reassociation is forced. This algorithm is invoked repeatedly, typically with a 15 second wait period, in order for the network to re-attain steady state.

5.2.2.7 Traffic Workload Engine

This module is used to control the traffic in the testbed. The user (person who is running experiments on the testbed) can determine before hand, the traffic patterns for each client machine in the testbed. This pattern is entered by him/her into a traffic descriptor file. An example of a traffic pattern is given below:

- At time 0s, let stations 192.168.1.2 and 192.168.1.3 start FTP downloads from the file

server. Note that in our implementation, the file server is located on the same machine as the controller.

- At time 30s, let stations 192.168.1.4 and 192.168.1.5 start FTP downloads from the file server.
- At time 300s, kill all the FTP flows present in the testbed.

The `traffic workload engine` parses this descriptor file, and starts / kills flows accordingly. This way, we can run experiments in a controlled manner, such that we can study the performance of the testbed and association control algorithm.

5.2.2.8 Main Controller

This module is used to synchronize all the other modules in the controller machine. For example, it invokes the `association engine` once every x seconds, where x is an adjustable parameter. It also invokes the traffic start / stop engine during the start / stop of an experimental run.

5.3 Results

In this section, we make three main observations, which are explained in the next few subsections.

5.3.1 Association Engine Performance

In order to observe the performance of our association control algorithm, we start traffic workloads, and observe how the total throughput varies as the reassociations are forced off. Initially, all 7 stations are associated with the same access point. At time 0s, all 7 stations start downloading a very large file from the file server. In our implementation, the file server is located on the controller machine itself. The stations are configured not to handoff to other access points, unless explicitly instructed to do so by the controller.

The `main controller` module of the controller periodically invokes the association engine. We have configured this action to be triggered once every 30s. Recall that in every invocation of the association engine, at most one reassociation can be forced. This behavior is not required; However in order to study the performance implication of a single handoff, we have configured the system to behave in such a manner.

The throughput results are shown in figure 5.5. In this figure, the X axis is time, and the Y axis is the total number of frames transmitted to the stations by all access points in the previous 30 seconds. We notice that when a reassociation is triggered, there is usually a change in the total throughput. Clearly, we see that a single reassociation can significantly change the total throughput in the network. We ran this experiment 5 times, and the results are plotted.

In order to demonstrate the performance of our association engine, we ran the experiment once, with the engine turned off. The throughput obtained is also shown in the graph. Clearly, we see that the association engine helps in increasing the overall throughput and network utilization. We did observe however, that sometimes the association engine can make suboptimal decisions. We believe this is due to the fact that we have not chosen α carefully enough (see section 5.2.2.1). In the future, we intend running experiments to find out what value of α is *optimal*. However, for a large majority of cases, we found that setting α to 2 works well.

Our association control algorithm does not explicitly control fairness. The algorithm we use is rather simplistic: find an active station that is associated with an overloaded access point, and bump it off to a non-overloaded access point in its range. Although we do not currently know whether this algorithm is fair, we intuitively say that by *balancing the contention* across access points, we provide better access opportunities. We do not know if our forced reassociations lead to balanced or skewed per-station throughput, although this is one of the several aspects we intend to study further in the future. If we find that stations get unfair fractions of the available bandwidth, then we will need to fundamentally redesign our association control algorithm, to make smarter decisions. At the same time we note however, that *MaxMin* fairness isn't probably the ideal metric to judge fairness, as it potentially can conflict with the goal of maximizing the network utilization and throughput [50].

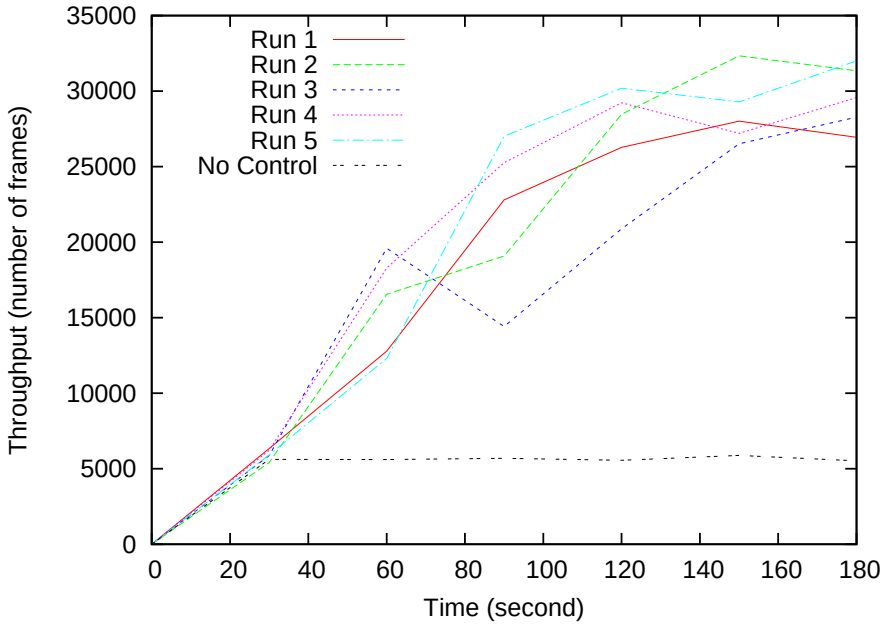

Figure 5.5 Performance of the Association Engine

5.3.2 Observing the Effect of Collisions

In order to understand the performance implications of high uplink contention, we ran the following experiment. We associated all 7 stations with the same access point. Next, we began uploading a very large file from each of the 7 stations, to the file server located on the controller machine. As we have done before, the stations are configured not to handoff, unless explicitly instructed to do so by the controller. In this experiment, we turned off the association engine. We observed that the total throughput saturates very quickly, and then remains more or less constant. These results are shown in figure 5.6.

We repeated the experiment three times, with 5, 3 and 1 station respectively. We found that as the number of stations contending in the uplink channel is lowered, then the overall throughput achievable is significantly higher. With only one station, the throughput reaches close to the theoretically observed maximum throughput.

We repeated this experiment, but with downlink traffic instead of uplink traffic. We ran the experiment 4 times, with 7, 5, 3 and 1 station respectively. As we mentioned before, since

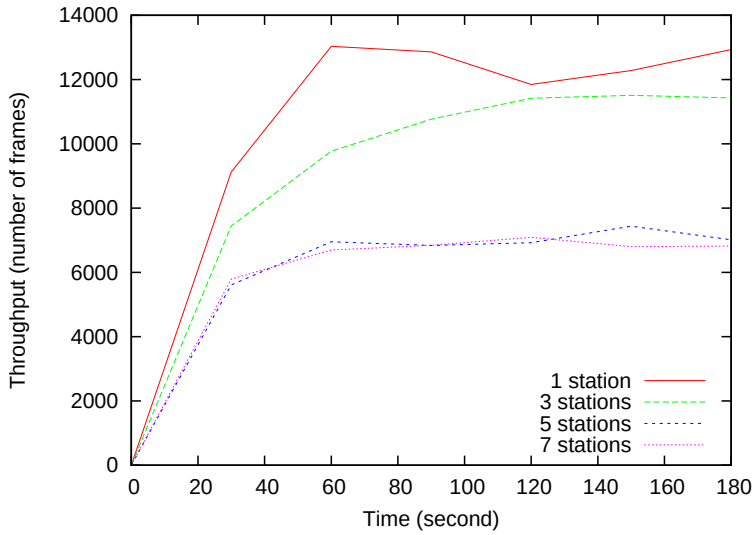

Figure 5.6 Uplink Saturation Throughput

protocols such as TCP and FTP involve end-to-end handshaking, an increase in downlink load is typically followed by an increase in uplink load as well. However, the differences in throughput was not as significant as observed in the case of uplink traffic. This behavior is expected, because in this case, the amount of uplink traffic is not as significant when compared to the case where the stations upload files to the file server.

From figure 5.6, it becomes very obvious that defining *load* as the number of frames handled per unit time is not a good metric. By extension of a similar argument, even busy-time based estimates of load are not accurate. In fact, they are counter intuitive, and may degrade network performance, when applied to association control.

Based on our definition of load, we observed that when there were multiple actively contending stations, the load was significantly higher than the case where just one station was blasting away without any contention. This validates our claim that our definition of load is meaningful, in such contexts.

5.3.3 Unfairness in Channel Access Opportunities

It is interesting to note that although the 802.11 MAC is theoritically fair, in practice some stations seem to get preferential treatment over other stations. To illustrate this, we ran the following experiment: we associated all seven stations to the same access point; as before, stations do not initiate any handoffs on their own accord. Each station begins to download a large file from the file server using FTP over TCP.

At this point, we note that the only bottleneck in the network is the access point to client channel. The wired back bone operates at nearly twice the data rate, when compared to the wireless side. Moreover, the protocol overheads of 802.3 ethernet are smaller, when compared to the 802.11 MAC.

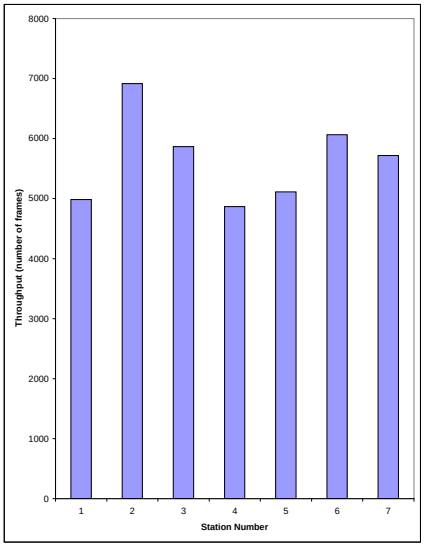

Figure 5.7 Throughput Distribution among the Stations

We run this experiment, while keeping a tab on the number of frames each of the seven stations has received. At the end of 150 seconds, we terminate the traffic flows, and observe the results. As shown in figure 5.7, some stations receive a significantly higher share of the throughput, when compared to other stations. The possible causes for this behavior are:

- Since these radios are inexpensive, their timers and clocks may have a certain skew. This leads to some stations perpetually having an advantage over others.
- Although these 7 stations were placed in different physical locations in the laboratory, they all were able to communicate with the access point at the full data rate. This means that the throughput difference cannot be attributed to near-far effects. One explanation is that due to multipath propagation, different stations perceive the channel quality differenly.
- Although the interaction of TCP with the MAC layer could potentially cause this behavior, our experiments are inconclusive in this regard. We intend to study such cross layer interactions in the future.

We have observed that different stations get different throughputs from the access point over a large time scale. This effect is more pronounced over shorter time scales. We repeated the same experiment; however, we observed the bandwidth distribution among the nodes over a 3 second aggregation interval, as opposed to a 150 second aggregation interval. A few of these distributions observed over different 3-second aggregation periods in a 150 second run are shown in fig 5.8 and 5.9.

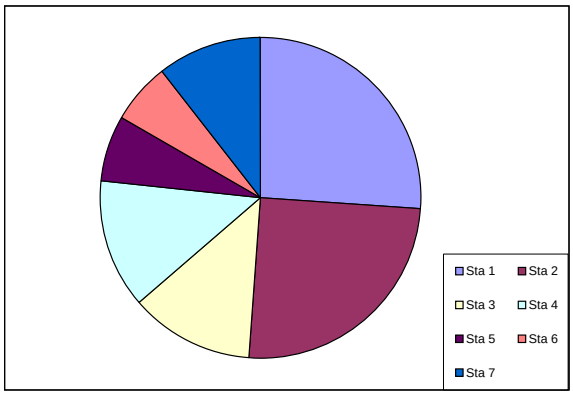

Figure 5.8 Throughput Distribution between 87s and 90s

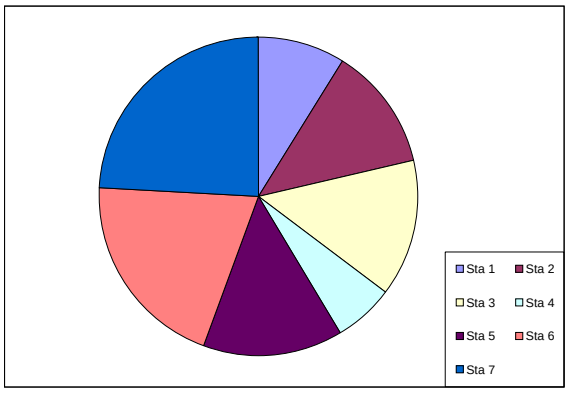

Figure 5.9 Throughput Distribution between 114s and 117s

5.4 Conclusion

Through this testbed, we have validated our load estimation schemes, and have demonstrated its performance using association control as a target application. In the future, we plan to design load estimation with adaptive α (explained in section 5.2.2.1). We also plan to extend our network management plane to include channel selection and power control as well.

Based on our experience, we realised that building a real system that works is indeed very challenging. We faced many issues during the process, primarily due to buggy software on the MadWiFi device drivers. The challenges and subtleties involved in building this system, trouble-shooting and designing practical solutions to problems faced cannot be emphasized enough.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

We know that careful management of resources can significantly improve the overall performance of the network. Many of these management techniques (such as association control, channel selection and power control) need to be carefully used in order to be effective.

We have also seen how and why the efficacy of these functions relies on a meaningful load estimate. This is the motivation behind our work: to design metrics and techniques for meaningful load estimation, in the context of network management functions.

Our initial work was on load estimation that required changes on the client code. We extended this work, and designed a set of techniques to non-intrusively estimate the uplink and downlink load on each access point.

In the future, we will see proliferation of many real-time applications being used over wireless channels. An obvious extension to our work on load estimation is to estimate the load for each of the traffic priorities independently, such that even smarter network management decisions can be made.

Through testbed experiments, we have validated our load estimation techniques, by using association control as the target application. We have also shown how our association control algorithm performs well by significantly increasing the overall throughput. For our testbed, we have modified the client unaware downlink load estimate technique, to generate a unified load estimate that captures an intuitive notion of the amount of contention in the network.

We have realised that no amount of simulation or analysis can capture the behavior of an actual network when in deployment. Through our experiments, we have seen that many assumptions we commonly make do not hold in real life.

These advantages of a testbed have motivated us to extend our work on network manage-

ment, by implementing channel selection and power control on the management plane.

We believe that there is enormous scope for more research in this field, as there are many problems that are unsolved. This field is indeed very exciting, as the potential applications of wireless technologies are innumerable.

Bibliography

[1] IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[2] R. R. Kompella, S. Ramabhadran, I. Ramani, and A. C. Snoeren, Cooperative packet scheduling via pipelining in 802.11 wireless networks, in *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pp. 35–40, New York, NY, USA, 2005, ACM.

[3] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall, Improved access point selection, in *MobiSys 2006: Proceedings of the 4th international conference on Mobile systems, applications and services*, pp. 233–245, New York, NY, USA, 2006, ACM.

[4] V. Mhatre and K. Papagiannaki, Optimal Design of High Density 802.11 WLANs, in *CoNEXT '06: 2nd Conference on Future Networking Technologies*, 2006.

[5] Bruno Kauffmann, Francois Baccelli, Augustin Chaintreau, Vivek Mhatre, Konstantina Papagiannaki and Christophe Diot, Measurement-Based Self Organization of Interfering 802.11 Wireless Access Networks, in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1451–1459, 6-12 May 2007.

[6] Chih-Chiang Wu Shiann-Tsong Sheu, Dynamic Load Balance Algorithm (DLBA) for IEEE 802.11 Wireless LAN, in *Tamkang Journal of Science and Engineering*, pp. 45–52, 1999.

[7] I. Papanikos and M. Logothetis, A Study on Dynamic Load Balance for IEEE 802.11b Wireless LAN, in *In proceedings of COMCON*, 2001.

[8] Rohan Murty, Jitu Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill, Designing High Performance Enterprise Wi-Fi Networks, in *In Proceedings of the 5th USENIX/ACM Symposium on Networked Design and Implementation (NSDI)*, 2008.

[9] Dongmei Zhao and Jun Zou and Terence D. Todd. Admission control with load balancing in IEEE 802.11-based ESS mesh networks, Wireless Networks vol 13, num 3 , 351 (2007).

[10] I. Broustis, K. Papagiannaki, S. V. Krishnamurthy, M. Faloutsos, and V. Mhatre, Mdg: measurement-driven guidelines for 802.11 wlan design, in *MobiCom '07: Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pp. 254–265, New York, NY, USA, 2007, ACM.

[11] Nabeel Ahmed and Vivek Shrivastava and Arunesh Mishra and Suman Banerjee and Srinivasan Keshav and Konstantina Papagiannaki, Interference mitigation in enterprise WLANs through speculative scheduling, in *MobiCom '07: Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pp. 342–345, New York, NY, USA, 2007, ACM.

[12] Velayos, H. and Aleo, V. and Karlsson, G., Load balancing in overlapping wireless LAN cells, in *Communications, 2004 IEEE International Conference on*, pp. 3833–3836, 20-24 June 2004.

[13] Y. Bejerano, S.-J. Han, and L. E. Li, Fairness and load balancing in wireless lans using association control, in *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pp. 315–329, New York, NY, USA, 2004, ACM.

[14] Anand Balachandran and Paramvir Bahl and Geoffrey M. Voelker, Hot-spot congestion relief and service guarantees in public-area wireless networks, in *SIGCOMM Compututer Communication Review*, New York, NY, USA, 2002, ACM.

[15] T-C. Tsai and C-F. Lien, IEEE 802.11 Hot Spot Load Balance and QoS-maintained Seamless Roaming, in *In Proceedings of National Computer Symposium (NCS)*, 2003.

[16] N. Ahmed and S. Keshav, SMARTA: A Self-Managing Architecture for Thin Access Points, in *CoNEXT*, 2006.

[17] V. Mhatre, K. Papagiannaki, and F. Baccelli, Inteference Mitigation through Power Control in High Density 802.11 WLANs, in *Infocom*, 2007.

[18] Y. Bejerano and R. S. Bhatia, MiFi: a framework for fairness and QoS assurance in current IEEE 802.11 Networks with Multiple Access Points, in *Infocom*, 2004.

[19] N.-O. Song, B.-J. Kwak, J. Song, and L. E.Miller, Enhancement of IEEE 802.11 Distributed Coordination Function with Exponential Increase Exponential Decrease Backoff Algorithm, in *In Proceedings of the 57th IEEE Semi-annual Spring VTC, vol. 4*, pp. 2775–2778, 2003.

[20] E. G. Villegas, R. V. Ferre, and J. P. Aspas, Load Balancing in WLANs through IEEE 802.11k Mechanisms, in *ISCC*, 2006.

[21] David Kotz and Kobby Essien, Analysis of a campus-wide wireless network, in *Wireless Networks vol 11*, pp. 115–133, Hingham, MA, USA, 2005, Kluwer Academic Publishers.

[22] Magdalena Balazinska and Paul Castro, Characterizing mobility and network usage in a corporate wireless local-area network, in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 303–316, New York, NY, USA, 2003, ACM.

[23] V. Brik, A. Mishra, and S. Banerjee, Eliminating handoff latencies in 802.11 WLANs using multiple radios: Applications, experience, and evaluation, in *IMC*, 2005.

[24] I. Ramani and S. Savage, SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks, in *Infocom*, Miami, FL, 2005.

[25] Ramon Caceres and Venkata N. Padmanabhan, Fast and Scalable Wireless Handoffs in Support of Mobile Internet Audio, in *Mobile Networks and Applications vol 3, num 4*, pp. 351–363, 1998.

[26] V. Mhatre and K. Papagiannaki, Using smart triggers for improved user performance in 802.11 wireless networks, in *MobiSys*, 2006.

[27] Y. Amir, Claudiu, M. Hilsdale, R. Musaloiu-Elefteri, and N. Rivera, Fast handoff for seamless wireless mesh networks, in *MobiSys*, 2006.

[28] G. Davis. Using picocells to build high-throughput 802.11 networks, Using picocells to build high-throughput 802.11 networks, 2004.

[29] Arunchandar Vasan, Ramachandran Ramjee and Thomas Woo, INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE **3**, 1562 (13-17 March 2005).

[30] B.-J. Ko, V. Misra, J. Padhye, and D. Rubenstein, Distributed channel assignment in multi-radio 802.11 mesh networks, in *WCNC*, 2007.

[31] Arunesh Mishra and Vivek Shrivastava and Suman Banerjee and William Arbaugh, Partially overlapped channels not considered harmful, in *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pp. 63–74, New York, NY, USA, 2006, ACM.

[32] Aditya Akella and Glenn Judd and Srinivasan Seshan and Peter Steenkiste, Self-management in chaotic wireless deployments, in *Wireless Networks vol 13, num 6*, pp. 737–755, Hingham, MA, USA, 2007, Kluwer Academic Publishers.

[33] G. Judd and P. Steenkiste, Fixing 802.11 Access Point Selection, in *SIGCOMM Poster Session*, Pittsburgh, PA, 2002.

[34] S. Vasudevan, K. Papagiannaki, C. Diot, J. Kurose, and D. Towsley, Facilitating Access Point Selection in IEEE 802.11 Wireless Networks, in *IMC*, Berkeley, CA, 2005.

[35] P. Bahl and M.T. Hajiaghayi and K. Jain and V. Mirrokni and L. Qiu and A. Seberi, Cell Breathing in Wireless LANs: Algorithms and Evaluation, in *IEEE Transactions on Mobile Computing*, 2006.

[36] The Network Simulator - NS-2.

[37] G. Wu and T. cker Chiueh, Passive and accurate traffic load estimation for infrastructure-mode wireless lan, in *MSWiM '07: Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pp. 109–116, New York, NY, USA, 2007, ACM.

[38] Tristan Henderson and David Kotz and Ilya Abyzov, The changing usage of a mature campus-wide wireless network, in *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pp. 187–201, New York, NY, USA, 2004, ACM.

[39] Anand Balachandran and Geoffrey M. Voelker and Paramvir Bahl and P. Venkat Rangan, Characterizing user behavior and network performance in a public wireless LAN, in *SIGMETRICS Performance Evalaluation Review vol 30, num 1*, pp. 195–205, New York, NY, USA, 2002, ACM.

[40] Y.-C. Cheng *et al.*, Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis, in *SIGCOMM*, 2006.

[41] 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*, pp. 1–10, New York, NY, USA, 2000, ACM.

[42] Mathieu Lacage and Mohammad Hossein Manshaei and Thierry Turletti, IEEE 802.11 Rate Adaptation: A Practical Approach.

[43] Starsky H. Y. Wong and Hao Yang and Songwu Lu and Vaduvur Bharghavan, Robust rate adaptation for 802.11 wireless networks, in *MobiCom '06: Proceedings of the 12th*

annual international conference on Mobile computing and networking, pp. 146–157, New York, NY, USA, 2006, ACM.

[44] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, Bandwidth estimation in broadband access networks, in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 314–321, New York, NY, USA, 2004, ACM.

[45] Aditya Dhananjay and Lu Ruan, PigWin: Meaningful Load Estimation in IEEE 802.11 Based Wireless LANs., in *To appear in the proceedings of the International Conference on Communications (ICC)*, 2008.

[46] Yue Wang, A Tutorial of 802.11 Implementation in ns-2.

[47] IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements.

[48] Aditya Dhananjay and Lu Ruan, Client Unaware Load Estimation in IEEE 802.11 Wireless LANs., in *Currently under submission*, 2008.

[49] Sobrinho, J.L. and de Haan, R. and Brazio, J.M., Why RTS-CTS is not your ideal wireless LAN multiple access protocol, in *Wireless Communications and Networking Conference, 2005 IEEE*, pp. 81–87, 13-17 March 2005.

[50] Heusse, M. and Rousseau, F. and Berger-Sabbatel, G. and Duda, A., Performance anomaly of 802.11b, in *Proceedings of IEEE INFOCOM 2003*, San Francisco, USA, 2003.