DAPNET: A TESTBED FOR MEASURING THE IMPACT OF TRANSMISSION PARAMETERS ON AN 802.11 BASED DENSE ACCESS POINT NETWORK

BY

SYEDA PERSIA AZIZ

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor Nitin H. Vaidya
Abstract

There are several parameters that influence the performance of a wireless network. In this thesis, we investigate the impact of minimum size of contention window (cwmin), transmission power, and transmission rate on a dense multi-AP network. We have built a testbed called DAPnet for this experiment. DAPnet lets us change cwmin, cwmax, transmission rate, transmission power and operating channel of the nodes in the network remotely. The testbed was deployed at the Coordinate Science Lab (CSL) which is a research center at the University of Illinois. This testbed consists of 40 computing devices equipped with ath9k_htc based USB wireless cards. We studied the control flow path of the Linux Networking and ath9k_htc driver to implement the testbed. The system lets us change the parameters of all the transmitting machines by running commands remotely from a controller machine. Our experimental results show how different combination of rate and minimum size of contention window affects the throughput and packet drop rate of the dense access point network that we set up in CSL. We also show a small-scale ideal layout for dense-AP network that can improve the reliability and throughput of a dense-AP network.
Acknowledgement

I am very grateful to my adviser Dr. Nitin Vaidya for guiding me throughout the research and development process of the testbed. I also thank everyone in the business office, student labs and other offices at CSL who helped me setup the testbed. Finally I would like to thank Ahsan, my family, in-laws and friends whose continuous inspiration and support helped me finish this thesis work.

This research was supported in part by funding from the National Science Foundation and Futurewei Technologies.
# Table of Contents

Chapter 1  Introduction ........................................... 1

1.1 Dense Access Point System .................................. 3

1.2 802.11DCF .................................................. 3

1.3 Transmission Parameters ................................... 4

Chapter 2  Related Works ...................................... 7

Chapter 3  Network Architecture in Linux .................... 9

3.1 sk_buff ....................................................... 9

3.2 Network Layer: IPv4 ........................................ 10

3.3 mac80211 ................................................... 11

3.4 ath9k_htc ................................................... 11

Chapter 4  DAPnet .................................................. 13

4.1 DAPnet Architecture ........................................ 14

4.1.1 Ad-hoc network ......................................... 15

4.1.2 Controller node .......................................... 15

4.1.3 Client node ............................................... 16

4.1.4 Access Points ........................................... 17
Chapter 1

Introduction

Wireless communication has been an area of research interest for a long time. There has been research on various aspects of wireless communication for e.g. hardware, physical layer, coding, medium access protocol, wireless application etc. Now that we are entering an era of Internet of Things (IoT) a lot of aspects of wireless communication may need to be revisited to accommodate the need of an IoT application. Dense-AP environment has proven to bring benefit in terms of performance of a wireless network [7], [12], [11], [8], [6]. Before going to the details of a dense-AP network and how this can be used to improve the performance and reliability of a wireless network, it is important to know the characteristics of wireless communication. Following is a list of some of the characteristics of communication in a wireless network.

- Shared medium: Wireless is a shared medium. As the number of devices operating in the same channel increase, the contention for accessing the channel also increases. There are many protocols like TDMA (Time Division Multiple Access), CDMA (Coding Division Multiple Access),
CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), slotted ALOHA, OFDMA (orthogonal frequency division multiple access) etc. that define how the devices should share the wireless medium. We are using 802.11 standard for our experiment which employs CSMA/CA as the channel access method.

- Hidden and exposed node problem in carrier sensing methods: Hidden node problem occurs when a transmitter wireless device can not hear the transmission at the receiver end and transmits data thinking that the channel is available. As a result, a collision occurs at the receiver end. On the other hand, exposed node problem occurs when a wireless device decides not to transmit because of an ongoing transmission by a neighboring device even though that transmission is not going to interfere.

- Performance is dependent on the environment: Performance of a wireless network is heavily dependent on the physical location of the devices in the network. Presence of static or dynamic objects in the environment can cause multipath propagation, fading of the wireless signal, external noise etc. Again the relative distance between the participating devices in the network also affect the overall network performance.

- Capture Effect: Capture effect occurs when a device occupies the channel for a significant amount of time and thus causes unfairness issue.
1.1 Dense Access Point System

Dense deployment of wireless LAN is becoming an ordinary matter now. We see Wi-Fi almost everywhere including industrial, residential, commercial and academic areas. In a high-density network, there will be many users as well as many Access Points (AP). In this thesis we have investigated how the lower layer parameters impact the dense network.

1.2 802.11DCF

The wireless LANs we see today are mostly based on IEEE 802.11. The limitations of Distributed Coordination Function (DCF) of 802.11 results in low throughput, fairness issues and unreliable network. The IEEE 802.11 MAC protocol employs Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with binary exponential backoff. In this protocol a node first senses the channel before transmitting the data. If it finds the channel busy, it waits until the channel becomes idle. Sensing the channel to be idle does not necessarily mean that there is no ongoing reception in the receiver end. Once the channel becomes idle, then the node waits for a randomly chosen backoff interval before it transmits. The backoff is suspended if the channel is sensed busy during this wait time. The backoff is resumed after the channel is sensed idle again. Figure 1.1 shows the basic access mechanism of 802.11 DCF. In our experiment we disabled the Virtual Carrier Sensing mechanism. The random backoff interval is chosen between zero and CW. Here CW is called contention window. The minimum and maximum size of contention window can affect the
performance of dense-AP network.

Figure 1.1: Basic Access Mechanism of 802.11 DCF without Virtual Carrier Sensing

1.3 Transmission Parameters

- Minimum contention window: As discussed in section 1.2, smaller CWmin means that the nodes do not have to wait long before they can transmit when the channel is sensed to be idle, but if multiple nodes have small CWmin at the same time, then the probability of starting to transmit at the same time will increase. Thus collisions will increase at the receiver’s end.

- Transmission Power: High transmission power increases the transmission range of a device. A high transmission power is more likely to be sensed by contending nodes and thus reduce the probability of collision. But, a low transmission power will increase the probability of hidden terminal problem. 20 dBm is the legal limit for 802.11 b/g 2.4GHz WLAN in 20 MHz wide channels.

- Channel: Wireless is a shared medium. Every device operating on a channel is sharing the resource with other devices present on the same chan-
nel irrespective of what infrastructure the devices belong to. We can take the benefit of having multiple access points in the same channel in that it may increase the probability that at least one access point will receive the packet correctly and thus increase the reliability of the network. Of course to gain reliability in such a way there are many other parameters that we need to consider.

- **Data rate**: A higher rate means faster transmission of packet with higher probability of error or loss. On the other hand, lower rate will decrease the overall throughput. In 802.11ac we can get a data rate up to 780Mbps, but in 802.11n the highest data rate achievable is 135Mbps. Sending at a high rate transmits the data fast but also increases the probability of packet error.

- **Carrier Sense (CS) Threshold**: This parameter is used by CSMA/CA protocol for physical carrier sensing. If the energy perceived in the channel is greater than the CS threshold, then a node defers from transmitting. If the CS threshold is set to be too high then, the degree of interference will increase. For too low CS threshold the throughput will decrease because of the over-assessment of channel busy condition. [10] investigates the impact of MAC overhead on the optimal choice of physical carrier sense range. However, in our experiment we did not investigate the impact of carrier sense threshold and we used the default carrier sense mechanism used by TP-Link. We also disabled the virtual carrier sensing mechanism that requires explicit transmission of request-to-send (RTS) and clear-to-send (CTS) packets. The purpose of virtual carrier sensing is to avoid hid-
den node problem.

To understand the impact of the above mentioned parameters we have built a testbed called DAPnet. A lot of research has been done on managing the high-density networks by proper association scheme, channel assignment and power control which is described in chapter 2. We have built DAPnet that gives us more flexibility for our experiment. We have used commodity hardwares only to implement the testbed. The testbed is designed to represent an enterprise network where multiple access points are connected to the wired backbone. The main difference between DAPnet and an enterprise network is that the access points are densely deployed in DAPnet.
Chapter 2

Related Works

Performance of high-density network has been an area of interest in the research community for a long time. [7] uses path diversity relying on multiple access points and frame combining to reduce loss rate and improve throughput. [12] proposes a multi-AP architecture that has an AP controller employed to enable user to associate and cooperate with multiple AP. In [11], authors proposed an AP association algorithm for dynamically selecting APs for downlink transmission in a multi-AP system. They also propose an ACK management solution to avoid ACK collision. [8] shows how the dense deployment of access points can be exploited in various ways (for e.g. association control, channel assignment and power control) to increase the performance and reliability of wireless LAN. [6] present a framework called DAIR for managing and troubleshooting enterprise wireless networks using desktop infrastructure. They show how dense deployment of Access Points attached with the enterprise desktops can solve many wireless management problems. [9] presents an approach where the loss rate can be reduced by combining the confidence estimates received from multiple access
points to correct faulty bits in a corrupted packet. The access points send their confidence values about the ongoing transmission to the combiner over wired Ethernet. The multi-AP systems mentioned above do not take into account the lower layer parameters for performance measurement. We want to investigate the impact of these parameters that affect the performance of CSMA-based protocol in a dense-AP scenario.
Chapter 3

Network Architecture in Linux

The networking core in Linux kernel is vast and includes a lot of functionalities relevant to various networking protocols. The kernel contains the implementation of Transport Layer Protocols (UDP and TCP), IP layer (routing, IPv4, IPv6 etc.), driver interface, MAC sublayer (mac80211, mac802154), ieee802154, 8021q VLAN, DNS etc. It is necessary to understand the packet walk-through in Linux kernel. This chapter contains a short and high level description of the portions of kernel networking code that we required to build our testbed.

3.1 sk_buff

sk_buff (Socket Kernel Buffer) is the common data structure for incoming and outgoing packets that is used by the whole networking subsystem present in the kernel stack. This structure is defined in /include/linux/skbuff.h. It is defined as a doubly linked list so that it is easy to access the elements in the list. This structure contains the pointers to the transmitting or receiving device object,
data head, headers of transport layer, network layer and MAC layer of a packet frame. Linux kernel has several queues in its networking stack to buffer packets. A sk_buff queue in the networking layer of the kernel is of type sk_buff_head. sk_buff_head is a structure which is also defined in /include/linux/skbuff.h. sk_buff_head contains head ant tail pointers to sk_buff elements.

3.2 Network Layer: IPv4

The subsystem related to the IP sublayer can be found in Linux-x.x.x/net/ipv4/. This sublayer is responsible for routing, forwarding and other actions that requires the IP layer information of a packet. The main receive and send subroutines are written in ip_input.c and ip_output.c. ip_build_and_send_pkt is the subroutine that builds the IP header (struct iphdr) and sends packet to the next layer in Linux kernel stack. The IP header of each packet contains a 32 bit identifier. This identifier is not sequentially selected rather Linux uses true random value for security reason. The identifier is assigned in the subroutine _ip_select_ident in /net/ipv4/route.c. We used the tuple < source IP address, Packet ID > to uniquely identify the packets in the network. int ip_rcv (struct sk_buff * skb, struct net_device * dev, struct packet_type * pt, struct net_device * orig_dev) is the main subroutine called by lower handler for incoming packets. We modified this function for our experiment which is described in the Chapter 5.
3.3 mac80211

mac80211 is a subsystem inside the Linux kernel responsible for dealing with 802.11 softMAC devices. SoftMAC devices require the software in host device to implement the MAC sublayer Management Entity (MLME). On the other hand, devices of type FullMAC implement the MLME functionalities in their firmware or hardware hiding the complexities of 802.11 protocols. Most of the wireless 802.11 devices are softMAC type since it allows a fine control of the device. The source code for the mac80211 subsystem can be found in Linux-x.x.x/net/mac80211/. The receive and transmission routines are defined in rx.c and tx.c respectively. The main receive path handler is defined by the subroutine called void ieee80211_rx (struct ieee80211_hw *hw, struct sk_buff *skb). Lower level driver calls this handler when a 802.11 frame is received by the hardware. Rate control algorithms like minstrel and PID (proportional–integral–derivative) are also implemented as a part of mac80211 subsystem. The source code is compiled into kernel object module called mac80211.ko and placed in /lib/modules/kernel_version/kernel/net/mac80211. This module also works as an intermediate interface between userspace and driver. It receives the user defined configuration via APIs like CFG80211 or Wireless Extension and passes down to the driver to configure the wireless cards [5].

3.4 ath9k_htc

For this testbed we used TPLink-721N wireless USB card which is an atheros based USB card. This card requires the ath9k_htc driver which is open-source
and can be found in the modern Linux source distribution. The firmware of this driver is also open source. The source code of this driver can be found in Linux-x.x.x/drivers/net/wireless/ath/ath9k/. The main subroutines for configuration and mac80211 callbacks are written in htc_drv_main.c. This driver receives the parameters from mac80211 and passes them to the firmware. The configurations for minimum and maximum size of contention window and arbitration inter-frame spacing (AIFS) are set in the subroutine static int ath9k_htc_conf_tx(struct ieee80211_hw *hw, struct ieee80211_vif *vif, u16 queue, const struct ieee80211_tx_queue_params *params). The subroutines related to transmit and receive process can be found in htc_drv_txrx.c. ath9k_htc driver prepares a rate table containing the all possible rates and send the table to the firmware. The firmware uses this table for its rate control algorithm. static void ath9k_htc_setup_rate(struct ath9k_htc_priv *priv, struct ieee80211_sta *sta, struct ath9k_htc_target_rate *trate) is the subroutine where the rate table is built. We modified this driver and developed a module that lets us use a fixed rate during transmission.
Chapter 4

DAPnet

The testbed DAPnet consists of the following elements

- ASUS PC: There are 16 ASUS computers in the system. Ubuntu 14.04 was installed in all the ASUS computers.

- Raspberry PI: 20 Raspberry PI 2’s (Model B) are used. This model comes with 1 GB RAM and 900 MHz Quad-Core CPU. The operating system installed in the PI’s is Raspbian. The kernel was modified using a cross-compiling technique since compiling in a Raspberry PI itself is a very time-consuming process.

- TP-Link 721N: These wireless USB cards were used to build the wireless network. The reason for choosing this card is that, these cards are based on Atheros AR9271 chipset. These cards require ath9k_htc driver. The ath9k_htc driver is open-source and can be found in the recent Linux distributions.

- Dell Inspiron 1545: 5 Dell Inspiron 1545 laptops were used in this testbed.
These laptops are also running Ubuntu 14.04.

4.1 DAPnet Architecture

There are four types of components in this testbed named controller, client, combiner and access points (Figure 4.1). All the components are connected via a VLAN. This VLAN is part of the internal wired backbone of CSL. The network devices attached to the computers in the testbed are automatically configured with default parameters during the bootup.

![DAPnet Architecture Diagram]

Figure 4.1: DAPnet architecture
4.1.1 Ad-hoc network

The clients communicate with the AP via an ad-hoc network. The wireless interface of each of the APs and clients connect to an ad-hoc network during boot up. This means that there is no infrastructure in the wireless network. Every node broadcasts beacon frames to let others know of its existence. Using ad-hoc mode automatically employs 802.11g standard for communication. As a result the data rate is limited to 54 Mbps. The datarates supported by 802.11g are 6, 9, 12, 18, 24, 36, 48 and 54 Mbps. 802.11g operates in 2.4 GHz band and utilizes a channel bandwidth of 20 MHz. Each node in the ad-hoc network are made to perform either as a client or as an AP by configuring them properly.

4.1.2 Controller node

The controller initiates the experiments and changes the parameters of client nodes. The controller communicates with the client nodes and the combiner via the VLAN. The controller sends the commands to the computers using SSH and SSHpass. The controller supports the following actions

- Start java application in all the client nodes with appropriate parameters.
  This java application generates the network packets.

- Start new experiment by sending commands to the client nodes.

- Shutdown or reboot the whole network

- Change the minimum contention window, rate, channel, transmission power, default gateway and transmission queue length of all the client nodes.
• The controller also changes the default and maximum size of transport layer read and write buffer of the client nodes.

• Notify the combiner when one experiment is finished so that the combiner can extract the resultant throughput and packet delivery fraction of that experiment.

• Check if all the computers are alive in the network.

• Shut down the java application in the client nodes

4.1.3 Client node

Client nodes generate the uplink data destined for the combiner and uses their wireless interface to send the packets. Every client runs a java application that is responsible for generating the network packets after receiving the start command from the controller. This application generates back to back UDP (User Datagram Protocol) packets and hands them down to the socket. UDP is a connectionless protocol with no in-built reliability mechanism. It is appropriate for our experiment since we want to measure reliability of the wireless communication in the system without the intervention of any other reliability mechanism enforced by other protocols like TCP (Transmission Control Protocol). The java application is multi-threaded with separate threads for sending and receiving packets. The client nodes use a startup script to configure themselves during reboot or startup. However, the java application required to run the experiment is not launched during booting, rather the controller starts the application in each client computer with the appropriate parameters like number of packets to gen-
erate, destination address, Ethernet IP address and WLAN interface address.

4.1.4 Access Points

There are two types of access points in the system.

- Primary AP: These APs serve as the default gateways of the wireless network. Each client associates with a primary AP as the default gateway to reach the combiner. These APs have the IP forwarding enabled and forwards the testbed packets to the combiner.

- Secondary AP: These APs listen for ongoing transmission in the wireless network in promiscuous mode and forward the captured packet to the combiner. As a result the combiner receives multiple copies of the same packet. We want to investigate that whether we can rely on these secondary APs to forward the correctly captured packet and thus increase the reliability of the wireless network in the face of too many collisions happening in the wireless network at the primary APs’ end.

4.1.5 Combiner gateway

Combiner will act as the gateway connected to the main Internet backbone. Because the combiner receives multiple copies of the same packet from the secondary APs, it discards any duplicate packet that has already been processed. In our testbed, the combiner is connected to the access points via the same VLAN. For our experiment, we have developed a UDP application in Java which acts as the final destination of all the network packets and calculates the resulting throughput and packet delivery fraction. The combiner gets notification from
the controller when an ongoing experiment is finished. This notification also implies that the combiner should start a new experiment. The combiner keeps track of the arrival time of the first packet and the last packet in an experiment. After a notification from the combiner is received, it uses the time difference to calculate the throughput i.e. the number of bytes received per second.

4.2 Kernel Modules

Two kernel modules were written for changing the minimum contention window and the data rate. After running these modules, the wireless cards have to be reconfigured again since these USB cards get the configuration settings from the MAC layer of Linux kernel only when connecting to a network.

4.2.1 Minimum Contention Window

A module for setting the minimum and maximum size of contention window has been developed and installed in all the client nodes. The controller executes a bash file in each client node which loads this module. This module then assigns the values passed as parameters to the kernel variables that hold the size of minimum and maximum contention window. This module resides in the mac80211 subsystem of the linux kernel.

4.2.2 Rate

TP-Link 721N wireless cards do not obey the rate specified by the user since they have a default ath9k_htc_rate_control algorithm running in their firmware.
The way to make the wireless cards use a constant rate is by changing the rate table that the firmware receives from the ath9k_htc driver. This table includes the rates that are supported by the mode of communication. The rate control algorithm works on this table to select the appropriate rate for impending transmissions. We modified the kernel so that all the elements of this table contain the same value which is equal to the rate we want to set for our experiment. We have written a module which takes the rate as a parameter from the user space and assigns it to all the elements in the rate table of ath9k_htc driver. This rate table is passed down to the firmware when the card connects to a network. Hence whenever we want to change the rate, we need to reconfigure the wireless card.

### 4.3 Network Configuration

The following list shows the configuration used for different components of the testbed. Several configuration tuning has been done to minimize the possibility of buffer being overflowed in the Linux kernel stack of combiner and access points.

- **Size of backlog buffer**: The number of packets that can remain unprocessed in the kernel memory net.core.netdev_max_backlog. The buffer has been set to be 16777216 bytes (16MB) in all the AP’s and combiners. This setting is useful when the rate at which kernel can process the incoming packet is slower than the rate at which the packets arrive.

- **Size of the receive buffer**: The UDP receive buffer size has been specified in
net.core.rmem_max to be 16777216 bytes (16MB). In our experiment the clients send 10,000 packets of 1400 bytes each. So the total outgoing data is 14MB which is less than 16MB.

- Transmit Queue Length: The transmit queue length in the wireless interface has been set to 10000. This setting specifies the number of packets pointed to by the transmit queue of the interface.

- Frequency of checking for stale neighbors: The kernel checks for stale neighbors repeatedly to remove them from the entry. This interval of checking for stale neighbors is set to 3600sec.

- Promiscuous mode: The wireless cards of the secondary access points are set to promiscuous mode to listen for ongoing transmission and forward the received packets to the combiner.
Chapter 5

System Implementation

This chapter contains the flow chart, implementation details of each of the components of the testbed. Figure 5.1 shows a picture of our testbed.

![Figure 5.1: DAPnet Testbed](image)

5.1 Controller

The controller simply sends commands to the different components of the system and configures the senders and AP’s according to the experiment specification.
It communicates via SSH and SSHpass. A dell inspiron laptop serves as the controller in this testbed. All the shell scripts developed to do the actions listed in the Subsection 4.1.2 are stored in this laptop.

5.2 Client Nodes

The ASUS minicomputers and the dell inspiron laptops are used for the client nodes. The client nodes’ ubuntu startup scripts have been configured in such a way that they can automatically connect to the ad-hoc network with the appropriate network parameters like IP address, frequency, SSID etc. when they are powered on. As discussed in section 4.1.3, each client runs a java application that generates the network packets. The jar executable of that java application is stored in a specific location of the computers known to the controller. Also the scripts that turn on the modules for changing the size of minimum contention window and rate are placed in a specific location for the controller to access. Figure 5.2 shows how the application operates in the client nodes.

5.3 Access Points

For our testbed, the APs need to be able to forward the overheard packets to the combiner. But by default, the IP layer of Linux kernel drops the packets received in promiscuous mode. The socket kernel buffer structure (sk_buff) contains a field called pkt_type which holds the nature of the packet. The list of packet classes can be found in [4]. If the pkt_type is of type PACKET_OTHERHOST, then the kernel knows that this packet does not be-
long to it and drops the packet before it reaches the forwarding routine. To allow the captured packets flow through the forwarding routine, the main receive routine in IP layer has been modified. Figure 5.3 shows the flow added to the `ip_recv()` routine in `ip_input.c` to allow the testbed packets that were received in promiscuous mode by setting the field `pkt_type` to `PACKET_HOST`. Lower layer drivers set the field `pkt_type` to `PACKET_HOST` when a packet is intended for the access point.

The ASUS PC’s served as the primary APs while Raspberry PIs were used to build the secondary AP. Raspberry PIs are relatively slower than ASUS PC’s, but they can forward a correctly captured packet reliably. The problem with using Raspberry PI as primary AP is that these PIs have low buffer size. Even if we increase the buffer size, the slow processor can not process the bursty traffic quickly and thus affecting the overall throughput.
5.4 Combiner

The combiner receives all the packets from the APs, discards the duplicates and passes the packets to the host UDP socket. In our implementation, the packet combining happens in the IP sublayer of Linux Networking stack. Each packet is uniquely identified by the tuple <Source IP address, IP Identifier>. The way Linux generates the packet id is discussed in section 3.2. Figure 5.4 shows the flow chart of the code added to the main receive subroutine of IP system for packet combining. The pseudocode presents the insertion and search functions. However, to avoid function calls, we placed the code inside the main receive rou-
The IP layer receives the packet as a socket kernel buffer structure `sk_buff` which has a pointer to the IP information of the packet. After extracting the source IP address and packet identifier, the kernel searches in a circular buffer to see if the kernel has already seen this packet. This circular buffer can hold 50 elements for now. This buffer is an array of structures where each structure element contains a tuple of IP address and packet identifier (Figure 5.5).
5.4.1 Search for Duplicates

The combiner maintains a pointer to the last index \( (C_i) \) where a new packet information has been stored. To find out if a new incoming packet is a duplicate, the combiner searches backward in the circular buffer starting from \( (C_i) \). This method is efficient because the kernel starts looking from new to old packet information. Because it is highly likely that the duplicate packet arrives at the kernel almost at the same time, this method saves a lot of time. If the packet is not a duplicate, a new tuple containing the packet source IP address and packet identifier is inserted into the next index in a circular manner. Because we want
to measure the reliability and throughput of a dense AP network, we do not want the combiner to be a bottleneck. Hence a high-performance computer has been selected as the combiner machine.

## 5.5 Kernel Modules

### 5.5.1 CWmin

The module for changing cwmin was developed as a part of mac80211 sublayer. The subroutine

```c
void ieee80211_set_wmm_default (struct ieee80211_sub_if_data * sdata, bool bss_notify)
```

assigns the configuration values in the structure `struct ieee80211_tx_queue_params`. This structure contains the field for minimum (`aCWmin`) and maximum (`aCWmax`) size of contention window.

We defined two variables corresponding to cwmin and cwmax which are accessible from our module. These two variables are assigned the module parameters each time the module is loaded. Then when the card is reconfigured,

```c
void ieee80211_set_wmm_default (struct ieee80211_sub_if_data * sdata, bool bss_notify)
```

assigns the values contained in the two variables to `aCWmin` and `aCWmax`. The driver ath9k_htc then passes the `struct ieee80211_tx_queue_params` to the hardware.

### 5.5.2 Fixed Rate

For our investigation we needed a fixed rate during the data transmission phase of an experiment. We wrote a module for this purpose as a part of
the ath9k_htc driver. ath9k_htc firmware has a default rate control algorithm running which can neither be controlled nor can be stopped from user space. In order to understand how the firmware gets configuration about the rate control algorithm, we dug through the part of ath9k_htc driver where the rates are set up. We found that ath9k_htc has a rate table where the driver stores the supported rates by a communication mode and then sends the rate table to the ath9k_htc firmware. This rate table is defined as \texttt{u32 supp_rates[IEEE80211\_NUM\_BANDS]} in \texttt{struct ieee80211\_sta} in mac80211.h. \texttt{static void ath9k_htc\_setup\_rate(struct ath9k_htc\_priv \ast priv, struct ieee80211\_sta \ast sta, struct ath9k_htc\_target\_rate \ast trate)} is the main subroutine that builds the rate table. This rate table is then used by the firmware along with its rate control algorithm to decide on rates for subsequent transmissions. We defined a variable accessible from our module. The value contained in the variable is assigned to all the elements of the rate table. As a result the rate control algorithm of the firmware now works with fixed rate only.

### 5.6 Challenges and Problems

We faced several challenges while implementing the testbed. Most of the challenges were related to the wireless card we selected for our wireless transmission. Following is a list of some of the challenges that are worth mentioning.

- The first problem is that ath9k_htc has its own rate control algorithm implemented in the firmware which we could not turn off from the user space. For this experiment we need to have fixed transmission rate. We could have used other cards which has no rate control in their firmware,
but the main reason behind using this ath9k_htc card is that, the htc_9271 firmware is open and we found that the coding implementation of ath9k_htc driver is better than other drivers. We tried several cards with different drivers including intel(iwlwifi), broadcom( BCM94323U) and RALink(rt2800), but decided to use the tplink cards. We were able to control the transmission rate by changing the source code of ath9k_htc driver inside the linux-3.13 kernel.

- The second problem is a limitation of ath9k_htc based cards in that these cards can only support upto 7 stations in AP mode and can handle only 7 peer stations in ad-hoc mode [1]. This limitation was found later when we fully deployed the testbed in the 4th floor of CSL. The network used to become unstable after some time without any implicit or explicit notification and thus rendered a lot of experimental results useless. This problem was tracked down by looking at the debug messages logged by the kernel. ath9k_htc driver keeps track of the neighboring stations and stores their information. When the number of neighbors become more 8 (defined as ATH9K_HTC_MAX_STA), then the kernel returns an error [2]. But the problem is that the card itself fails and the kernel driver keeps trying to reset the chip without any success. Somehow, the wireless card still broadcasts their beacon frames which make the ad-hoc network visible, even though the network is practically out of order. Because we want a dense network with a lot of APs and clients, we found a way around by making clusters of ad-hoc networks on the same channel with different SSIDs.

- The next challenge was to understand the data transmission path inside
the linux kernel for changing the transmission parameters. The modern
kernels have CFG80211 as the configuration API for 80211 devices and
NL80211 as the netlink interface header. NL80211 has functions defined
for changing different wireless parameters. The command line tool iw is
built using nl80211 and CFG80211 but does not support changing cwmin
and cwmax from the user space. Older tools like iwpriv used for chang-
ing the private parameters of wireless device have been deprecated. Also
ath9k_htc does not let iwpriv to access its private parameter. So we wrote
a separate module that can change these parameters when the cards are
configured.
Chapter 6

Experimental Results

6.1 Performance Metrics

- Throughput: Throughput was calculated as the number of bits received per millisecond by the UDP application running at the combiner. The UDP does not receive the duplicate messages since duplicates are filtered in the IP sublayer of the kernel. The UDP application keeps track of the time of first received packet and the last received packet in an experiment. After each experiment, the controller notifies the combiner that the experiment is finished. The combiner then takes the time difference between the arrivals of first and last packets, $T$. Then the throughput for that experiment is calculated as $[8 \times \frac{\text{Number of bytes received}}{T}]$.

- Packet Delivery Fraction (PDF): Packet delivery fraction is defined as the ratio of number of packets received by the UDP application of the combiner and the total number of packets sent by all the senders. The higher the ratio is, the more reliable a network is.
Every value presented in the graphs is averaged over 15-20 runs of the same experiment. In all the experiments except the one where we tested transmission parameters, we used 0 dBm transmission power. This was done so that the diameter of the coverage cell around a transmitting device is minimum.

6.2 Experimental setup

The testbed was deployed in CSL (Coordinated Science Lab). We used student labs, business office and office rooms that are easily accessible. We had two setups for our experiment. Setup1 consists of 10 Clients (Sender node) and 15 APs whereas setup2 contains 14 clients and 15 APs. Table 6.1 shows the two setups on floor-map of CSL. Figures in table 6.1 shows the two setups we used for our experiment. Since there are more sender nodes in setup2, the amount of contention is larger than that of setup1. All the rooms and labs have furniture, computers and electronic devices and other obstacles that have impact on the wireless signal. All the nodes operate on a single channel.

Table 6.1: Experimental setup: a) Setup1 with 10 clients and 15 APs (Left) b) Setup2 with 14 clients and 15 APs (Right)
6.3 Impact of size of minimum contention window (CWmin)

The experiment to test the CWmin was done on both the setups. Table 6.2 shows the throughput and PDF obtained in this experiment. Following are the configurations used for this experiment:

- Power: 0 dBm
- Data Rate: 54 Mbps
- Number of packets per client: 10000
- Virtual Carrier Sensing Disabled
- ARP enabled
- Packet Size: 1400 bytes

The CWmin was varied from 3 to 255. Setup 2 has more contention than setup 1. Throughput for setup 2 is higher than that of setup 1, on the other hand packet loss in setup 2 is higher than setup 1. Because throughput is calculated as the time difference between first packet and last packet in an experiment, this means that for setup 2 network utilization is higher than setup 1. At the same time the amount of collision is also higher in setup 2. Packet delivery fraction seems to be increasing monotonically with CWmin for setup 2. An improvement in the results for setup 2 may be obtained if we could lower the power from 0 dBm, so that the number of collisions decreases.
Table 6.2: Throughput and PDF obtained varying the size of Minimum Contention Window

6.4 Impact of Data Rate and CWmin

Following are the fixed configuration used for this experiment

- Power: 0 dBm
- Number of packets per client: 10000
- Virtual Carrier Sensing Disabled
- ARP enabled
Table 6.4 shows the throughput and PDF obtained for different rates varying over different CWmin. Since we have an ad-hoc network, the mode of communication is 802.11g. Hence, we varied the rate from 9 Mbps to 54 Mbps. This experiment was done on setup2 only. One interesting finding is that for setup2 a rate of 36 Mbps with 15 as the CWmin resulted in best throughput and a comparable PDF. We can see that for each CWmin, the throughput initially increases with the decrease of rate and reaches the peak at the rate of 36 Mbps. Then the throughput starts decreasing again. For a data rate of 9 Mbps, we got the poorest throughput and packet delivery fraction.

6.5 Impact of transmission power

- CWmin: 15
- Rate: 54 Mbps
- Number of packets per client: 10000
- Virtual Carrier Sensing Disabled
- ARP enabled
- Packet Size: 1400 bytes

This experiment was done on setup 2 as well. For this experiment we varied the transmission power of the wireless interfaces between 0 to 20 dBm. 20 dBm is a legal limit imposed by US FCC for 802.11b/g wireless LAN. The cards do not
Table 6.3: Throughput and PDF obtained varying rate and CWmin

accept a txpower under 0 dBm. We see a slight improvement in the throughput and packet delivery fraction as the power is increased. This result implies that probably most of the APs encountered the same collision when the clients were sending at a lower power. It would have been interesting to see if we could use a power lower than 0 dBm so that we could get more exclusive coverage zone around the client nodes.
6.6 Reliability test

We did a simple test to see how dense AP can actually improve reliability of a wireless network. This basically means that any transmission that collided at the receiver end was correctly captured by at least one AP. The problem with setup1 and setup 2 is that we had space constraint. The VLAN in CSL was created with specific ports. So we had to place some of our nodes together near the VLAN ports. Hence there is a high probability that the transmission colliding at the primary AP’s end might also collide at the nearby secondary AP’s end.
Hence we wanted to have a setup that shows that we can benefit from these secondary APs if we place them near the sender nodes that are far from primary APs. In a real dense-AP scenario, it is highly likely that the primary APs will be installed in a fixed location. Thus if there are many transmitting devices in the network, there will be some devices who are far from the primary AP. If we can place a secondary AP away from the primary AP and place them near the transmitting devices, this should increase the reliability and throughput of the network. To do this test, we setup the network in our lab as illustrated in Figure 6.1. The distance between each client and its gateway primary AP is shown by drawing an arrow from the client to the primary AP. The primary APs were intentionally placed far but within the range of the clients, where the secondary APs were close. We did two experiments. The first experiment contained only the two primary APs. The secondary APs were turned down. The second experiment contained 2 primary APs and 2 secondary APs, all up and running. We had the following transmission parameters throughout the experiment.

- Data rate: 54 Mbps
• CWmin: 15
• Transmission power: 0 dBm
• Virtual Carrier Sensing Disabled
• ARP enabled
• Packet Size: 1400 bytes

Table 6.5 shows the improvement we obtained by increasing the number of AP’s. Just by increasing the number of APs from 2 to 4 increased the packet delivery fraction (about 16% increase) and throughput (about 26% increase)

6.7 Discussion

There are several factors that might have affected the throughput and PDF measurement of our experiments.

• Firstly, the MAC retransmission was enabled during all the experiments. So even though a secondary AP captured and forwarded a packet to the combiner, if the sender does not get acknowledgement for that packet from the primary AP, the sender will keep retransmitting. We could not turn off the MAC retransmission of the wireless cards. MAC retransmission is implemented in the firmware of ath9k_hwc cards. The cards do not obey the user-space command for changing MAC retry limit.

• Second, the throughput was calculated in the Java application running in combiner. This application has a single receiver thread that extracts IP information from the packet buffer to identify if the packet is a data packet
or a control packet from the controller. Thus, the application behavior may also degrade the measured throughput.

- Third, the Ethernet communication in our testbed is done using the CSL VLAN. We had to place some of our nodes near the VLAN ports. Hence we could not achieve an evenly distributed layout. So some of the secondary APs had to be placed near the receiver primary AP. Hence some of the secondary APs faced the same collision the primary APs faced. Also, we placed some machines in the student labs that contain a lot of static
objects including furniture and electronic devices which may have affected the wireless communication.

- We could not find a way to turn off the rate control algorithm from the user-space. We made the rate control algorithm use fixed rate by modifying the rate table to contain the same rate value in all the elements of the table. The fact is that the rate control algorithm is still running in the firmware and we did not find any information about the complexity of the algorithm which is running on a low-memory Atheros AR9271 chipset [3]. It is necessary to know how the rate control algorithm behaves with dynamic changes in the channel condition in order to find out if it might affect the throughput.
Chapter 7

Conclusion

In this thesis, we have implemented a testbed called DAPnet for measuring the impact of transmission parameters on a dense-AP network. We have configured 16 ASUS mini computers, 20 Raspberry Pis, 4 Dell Laptops and 1 Dell desktop for this testbed. The devices are equipped with ath9k_htc based USB wireless cards. There are four components in the system: client, access point, controller and combiner. The controller can configure all the devices in the network via SSH. It also initiates experiments and notifies the combiner when an experiment finished to instruct the combiner to calculate the throughput and PDF obtained for that experiment. Access points forward packets to the combiner. Combiner gets multiple copies of the same packet from multiple AP’s but it only forwards the first corresponding packet to its UDP receive application. Two modules have been implemented that can change the minimum and maximum contention window and transmission rate of transmitting devices. We studied the Linux network packet walk-through in order to build the modules. The Linux kernel inside the access point devices was modified to allow forward-
ing of the packets captured in promiscuous mode. We also modified the kernel in combiner so that duplicate packets do not reach the UDP receive application. We did some preliminary experiments to measure the throughput and packet delivery fraction by varying CWmin, data rate and transmission power. The results show that a CWmin of 15 with data rate of 36Mbps results in the highest throughput and PDF for our configuration and layout of the wireless network. We also did a reliability test that shows that placing the secondary APs close to the clients and away from primary APs can increase the throughput and reliability of a dense-AP network. Otherwise if we place the APs close to each other, it is highly likely that all the APs will face the same collision.

There are many parameters in 802.11 based protocols that are involved in data transmission in a wireless network. For this investigation, we did not take the receiver sensitivity into account. It would be interesting to see how various carrier sense thresholds behave in dense-AP network along with CWmin, transmission rate and transmission power. These parameters can be selected based on the layout of the wireless network to improve the performance. It would also be interesting to find out a utility function of these lower layer parameters that can result in the best possible performance based on the layout and other configuration of a 802.11 based dense-AP wireless network.
References


