University of Twente
Enschede - The Netherlands

On Access Network Identification and Characterization

Master Thesis

Rafael Ramos Regis Barbosa

Telematics Programme (MTE)
Chair for Design and Analysis of Communication Systems (DACS)
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
University of Twente, The Netherlands

Supervisors:
Dr. ir. Pieter-Tjerk de Boer (UT/DACS)
Dr. ir. Geert Heijenk (UT/DACS)
Dr.ir. Aiko Pras (UT/DACS)

June 2009

"There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after."

J.R.R. Tolkien

Abstract

The proliferation of portable computing devices, such as laptops, netbooks, PDAs and smart phones, increased the demand for wireless network connectivity. The deployment of IEEE 802.11 Wireless LANs (WLANs) appears as an attractive solution for providing network connectivity in enterprises and universities, and in public places like conference venues and airports. The use of 802.11 WLANs brings new challenges to network administrators. They need to understand the extent of wireless usage to properly allocate the networks resources, such as Access Points (APs).

In this report we propose a novel approach for the identification of access network types from passive measurements performed in an aggregation point of the network backbone. Based on basic characteristics of Ethernet and 802.11 protocols, like transmission rates of the links and duplex capabilities of the medium, we show that it is possible to distinguish TCP flows that cross these types of networks. As a side effect, our method provides information on the transmission rate in which the protocols are operating in. We validate our findings using traces generated in a semi-controlled laboratory environment and "real-world" traces collected at our university campus network.

Acknowledgements

I would like to thank everyone that somehow helped me finish this thesis. To thank my advisors Pieter-Tjerk and Geert for our discussions in (*quasi*) weekly meetings and constructive comments on the report, even on short notice. I would like to thank Aiko that provided some useful ideas that are part of this work. I would also like to thank my friends and my girlfriend Aleksandra that made this time in the Netherlands two of the best years of my life. Last but not least, I am grateful to my parents Florencio and Angela, who always supported me, even when I decided to study overseas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The proliferation of portable computing devices, such as laptops, netbooks, PDAs and smart phones, increased the demand for wireless network connectivity. The deployment of IEEE 802.11 Wireless LANs (WLANs) [1] appears as an attractive solution for providing network connectivity in enterprises and universities, and in public places like conference venues and airports.

802.11 networks bring some new challenges to network administrators. As wireless nodes tend also to be mobile, they can access the network from different points in a short period of time, affecting different parts of the infrastructure. It is also hard to guarantee Quality of Service (Qos) parameters in these environments due to the unpredictable nature of the wireless medium. It is useful to understand the extent of wireless usage to properly allocate the networks resources, such as Access Points (APs).

The main goal of this work is to derive traffic characteristics from wireless and wired connections based on passive measurements performed in an aggregation point of the network backbone. We describe a method to differentiate hosts using these access networks based on differences in their transmission rates and duplex capabilities. As a side-effect, we are also able to infer the transmission rate at which these hosts access the network.

Identifying the access network type from this types of measurements is not an easy task to perform. The Medium Access Control (MAC) headers contain useful information regarding the network type, however they are replaced in every link crossed in the communication path, thus are not available in the monitoring

point. Our method extracts a timing *signature* created by the access link, based in measurements at the TCP/IP level. Moreover, wireless access points are not invisible to conventional topology discovery tools such as *tracepath*, as they operate at the IP level.

A possible application of our method is the discover of unauthorized access points. The detection of wireless connections in portions of the infrastructure reserved to wired access indicates the existence of such rogue access points. Besides representing a clear security problem, they may interfere with other nearby APs, causing also performance problems.

Our technique also can be used to monitor the performance of the wireless networks. The identification of links operating at lower than expected transmission rates provide useful information to network administrators, as adding new APs or simply repositioning the existing ones could make the network perform better and increase radio coverage.

1.1 Problem Statement

Consider the scenario depicted in Figure 1.1: a local network that consists of wired and wireless clients. This network is connected via an arbitrary topology to servers in the Internet. A network traffic monitor is deployed in this local network in a way that it is able to capture packets exchanged between clients and servers in both directions. Our main objective is to be able to differentiate connections made from wired Ethernet and wireless 802.11 clients based on timing characteristics of the captured traffic. We are also interested in deriving some characteristics from the wireless networks, such as the transmission rates in which they are operating.

Our method is based on the behaviour of *Transmission Control Protocol* (TCP) connections in which most of the data is downloaded from the servers, and consequently, TCP Acknowledgements (ACKs) are generated in response by the clients. By studying the interval of consecutive ACKs in the monitoring point we observe that the duplex capabilities and transmission rates of Ethernet and 802.11 protocols leave a *signature* that allows us to distinguish them.

Two basic assumptions are made in this work. The first is that the access network link is the one with the smallest capacity, i.e. smallest transmission

rate. The presence of a link with smaller capacity on the path from the client to the monitoring point would destroy the timing signature created by the access network link. It is reasonable to assume that the access link is the bottleneck, as core networks are normally over provisioned.

The second assumption is that the monitoring point is near to the clients, this is to limit the effect of cross-traffic in our measurements. The closer the monitoring point is to the client the clearer characteristics from the access network can be visualized. We believe that performing the capture in some aggregation point near the client it is not a difficult task.

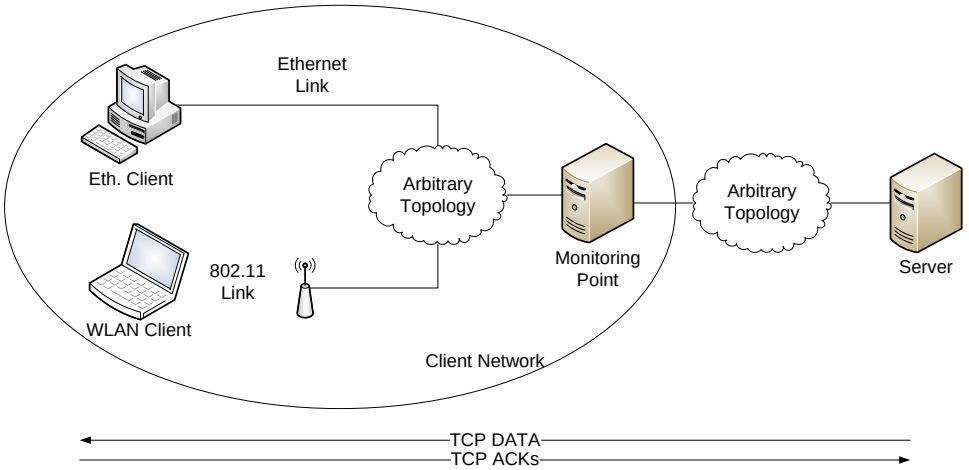


FIGURE 1.1: Problem scenario

1.2 Approach

The following approach is used in this research work. With a literature study we collect detailed information on the functioning of the access network protocols considered: Ethernet and 802.11. We also perform a review of the TCP acknowledgement mechanism, which is of great importance for our identification method. This literature study includes a research on the state of the art in access network identification.

Using two public available data sets [2, 3] and measurements performed in a semi-controlled laboratory environment we perform a series of experiments that allow us to see how the studied protocols behave in practice. Based on the results obtained, we describe a visual access network classification method described in Chapter 5.

Finally we validate our findings with further experiments in our laboratory and "real world" traces collected at our campus university.

1.3 Outline

The remaining of this document is organized as follows. In Chapter 2 we provide some background information on the problem of access network identification, including a review of Ethernet, 802.11 and TCP protocols, an overview of related work and a short description of the used data sets.

In Chapter 3 we discuss our attempt to reproduce the results of one of the related works, presented in [4].

The works proposed in [5, 6] serve as a starting point for Chapter 4, where we perform a series of studies on the ACK inter-arrival times.

This preliminary work leads to the description of the distribution of ACK inter-arrivals in Chapter 5. This chapter includes a detailed discussion of the effects of transmission rates and duplex capabilities on this distribution. Also some practical issues are discussed.

In Chapter 6 we present the validation our method, using traces generated on semi-controlled laboratory experiments and *real-world* network traces collected at our university.

Finally in Chapter 7 we present our conclusions and future work.

Chapter 2

Background Information

In this chapter we introduce some background information related to this thesis. In Sections 2.1, 2.2 and 2.3 we discuss important aspects on the studied protocols: Ethernet, 802.11 Wireless LAN and TCP. Section 2.4 we discuss some of the related work found in the literature. Finally, in Section 2.5 we describe the data sets used throughout this work.

2.1 Ethernet

Ethernet is the *de facto* standard for wired Local Area Network (LAN) originally designed by Bob Metcalfe and David Boggs in the mid-1970s. In its first version, it used a coaxial bus (the *ether*) to interconnect the nodes. In this topology the bus serves as a broadcast LAN, where all transmissions are received by all nodes connected to it. At a given time only one node can be transmitting, otherwise a collision occurs, thus this medium is said to be half-duplex. The *Medium Access Control* (MAC) mechanism is defined by the *carrier sense multiple access with collision detection* (CSMA/CD) protocol. The basic functioning of this protocol can be described as follows [7]:

1. If the channel is idle, an adapter may transmit at any time, that is, there is no notion of time slots.

2. If the adapter senses a transmission from another adapter it never starts a transmission, that is, it uses carrier sensing. As soon as the medium is free again, the adapter can start its the transmission.

3. If a collision is detected, the adapter aborts the current transmission and transmits a jam signal to make sure that every other adapter will also detect the collision, that is, it uses collision detection.

4. After a collision an adapter waits for a random interval of time before attempting a retransmission. This is said to be an *exponential backoff* phase as, the interval increases exponentially with the number of collisions. Specifically, after experiencing the nth collision, the adapter picks a random number K at random from the contention window $\{0, 1, 2, ..., 2^m - 1\}$ where $m = min(n, 10)$. The adapter then waits for a period equal to $K \times 512$ bit times before attempting a retransmission.

Note that this protocol suffers from an unfairness issue. Following item 4 above, when a collision occurs, each node has to wait for a random period before a attempting a retransmission, and, as the contention window increases with the number of attempts, this period also tends to increase. The problem occurs when a node continues to "win" the dispute for medium access.

For example, consider the case where a node A and a node B try to access an idle channel at the same time, causing a collision. Both nodes select a random number between 0 and 1, and use it to calculate their backoff time. Consider that node A chooses a lower backoff time. Node A then starts its transmission and node B will not transmit, sensing the busy medium. If node A has more frames to transmit, another collision will happen. Once again node A will choose a random number between 0 and 1, but node B chooses its backoff between 0 and 3 (as it is its second attempt). Clearly node A has a higher probability of gaining access to the medium at this time. Actually as long as node A has new frames to transmit it will have a higher and increasing probability to get access to the medium. This problem is well known and normally referred to as the *channel capture effect*. The number of packets consecutively transmitted by the node capturing the channel can potentially be hundreds of packets or more [8].

In the mid-1980s, a new topology was introduced. The coaxial bus was replaced by twisted-pair cables, the widely know 10BASE-T, connected via a *hub* in a star topology. The hub is a physical device that repeats every incoming bit in a given interface to every other interface. Consequently this new hub-based star topology is still a broadcast LAN, which is half-duplex and also suffers from the channel capture effect.

Switched Ethernet was introduced in the early 1990s, and has become dominant in current installations [7]. In this version the nodes are still connected in a star topology, but in its center the hub was replaced with a *switch*. This new device is considerably more intelligent than a hub. Instead of simply reproducing the bits received in a given interface to all others, switches are capable of learning the MAC addresses from the nodes connected to it. Basically the role of a switch is to receive a link-layer frame, find the interface connected to its destination and forward it to the correct output interfaces. Switches also have the ability to temporally store frames, as the amount of received traffic to be forwarded to a given interface can be higher than the link capacity of that interface. If one received frame is destined to a link that is occupied, the switch stores this frame and forwards it as soon as the link is free again.

Modern switches and Ethernet adapters are full-duplex, i.e. a switch and a node can both send frames at the same time without causing a collision. Current switched Ethernet deployments are thus a collision-free environment where the CSMA/CD protocol is no longer necessary.

The transmission rate of Ethernet depends on the actual technology used. The original paper in which Ethernet was described reports an experiment running at 3 Mbps, while current technology offers transmission rates up to 10 Gbps. In this work we only consider two of the typically used technologies, the 10BASE-T and 100BASE-T, which refers to 10 Mbps and 100 Mbps twisted-pair copper wire, respectively.

2.2 802.11 Wireless LAN

The IEEE 802.11 Wireless LAN standard [1] specifies a family of *Wireless LANs* (WLANs), which are one of the most important access network technologies existent. In this work we only consider the 802.11a, 802.11b and 802.11g versions, which are the most commonly used today. When discussing common aspects of these versions, we refer to them simply as 802.11.

In this work we consider only 802.11 infrastructure-based networks. This architecture has the *Basic Service Set* (BSS) as its fundamental building block. A BSS is formed by a central base station, know as Access Point (AP) and one or more wireless stations. BSS may be connected to each other via a *distribution*

system to increase wireless coverage, forming an *Extended Service Set* (ESS). This distribution system is normally connected to other networks, through a logical element referred to as *portal* in the standard.

Although these technologies have major differences in the physical layer, e.g. the frequency range utilized and the maximum transmission rate they support, they all provide medium access in the same way: *CSMA with Collision Avoidance* (CSMA/CA). This means that before a transmission an 802.11 node always senses the medium, but unlike Ethernet, 802.11 does not implement collision detection. The first reason why this is not done is that collision detection would require the ability to simultaneously send and receive signals. Because the power of a transmitted signal is, in most of the cases, much higher than the power of a received signal, it would be too expensive to build a radio capable of detecting collisions. The second reason is that even if the radio was able to transmit and receive at the same time it would still not be able to detect all collisions due to the *hidden terminal problem* [7]. 802.11 connections are thus half-duplex by design.

The basic CSMA/CA functioning depicted in Figure 2.1 can be described as follows. If a node senses the medium idle for more than a *Distribute Inter-Frame Space* (DIFS) period it is allowed to transmit. If the medium is sensed to be busy, the node waits for the duration of a DIFS and enters the exponential backoff phase. In this phase the node chooses a random backoff time from a contention window, which is defined in terms of a reference *slot time* (the number of slots and their duration is technology dependent). When the medium is idle again, the node has to wait for a new DIFS period and starts its backoff timer. If the medium is still free when the backoff period is over, the node gets access to the medium and can start its transmission. However if the medium becomes busy before the node is allowed to transmit (i.e. another station has a shorter backoff time), the node has lost this cycle and has to again wait for the medium to be idle for a DIFS period before attempting to gain access to the medium again.

To provide some fairness, if a node does not get access to the medium in one cycle, it stops its backoff timer. After the medium is again idle for a DIFS period, it resumes the timer. As soon as the timer is over the node will get access to the medium. This means that deferred nodes have some advantage over stations that just start to contend for the medium, as they have to wait for only for the remainder of their backoff timer from previous cycles [9].

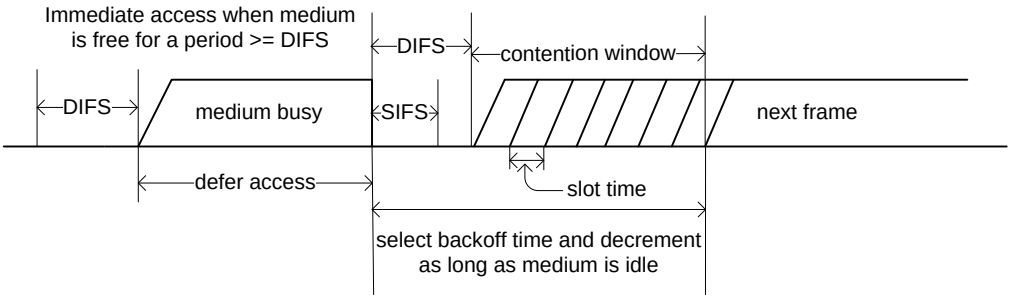


FIGURE 2.1: The basic medium access method (adapted from [1]).

It is important to note that this backoff procedure is also performed after a successful transmission. After a transmission the node selects a random backoff time to be utilized for the next transmission, even if there are no other nodes transmitting. This is done to avoid the channel capture effect present on Ethernet. As a side effect two frames transmitted back-to-back over a wireless hop using 802.11 are always be separated by a random time period.

The 802.11 MAC protocol also defines acknowledgement (ACK) frames. After correctly receiving an 802.11 frame the receiver accesses the medium after waiting for a *Short Inter-Frame Space* (SIFS) period. As the SIFS period is smaller than the DIFS, the receiver has priority over other nodes. The ACK frame is a confirmation that the previous frame was received correctly, which is important in error-prone environments such as wireless connections. If after a transmission an ACK frame is not received has to content form medium access, entering the exponential backoff phase described above. For each retransmission attempt the sender doubles its contention windown, as in the CSMA/CD protocol used in Ethernet.

The transmission rates for 802.11 protocols depend on the actual technology used. In the original standard the transmission rate is, at maximum, only 2Mbps using *Direct Sequence Spread Spectrum* (DSSS) with 11-chip Baker sequence and *Differential Quadrature Phase Shift Keying* (DQPSK) modulation.

The 802.11b standard, which has been added as an amendment to the original standard, describes a new physical layer that provides transmission rates up to 11Mbps by using 8-chip *Complementary Code Keying* (CCK) as the modulation scheme. This new capability is referred to as *High Rate DSSS* (HR/DSSS), and it is compatible with the original physical layer. The standard also states that all control frames may be exchanged at basic data rates (i.e. the ones defined in the original standard) to keep backwards compatibility.

Two packet formats are standardized for 802.11b; they are basically formed by a *Physical Layer Convergence Protocol* (PLCP) preamble, a PLCP header and the payload. Basically these headers provide means for the nodes to synchronize, perform energy detection (for carrier sensing), etc. and they also contain information such as transmission rate, length of payload and error checking. The first, and mandatory, format is called *long PLPC PPDU*, which is 192-bit long (PLCP preamble plus PLCP header) and is transmitted at 1Mbps. The *short PLPC PPDU*, the second format, has defines a smaller 72-bit PLCP preamble which is transmitted at 1Mbps and uses the same PLCP header, but it is transmitted at 2Mbps. For both formats, the payload can be transmitted up to 11Mbps.

802.11b operates in the 2.4 GHz ISM band which is divided into 14 channels. Depending on national regulations, a different number of channels is actually used. For instance, in the US and Canada 11 channels are used, while in Europe, with a few exceptions, 14 channels.

In 802.11a defines a new physical layer which offers up to 54Mbps using *Orthogonal Frequency-Division Multiplexing* (OFDM). To achieve this transmission rate, 216 data bits are coded into an OFDM symbol and transmitted using 64-QAM modulation. Due to the nature of OFDM, the packet format defined by 802.11a is quite different from the one defined in 802.11b. It can be divided in PLCP preamble, *signal* and *data*. The PLCP preamble is used for frequency acquisition, channel estimation and synchronization. It is 12 symbols long and it takes 16μs to be transmitted. The signal field contains information such as the data rate and modulation of the rest of the packet and length of payload. It is 1 symbol long and it is transmitted at 6Mbps using BPSK modulation. The data field contains information to synchronize the receiver, the upper layer payload and padding, which guarantees that the number of bytes of the frame maps to an integer number of OFDM symbols.

802.11a operates in the 5 GHz band, which depending on national regulation represent a different frequency range. For instance, in the US, the FCC authorized three domains for the US, 5.15-5.25 GHz, 5.25-5.35 GHz and 5.725-5.825 GHz, while in Europe the ETSI defined two frequency bands, 5.15-5.35GHz and 5.47-5.725 GHz. Depending on the actual band in use, different non-overlapping channels are available.

Finally, 802.11g also uses OFDM for modulation, achieving transmission rates up to 54 Mbps, using a physical layer very similar to the one defined in 802.11a, but it operates at the same 2.4 GHz band as 802.11b. Protection mechanisms are necessary to allow co-existence of 802.11b and 802.11g nodes in the same BSS as they define incompatible physical layers. Two are the defined protection mechanisms: RTS/CTS and CTS-to-self which basically consist in the exchange of extra frames to reserve the medium for a given amount of time. In the following we describe the CTS-to-self mechanism, which is present in some of the data sets used in this work.

The medium reservation is defined by means of the *Network Allocation Vector* (NAV), which is an indicator of time periods when a transmission should not be initiated, even if nodes sense the medium as idle. When the CTS-to-self mechanism is in use, before transmitting a frame at a non-basic data rate (like the ones defined in 802.11g) a node must distribute NAV information, to reserve the medium. For that the node transmits a CTS frame at basic data rate with its own MAC address (the *CTS-to-self*). This frame contains a duration value that protects the transmission of the pending frame and the ACK to be sent in response. As a result other nodes that receive the CTS-to-self frame, including the 802.11b nodes that are not able to understand ODFM modulation, refrain from transmitting during the NAV duration. Clearly this mechanism reduces the throughput offered to upper layer protocols.

Is important to note that the transmission rates reported in this section simply describe how fast the physical layer can transmit a frame once the medium access is obtained. When calculating the effective transmission rates that 802.11 protocols make available to higher layer protocols is important to consider the delays introduced by the CSMA/CA protocol, such as the slot time and the inter-frame spaces (SIFS and DIFS).

2.3 Acknowledgements in TCP

The Transmission Control Protocol (TCP) is, together with the Internet Protocol (IP), one of main protocols of today's Internet protocol suite. It provides a connection-oriented, ordered delivery and reliable transport service used by a number of traditional applications, such as the *Hypertext Transfer Protocol* (HTTP),

Simple Mail Transfer Protocol (SMTP), *File Transfer Protocol* (FTP) and *Secure Shell* (SSH). TCP is defined in a series of documents known as *Request For Comments* (RFC). [10] provides a "roadmap" to the RFC documents relating to the Internet's TCP.

TCP provides a logical end-to-end (abstracts the network connecting the hosts), point-to-point (always involves only two hosts) and full-duplex connection between processes running of different hosts. TCP ports are used to deliver the data to the right process, in a process called demultiplexing. A TCP connection can be uniquely identified by the IP addresses of the communicating hosts plus the TCP ports chosen for the connection in both ends.

It provides a reliable delivery service that is based on the use of sequence numbers, acknowledgments (ACKs) and timers. In TCP data is viewed as an ordered stream of data, and, to reflect this view, the sequence number used in a transmitted segment represents the byte-stream number of the first byte in the segment. The sequence number allows data to be delivered orderly in the destination, regardless of any disordering or packet loss that may occur during transmission.

Receivers confirm the correct reception using a *cumulative acknowledgment* scheme, where the receiver explicitly sends an acknowledgment informing that it received all data preceding the *acknowledgment number*. Consider the transmission from segments from a Host A to a Host B. In this scenario every incoming segment in Host B has a sequence number for the data flowing from A to B. The acknowledgment number that Host B puts in its segment is the sequence number of the next byte Host A is expecting from Host B [7]. The use of sequence numbers and acknowledgment numbers allows the correlation of data segments with their respective ACK segments.

When a host sends a segment over a TCP connection, it starts a timer, if it is currently not running, and passes the segment to the network layer for transmission. The value of this timer is based on the *Round Trip Time* RTT which is constantly estimated by TCP. In case the timer expires before an ACK for that segment is received, a retransmission is triggered. When receiving an ACK that is acknowledging one or more previously unacknowledged segments the timer is restarted.

While this covers the basis of the reliable transfer, TCP contains a series of improvements over this basic method, such as selective acknowledgments, fast retransmit and delayed acknowledgments. We are particularly interested in the last mechanism, which determines how ACKs should be generated. The main idea of this mechanism is to reduce the required bandwidth for as TCP connection by sending less than one ACK segment per data segment received, which is referred to as a "delayed ACK".

These are the requirements for the delayed ACK mechanism [11]:

> A TCP SHOULD implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds, and in a stream of full-sized segments there SHOULD be an ACK for at least every second segment.

It is important to note that by RFC documents conventions [12], the word *MUST* means that the definition is an absolute requirement of the specification while *SHOULD* means that there may exist valid reasons in particular circumstances to ignore a particular item, that is, it represents a recommendation rather than a requirement for compliance.

The main reasoning is that a delayed ACK gives the application an opportunity to process the received data and perhaps to send an immediate response. As the acknowledgement information can be *piggybacked* in a data segment, this avoids the transmission of one TCP segment.

2.4 Related Work

Network measurements have been used to study the performance and user behavior in wireless networks [13, 14]. These works provide valuable information about typical characteristics of wireless environments. [15] observes differences in TCP connections established by wired and wireless clients. Characteristics like delay, losses and termination of TCP connections are studied, but no classification scheme is proposed. In [16] is proposed that wireless and wired access networks can be differentiated based on the RTT of probe packets. The classification mechanism described in this work assumes high loss and low bandwidth on the wireless link. In

contrast, our method is based in differences on how the considered access networks provide medium access.

Packet inter-arrival times were originally used to solve problems related to capacity or available bandwidth estimation, using both active, where packets are injected on the network, and passive measurements, where the traffic is captured as it passes by a network device (e.g. sniffer). Pathrate [17] and CapProbe [18] are examples of the use of packet inter-arrival times in active measurements for this purpose, while in tools like Nettimer [19], multiq [20] and pprate [21] passive measurements are used.

In recent research work, packet inter-arrival times have been use for the identification of access networks, either using active measurements [22], or analysing passively captured traffic [4, 5]. These works exploit differences between 802.11 and Ethernet protocols transmission bandwidth and on how they provide medium access. By analyzing the interval between packets, they propose classification methods for access network identification.

In these works the use of two metrics are recurring: entropy and median. These metrics were chosen to reflect some of the basic characteristics of the protocols and the medium where they operate. First, 802.11 protocols operate in a shared half-duplex medium, where collision/contention is expected to happen. Also link-layer retransmission can occur depending on the conditions of the wireless medium. These conditions may vary depending on the distance from the client to the access point (AP) and on the level of interference present. It is also important to keep in mind that these protocols determine that if a station is to send two or more back-to-back packets, they should be separated by a random backoff even if no other station is transmitting. Ethernet connections on the other side normally operate on full-duplex dedicated link, as discussed in Section 2.1.

From this one should expect that the inter-arrival times of packets crossing a wireless hop are fundamentally more random when compared to the ones crossing an Ethernet link. The entropy metric is used to measure the uncertainty associated with the random variable that represents these inter-arrival times. It is necessary to discretize the inter-arrival times in order to calculate its entropy, and different bin sizes are proposed. The entropy of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ is defined as

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i), \tag{2.1}$$

where the function $p(x)$ denotes the probability mass function of X and b is the base of the logarithm used. When $b = 2$ the entropy is said to be expressed in bits.

The median is a type of average defined as the middle value of a distribution. At most half of the observations are lower than the median and at most half of the observations are higher than the median. It can be found by sorting all observations in ascending order, i.e. from the lowest to the highest value, and then selecting the middle one. In case there are two middle values, the mean of them represents the median.

The median of inter-arrival times of packets is a useful measure that can capture differences in the transmission rate of the protocols, and also on how they provide medium access. For instance, 802.11 protocols explicitly acknowledge data frames with a control frame, which leads to larger median values when compared to Ethernet, where these link-layer acknowledgements are not used. Median is preferred to mean as it is more robust against the presence of outliers.

The work proposed in [22] is based on transmission of packet pairs (a packet pair contains two back-to-back packets) from a sender to a receiver. The receiver then classifies the sender using entropy and median of the inter-arrival times of packet pairs. Three classes of endpoints are defined: Ethernet (high-bandwidth wired), 802.11 (wireless LAN) and ADSL/Cable/Dial up (low-bandwidth wired). To distinguish the access networks the classification scheme uses *fixed* thresholds for entropy and median, which are calculated using in a simplified analytical model. The need for cooperation between sender and receiver is the major shortcoming of this approach and why it does not apply to our scenario.

In the first passive approach [5] classification is done using so called ACK-pairs, two TCP ACKs generated in response to data segments that arrive close in time at the measurement point. The idea behind this approach is that the time interval between the ACKs in a pair (*inter-ACK time*) can be used to determine whether the TCP flow crosses an 802.11 hop or not. An analytical model similar to the one used in [22] is used to study the effects that the access link have on the inter-ACK

time. As a conclusion of this study they state that deterministic classification of 802.11 and Ethernet flows based on *fixed* thresholds for the median inter-ACK time would not provide accurate results.

Motivated by the results taken from this analytical model, they propose an probabilistic method to classify the flows according to their access network type. For each TCP flow the median inter-ACK time is fed into an iterative Bayesian inference algorithm which classifies the flow. Although the algorithm is claimed to have a low inference error, downsides of this method are: (1) it classifies flows, not endpoints; (2) it does not use the network traces efficiently, as TCP flows tend to contain a low number of ACK-pairs; and (3) it requires a training set of TCP flows from which 802.11 and Ethernet observation distributions can be obtained.

Rather than describing a method to differentiate wireless and wired endpoints which based in the median of inter-ACK time intervals as in [5], we analyze the effects of the access network technologies on the inter-ACK time distribution. We describe the general behavior of this distribution for different scenarios, and show how it can be used for the classification of access network types.

The second passive approach [4] is also based on the interval between two consecutive segments in the monitoring point, although this method does not restrict the evaluation to ACK-pairs and uses both TCP and UDP segments. The only restriction is that two consecutive segments in a 5-tuple (IP source/destination addresses, transport source/destination ports and transport protocol) are less than 10ms apart. The classification algorithm is based on the entropy of inter-arrival time of segments in a pair. For the two data sets used in the evaluation, the authors claim to have high classification accuracy when more than 200 intervals are used for classification, although in our tests this approach did not perform well. Our attempt to reproduce the results presented in this work is discussed in Chapter 3.

The ACK-pair technique is also used on [6] with the goal of identifying endpoints connected to rogue (unauthorized) access points. The downsides previously mentioned were addressed to some extent by using sequential hypothesis test [23], with and without training data. However they report that the accuracy of the method without the use of a training set is considerably lower than the one with

it and in addition to that, the method without training is only capable of reporting 802.11 endpoints, in contrast with the sequential hypothesis test with training reports both WLAN and Ethernet endpoints.

2.5 Data sets

In this work we use four sets of tcpdump/libpcap [24] traces: two traces are from publicly available data sets, traces generated in a semi-controlled laboratory environment and traces collected in our campus university. The first of these data sets is the *CRAWDAD data set Dartmouth/Campus* [2]. It consists of packet headers for wireless communication captured in some buildings of their university campus. In total, twenty two 802.11b access points were monitored. The monitoring point was connected to the same switches used by the access points. Through *port mirroring*, all traffic on the ports connected to access points is copied to the port connected to the monitoring point.

The second data set is the *Simpleweb / University of Twente - Traffic Measurement Data Repository* [3]. More specifically we use traces from *Location 6*. The description of this data set states that it has been made on the 100 Mbps Ethernet link that connects an educational organization to the Internet. Moreover all workstations on the location, 100 approximately, have a 100 Mbps LAN connection.

The semi-controlled laboratory environment built for our tests is represented in Figure 2.2. On a server installed on an arbitrary location in our campus network we run the *chargen* service [25] as a traffic generator. When a client opens a TCP connection to this service, *chargen* generates an arbitrary sequence of characters until the connection is closed. We perform tests from different clients, varying hardware and operating system, connected to both Ethernet and 802.11g networks. On the server, our monitoring point, we record all *chargen* connections for subsequent analysis. We refer to this laboratory environment as semi-controlled because we can determine the time and duration of the *chargen* connections, but we have no control on parameters like the quality of the wireless link and the amount of cross-traffic on the path, and these parameters tend to change throughout the day.

The last data set used consists in packet headers collected in our university campus in August of 2007. The capture was performed in the 1Gbps link that

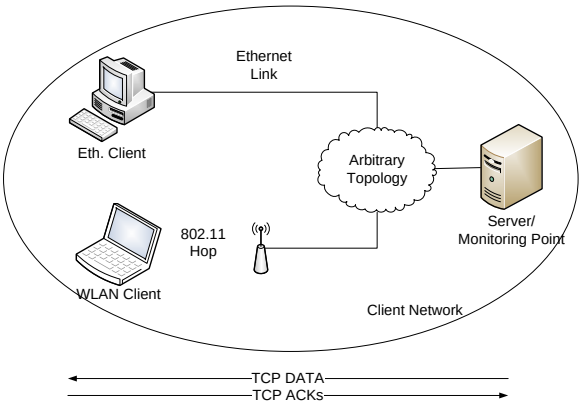


FIGURE 2.2: The laboratory setup.

connects part of the student houses to a router which is connected to the university's internet service provider. The IP addressing scheme allows us to distinguish 4 types of hosts by their connection type: Ethernet, 802.11 WLAN, Virtual Private Network (VPN) and ADSL. However we do not have information on the actual access network technology used. For instance it is possible that the Ethernet connections consist in a mixture of 10 Mbps, 100Mbps and 1Gbps hosts. Both 802.11a and 802.11b/g APs are present in our network, so it is possible to have hosts with 802.11a, 802.11b (using CTS-to-self protection) and 802.11g network cards. Finally, this data set presents a considerable amount of packet loss.

From this point on we simply refer to these data sets respectively as *CRAWDAD*, *Simpleweb Location 6*, *Laboratory* and *University of Twente (UT)* traces.

Chapter 3

Intel/Torino Reproduction

For the goal of identifying the type of access network we considered to reproduce some of the results presented in the related work. The simplicity of the classification method described on [4] is the main reason behind this reproduction attempt. From this point on we refer to this method as *Torino/Intel*.

This Chapter is organized as follows. In Section 3.1 we present the proposed classification algorithm in details. In Section 3.2 we analyze our results and discuss reasons why this classification method might not present high accuracy.

3.1 Classification Algorithm

The first step in this classification method is to separate the network trace based on IP address source. For each source, a second separation is made based on 5-tuple[1] flows. The interval between two consecutive packets in a flow is then computed and only the ones smaller than (on [4] nomenclature) $T_{RTT} = 10ms$ are kept. Two values are then calculated, (1) H_{IP}, the empirical entropy using $b = 2$ (see Eq. 2.1) calculated on the whole IP-source aggregated traces, and (2) $H_{IP,5}$, the empirical entropy of the largest 5-tuple flow (in terms of number of inter-arrivals) for each IP-source trace. The variation of entropy is defined as $\Delta H = H_{IP} - H_{IP,5}$.

Algorithm 1 shows how the classification is performed (the authors use the term *detection* in contrast *classification* used in this work). The thresholds proposed

[1] 5-tuple consists in IP source and destination address, transport protocol (TCP or UDP) and transport source and destination ports

on the work and the ones used on our tests are the same: $H_{lower} = 3.5bits$, $H_{upper} = 5bits$, and $\Delta H_{THR} = 0.5$. Their values are based on Figure 3.1, which shows the Probability Mass Function (PMF) of entropy computed over a training dataset (no information on the contents of this training set is given). As can be observed, the majority of wireless flows is where $H_{IP} >= H_{lower}$ while the wired flows are concentrated where $H_{IP} <= H_{lower}$. It is worth noting that some wired flows have $H_{IP} >= H_{lower}$ and also some wireless flows have $H_{IP} <= H_{lower}$ so this method cannot be expected to have 100% accuracy.

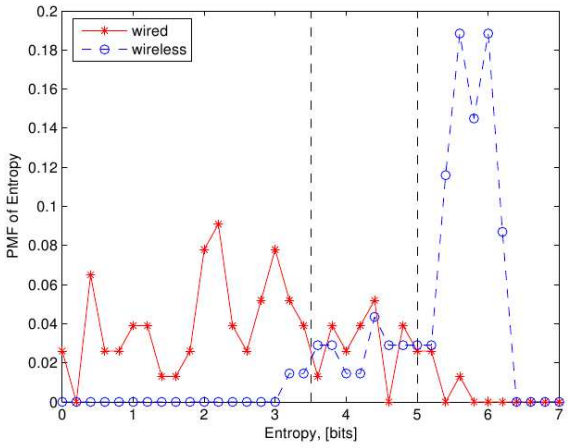


FIGURE 3.1: Probability Mass Function of entropy (copied from [4]).

In the interval in the range H_{lower}, H_{upper} bits the two distributions are superimposed. For flows in this region the variation of entropy is used as discriminator. It is argued that for wireless hosts, the uncertainty measured by $H_{IP,5}$ already accounts for the effects introduced by wireless transmission. As a consequence, adding other smaller 5-tuple flows has a marginal impact on the value of the aggregated entropy resulting in a low ΔH. In the case of wire hosts, instead, the variation of entropy is driven by different factors. When ΔH is measuring the

Algorithm 1: Classification Algorithm

if $H_{IP} <= H_{lower}$ **then**
  The host is wired
else if $H_{IP} >= H_{upper}$ **then**
  The host is wireless
else if $H_{lower} < H_{IP} < H_{upper}$ **then**
  if $\Delta H >= \Delta H_{THR}$ **then**
    The host is wired
  else if $\Delta H <= \Delta H_{THR}$ **then**
    The host is wireless

impact of aggregating different flows. By adding more outcomes the distribution defined over a limited interval becomes more informative, i.e. the aggregated entropy grows [4].

3.2 Results Discussion

To test the algorithm we used *CRAWDAD* and *Simpleweb Location 6*. Traces from the *CRAWDAD* data set are also used on [2]. Figure 3.2 shows the outcome of our implementation of the Torino/Intel algorithm for a trace from *CRAWDAD* data set. Only the endpoints containing 200 or more pairs of packets are showed, situation in which the algorithm is argued to present better results. The method classifies correctly 41 out of 49 endpoints (remember that on this data set all traffic is generated by wireless endpoints).

In a more detailed analysis of the traces we study what could have been the cause of the misclassification of some endpoints. The traces for the five endpoints that have empirical entropy lower than 3.5 present a large number of packets with the same timestamp. This is probably due some inaccuracy in the capture procedure and causes low empirical entropy value. For two consecutive packets to have the same timestamp, the interval between them should be smaller than $1\mu s$ (timestamp precision in the libpcap format) which, given the packets size and link bandwidth, should not be feasible. The other three misclassified points have empirical entropy near the threshold $H_{upper} = 5$, so this error can be explained by the method inaccuracy as explained before.

The results for Simpleweb Location 6 are shown on Figure 3.3. Again, only endpoints with 200 or more packet pairs are considered. Here only 95 out of 154 endpoints were correctly classified (remember that on this data set all traffic is generated by wired endpoints). By analyzing the traces in more depth we see that some flows present a large variation on the time between packets, which causes the entropy value to be large. This suggests that simply classifying endpoints based the empirical entropy of the interval between consecutive packets is ineffective as both wireless and wired traces can present high empirical entropy.

The entropy of packet inter-arrival can be a useful metric for the classification of access networks, but we believe this method fails to identify a timing behavior in transport layer connections that could be distorted or randomized when crossing

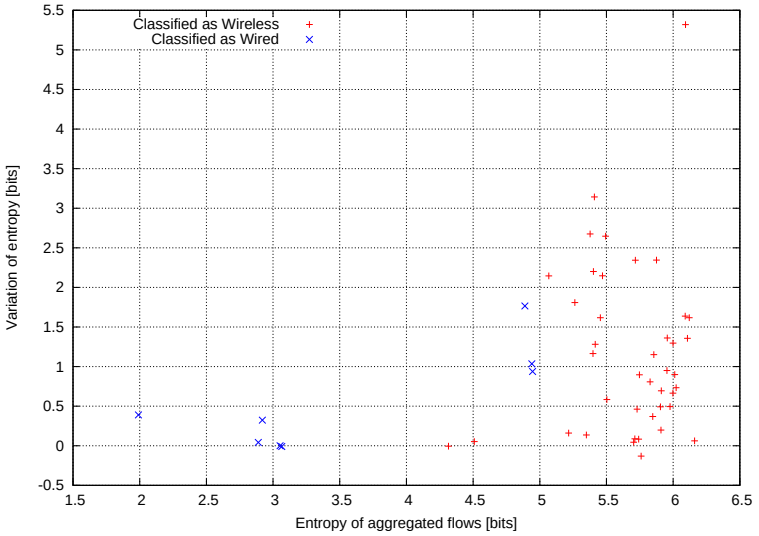


FIGURE 3.2: Result of Torino/Intel algorithm for *CRAWDAD* traces.

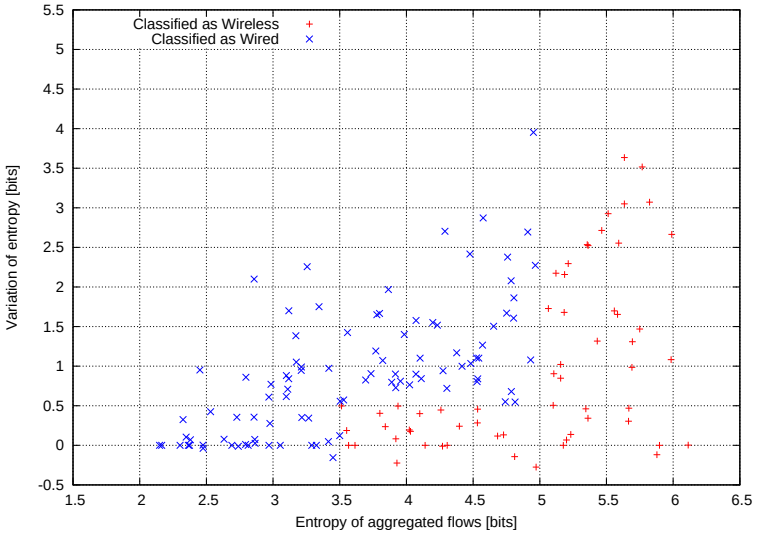


FIGURE 3.3: Result of Torino/Intel algorithm on a Simpleweb Location 6 traces.

a wireless link. Simply considering consecutive packets that are less than $10ms$ apart is not a good method to find packets that are sent back-to-back over a TCP or UDP connection as suggested in this work. For instance, when considering two 1500-byte packets (typical MTU size) sent back-to-back over a 10Mbps link, the interval between the packets is expected to be $1.2304ms$, less than 15% of the considered value. In faster links, this interval would be even smaller.

Also no assumption is made about the environment where the traces are made, so the inter-packet time could be seen as random (or with *high* entropy) for a number of reasons. For instance, an application could generate the packets in this fashion. In [20] it is shown that TCP connections facing queues in a congested

link can cause the probability density function (PDF) of the inter-packet time to have multiple peeks separated by *equally-spaced mode gaps*. This can be seen as packets that were sent back-to-back facing different amounts of cross traffic in this congested link. This is another behavior that makes the inter-packet time of a wired Ethernet connection to have high entropy and thus be wrongly classified as wireless by this method.

Chapter 4

ACK Inter-arrivals Study

As good results are not achieved in our reproduction of the classification method proposed in [4], we change our focus to the techniques proposed in [5, 6]. These works show that the inter-arrival times of consecutive ACKs on the monitoring point can differ significantly depending on whether the connection crosses a wireless hop or a wired Ethernet link. Differently from Chapter 3, we do not simply attempt to reproduce results of these works. We study the effect that the 802.11 and Ethernet protocols have on the inter-ACK time distribution searching for any pattern that can be useful for differentiating Ethernet from 802.11 connections.

For the series of experiments described in this chapter we consider TCP connections where a client mainly downloads data from a server and sends pure ACKs[1] in reply. For every pair of consecutive ACKs on the monitoring point we record the time between the ACKs (inter-ACK time) and the time between the respective data segments (inter-data time). The only restriction on the detection of such ACK-pairs is that we discard cases where segments are retransmitted or reordered.

In this chapter we perform a series of studies that give us some insight on how the behavior of such inter-arrival times. This is the basis for the description of the distribution of inter-ACK times for both wired and wireless connections that is discussed in Chapter 5.

[1] Pure ACKs contains no user-level data, i.e. only link, IP and TCP layer headers including possible options.

4.1 Inter-ACK Time vs. Inter-data Time

In this first study we try to find some relation between the inter-ACK time and the inter-data time. Here we analyze how the intervals between consecutive ACKs behave in different timescales. We plot a series of graphs of the inter-ACK time versus the inter data-time to gain some insight on data sets. In the first group of tests we extract ACK-pairs from randomly selected TCP flows existent on traces from *CRAWDAD* and *Simpleweb Location 6*. Later we perform the same study on traces collected on our laboratory environment.

4.1.1 CRAWDAD and Simpleweb tests

In Figure 4.1 we show the result for these tests. The inter-ACK and inter-data intervals of ACK-pairs present on the selected wireless *CRAWDAD* flows are represented by red crosses and the Ethernet ones in blue asterisks. No clear distinction between Ethernet and 802.11 connections can be made at this timescale. It can be observed that in cases where one of the intervals is large the other also is, i.e. if inter-ACK is larger than $200ms$ also is the inter-data time, and vice-versa. Also most of the intervals are concentrated near the origin, which is expected since in all considered protocols the time to transmit a full-sized TCP data segment (1500 bytes) is less than $2ms$.

Figure 4.2 shows the same graph in a much smaller time scale, namely from 0 to $10ms$ in both axis. Two support lines are included: $x = 500\mu s$ and $y = 1.2ms$. $500\mu s$ is roughly the minimum time to transmit one ACK in an 802.11b network, so we would expect not to have any inter-ACK time from a *CRAWDAD* connection left of this line. We believe that such small inter-ACK times are caused by limitations on the accuracy of the timestamp in this data set. $1.2ms$ is roughly the minimum time to transmit a full-sized TCP data segment (1500 bytes) over a 10Mbps link. As all Ethernet points below this line are less than 1500 bytes long, this is an indication that the traffic is crossing a 10 Mbps link. Because of the TCP delayed ACK mechanism, clients should send an ACK for every second data segments received, so many of the inter-data time represent the transmission time of two data packets. The concentration of inter-data intervals approximately at $2.4ms$, roughly the time to transmit two data packets over a 10Mbps link, is another indication that the traffic would be crossing a link with this capacity.

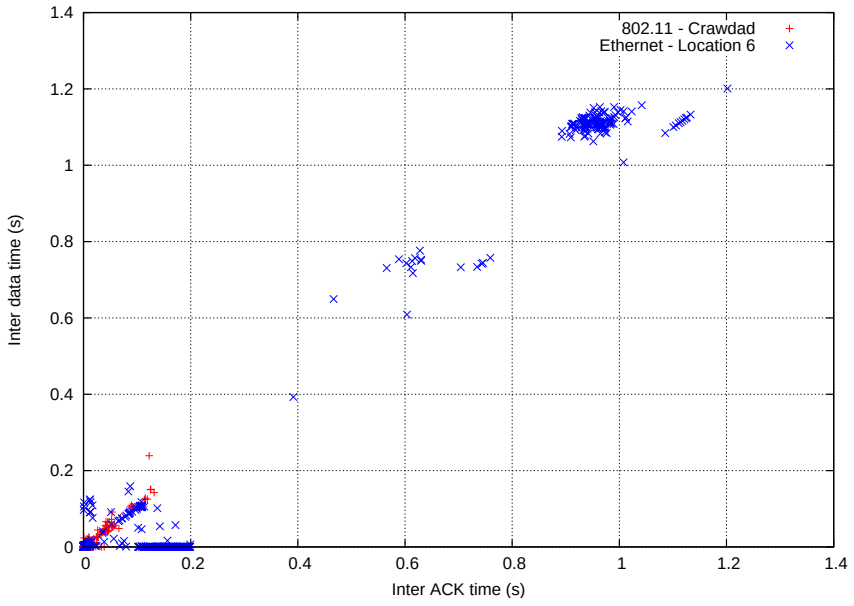


FIGURE 4.1: Inter-ACK Time vs Inter-data Time for *CRAWDAD* and *Simpleweb Location 6* traces.

Another interesting behavior in this graph is that the inter-ACK times are distributed in vertical lines spaced $125\mu s$ from each other. We have no plausible explanation for this fact. Our last observation is that a large number inter-ACK intervals for Ethernet connections tend to concentrate near the origin, while for 802.11 they are more spread, not being concentrated at any region.

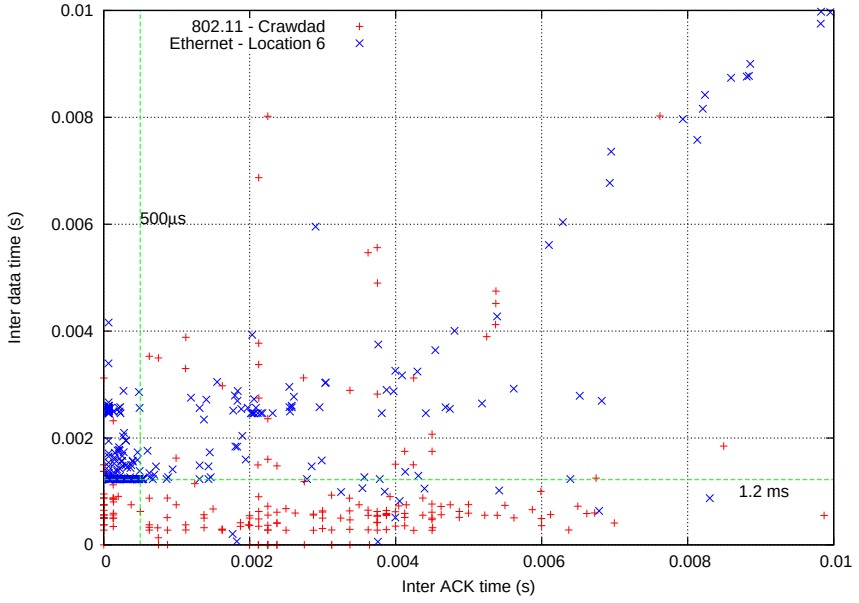


FIGURE 4.2: Inter-ACK Time vs Inter-data Time for *CRAWDAD* and *Simpleweb Location 6* traces zoomed on the origin.

4.1.2 Laboratory tests

We also perform this study on data generated in our laboratory. For this preliminary work we use a Mac OS client connecting to both 802.11 and Ethernet access networks. A series of traces of *chargen* flows are recorded, and we choose a representative flow for each access network type. The following graphs are plotted using these flows, unless noted otherwise.

As on the tests on the previous data sets we observe some relevant behaviors on small timescales. Due to the amount of ACK-pairs found in these traces we divide the results of 802.11 and Ethernet in two different graphs. On Figure 4.3 are present the results from the 802.11 trace. The most striking behavior is the scarce number of ACK-pairs in the interval $[0.0005, 0.001]s$ represented by the green support lines, in comparison with other regions of the graph. Another point to be noted is that even for small inter-data interval times (lower than $500\mu s$), the inter-ACK interval is rather spread over the plotted interval. On the Ethernet connection, plotted on Figure 4.4, it is clear that most of the data points are concentrated near the origin, having both inter-data and inter-ACK times smaller than $500\mu s$.

By observing the graphs more carefully, in a even smaller timescale, the points plotted seems to be concentrated in lines spaced $125\mu s$ from each other. As the pattern is observed on both axis, opposed to only the x axis (inter-ACK time) in *CRAWDAD* traces, we believe that this value represents the accuracy of our measurement method. This is clearly illustrated in Figure 4.5, which shows the inter-ACK and inter-data times for a selected Ethernet connection. This behavior is recurrent in all measurements done in our laboratory environment.

By the results of this study we identified some behaviors that might be used for the differentiation between Ethernet and 802.11 protocols. First, Ethernet inter-ACK times tend to be concentrated in low timescales, while for 802.11 the inter-ACK times are more spread. This is expected given the differences in transmission rate and on the use of the back-off mechanism in 802.11 protocols. Second, for the laboratory traces, we have a clear absence of ACK-pairs with inter-ACK time on the $[0.0005, 0.001]s$ interval. This behavior is not clear on the *CRAWDAD* traces, but it is possible that it is simply not visible because not enough data points are present. Our next step is to understand if this gap is caused by an

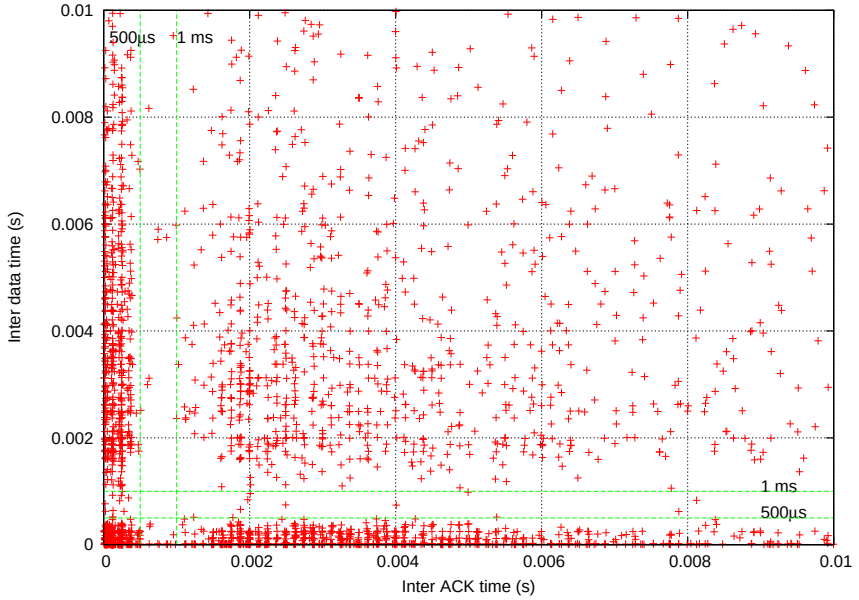


FIGURE 4.3: Inter-ACK Time vs Inter-data Time for a WLAN Laboratory trace.

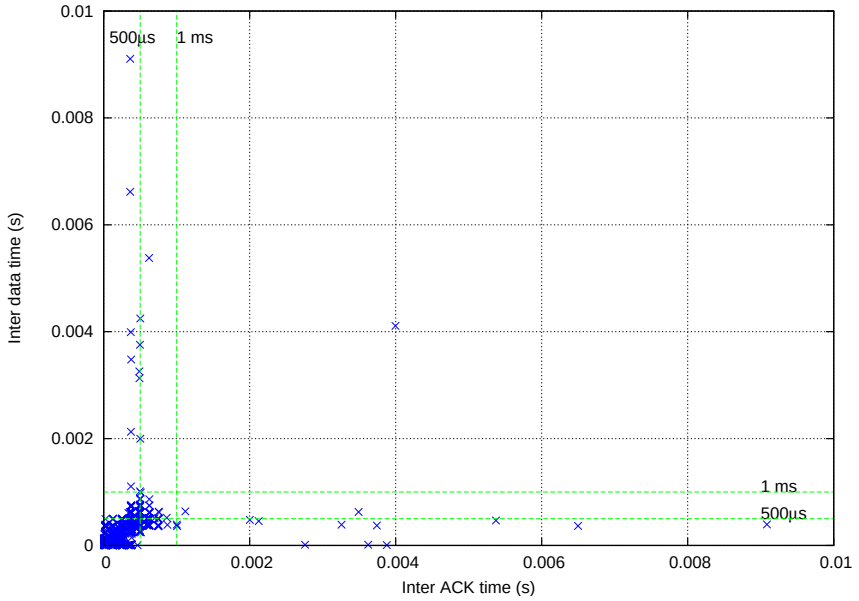


FIGURE 4.4: Inter-ACK Time vs Inter-data Time for an Ethernet Laboratory trace.

intrinsic characteristic of 802.11 protocols, and as such, can be used for access point network identification.

4.2 Inter-ACK Time observations

The objective of the set of experiments reported in this section is to study how packets from TCP connections are transmitted over a wireless hop, and try to identify there the reason behind the gap in the graph represented in Figure 4.3. For that we repeat our experiments on the laboratory setup, but this time, we use a third machine to monitor the connection on the wireless hop.

Note that to study the wireless hop would not be sufficient to perform the capture at the client-side, as packets sent by the client are copied to the trace before they are delivered to the network card. For this reason their timestamps would not reflect the time they are actually transmitted over the air. For the same reason it is not possible to observe link-layer retransmissions performing the capture at the client-side.

By performing the capture on the wireless hop with a third machine we have a better precision in timestamps, as their calculation takes into account the time in which the frame was actually transmitted over the air (and thus received by

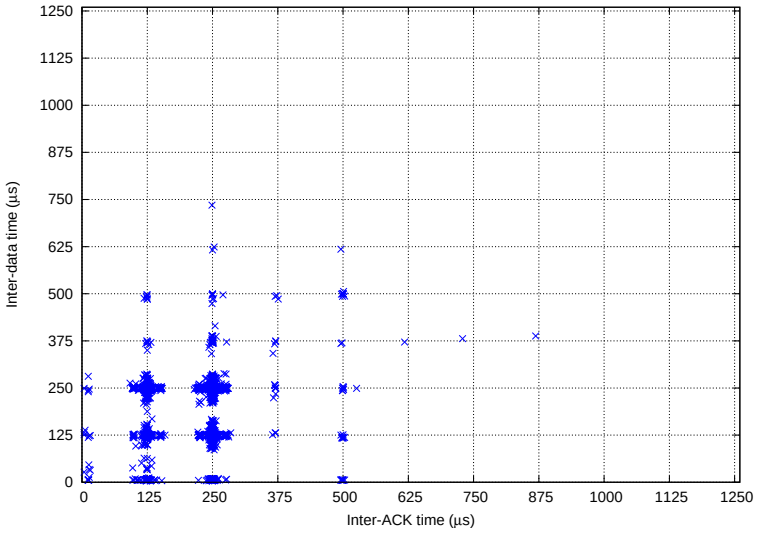


FIGURE 4.5: Data points concentrated in multiple of $125\mu s$ indicating the precision of the measurements.

this machine). Furthermore, this setup also makes possible to observe link-layer retransmissions.

4.2.1 Capturing on the air

Performing a network capture in a wireless link is tricky. It is necessary to perform a few steps to guarantee that the right information is being captured. First, in order to capture management/control frames, e.g. RTS, CTS, ACK, etc., it is necessary to select the right link-layer header type, as 802.11 adapters often transform 802.11 data packets into "fake" Ethernet packets before supplying them to the host, and, even if they don't, the drivers for the adapters often do so before supplying the packets to the operating system's networking stack and packet capture mechanism [26]. It might also be necessary, as it is in our case, to set the network card to the *monitor mode*, which allows it to capture traffic without associating to an access point. Note that this concept is different from *promiscuous mode*, which should also be used. Setting the network card to *promiscuous mode* determines that all packets received by the card should be delivered to the host, and not only the ones addressed to it.

Also it is important to select the right channel before starting the capture. Some monitoring tools like kismet [27] can operate in *channel hop mode*, capturing traffic from each of the channels for a small period, allowing the discovery of some information about all wireless networks in range. For instance it is possible to obtain the MAC address of an AP and its clients. In our tests we use kismet to find the channel used by our client in the *chargen* connection. Note that depending on the link quality, the client associates itself to another access point, which possibly operates in another channel, making necessary to check this information periodically when tests are being made.

Last, it is common that all wireless traffic is encrypted, making all information but the link-layer header to be unreadable after capture. One could still try to infer the content of the packets based on the MAC addresses, size and timestamp of the packets, comparing with information from unencrypted packets captured in another point of the network, e.g. on the client. However this is not a trivial task to do, possible problems are: the client could have been transmitting packets from another connection, link-layer retransmissions, the order of the packets in a connection might be different when observed from different points of the network,

etc. We were able to completely avoid this problem by setting an unencrypted connection to the access point.

4.2.2 Analysis of wireless link traces

After properly setting up our environment we start our tests. We establish a *chargen* connection with a wireless client, monitoring the connection both on the wireless hop and on the server. Using the trace generated by the server we identify the ACK-pairs. We them perform an in-depth analysis on the transmission of first 40 ACK-pairs on the wireless hop. On Figure 4.6 we plot the inter-ACK time for these pairs using the frame number of the first packet in the ACK-pair as an identifier. As before we plot two support lines to represent the $[0.0005, 0.001]s$ region. Note that only one pair, the one with identifier 264, is in this region.

We divide the ACK-pairs in three groups, depending on how they were transmitted on the wireless hop. The first group is formed by the ACK-pairs that were sent back-to-back packets, and it contains 10 pairs. All packets that are below the $0.5ms$ line are in this group. The pair 264, which has inter-ACK time of $0.504ms$, is also in this group. The second group is formed by the pairs that had at least one TCP data segment from the same connection transmitted in between the first and the second ACKs of the pair. This group has 14 pairs and their inter-ACK time varies from $1.615ms$ to $3.874ms$. The last group is formed by the 16 remaining pairs which faced some cross-traffic between the ACKs in a pair. For this group the inter-ACK times vary from $2.841ms$ to $33.495ms$. This division motivates the gap seen on the inter-ACK interval for wireless transmission, the first group is in the left side on the gap and the other two groups are on the right side.

The measurement in the wireless link also allows us to obtain detailed information on the quality of the wireless hop at the moment of the capture. We can observe that our network uses the CTS-to-self protection mechanism, which causes all stations that to want to transmit to reserve the medium with a CTS frame. Furthermore we see that all control frames are exchanged at 1Mbps, while the data frames from the monitored *chargen* connection are exchanged at 54Mbps. Under this conditions, including the CTS, data and ACK transmission times and the interframe times, it takes roughly $442\mu s$ to transmit a pure ACK and $658\mu s$ a full-sized TCP data segment. Details on the transmission time calculation are given on Section 5.1. These values explain the division seen on our measurements.

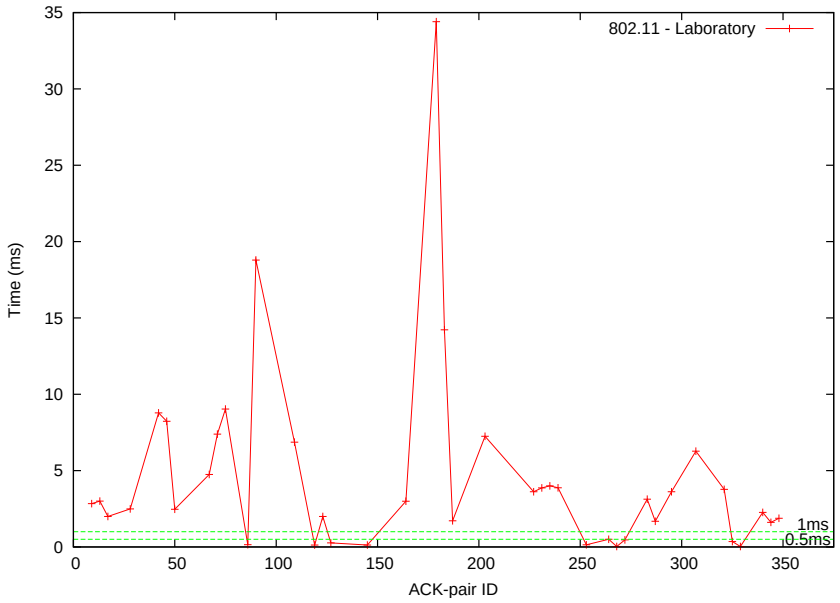


FIGURE 4.6: The inter-ACK time for the first 40 ACK-pairs.

For the group of back-to-back ACK-pairs, the inter-ACK time is approximately the transmission time of one ACK, and thus smaller than $0.5ms$, while for the second group the inter-ACK time includes at least the transmission time of an ACK plus and TCP data segment, and thus bigger than $1ms$.

4.2.3 $500\mu s$-bin Histograms

For a better visualization of this *peek-gap-peek* behavior we plot in Figure 4.7 a histogram for the wireless connection as follows. On the horizontal axis, time is divided in bins of $500\mu s$. The vertical axis represents the fraction of ACK-pairs with inter-ACK time in each of these bins. As discussed in Section 4.1.2, the measurements on the laboratory traces tend to concentrate in clusters separated by $125\mu s$ intervals (see Figure 4.5). We want that all points in these clusters contribute for the same bin on the histogram, as they likely represent the same measure. To accomplish this we define the beginning of the histogram to be -62.5 instead of zero, i.e. the first bin covers all inter-ACK times on the interval $[-62.5, 437.5)\mu s$, the second covers the interval $[437.5, 937.5)\mu s$, and so on. The gap can be clearly observed in this representation.

In Figure 4.8 the same type of histogram is plotted to an Ethernet connection. As observed before, inter-ACK times are concentrated on small timescales. When comparing to the histogram both histograms the differences are obvious.

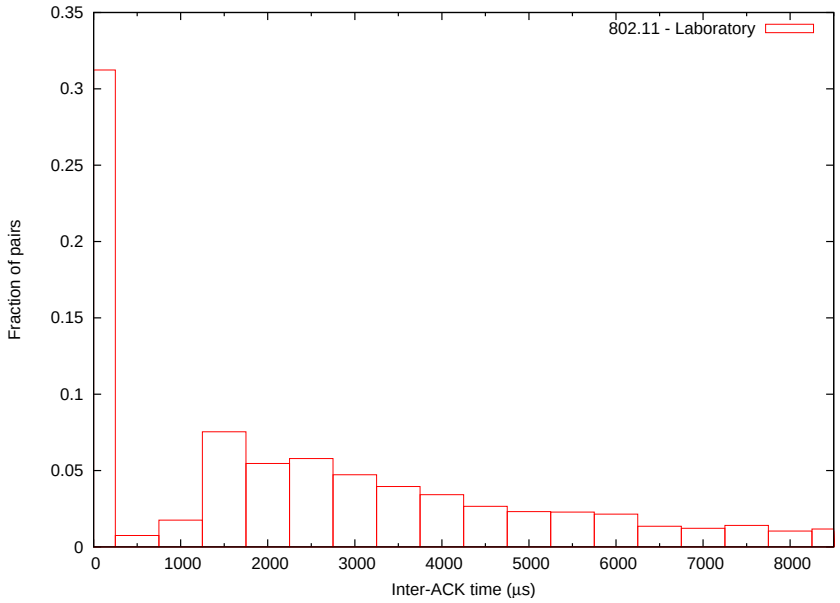


FIGURE 4.7: ACK-pair distribution histogram for a Mac OS 802.11 connection.

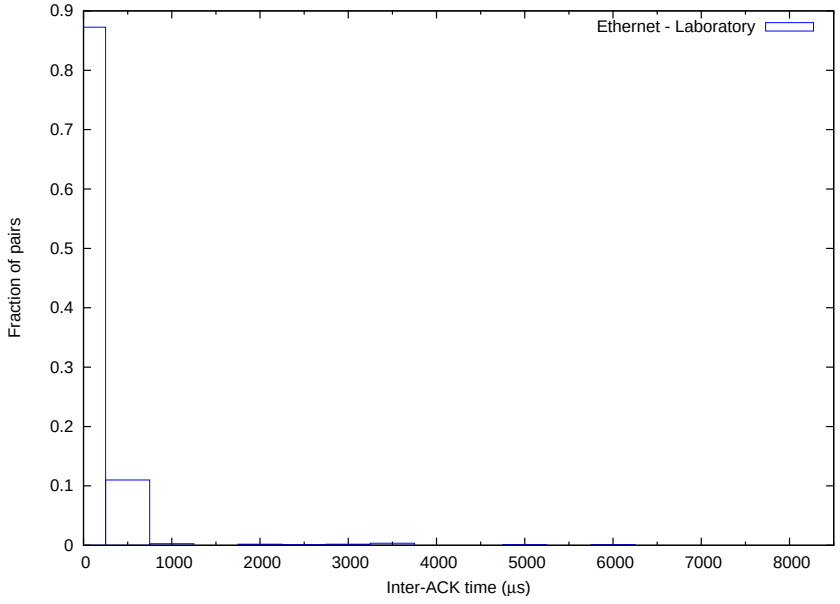


FIGURE 4.8: ACK-pair distribution histogram for a Mac OS Ethernet connection.

We then try to verify the generality of the *peek-gap-peek* behavior using a different wireless client, running the Windows XP operating system. The results are shown on Figure 4.9. No gap is visible on this graph, but this is due to the position of the bins the histogram. In the Mac OS test, the inter-ACK times for pairs transmitted back-to-back contribute to the first bin in the histogram, i.e. they are smaller than $437.5\mu s$. For this test, the inter-ACK times for back-to-back pairs are concentrated at $500\mu s$, thus contributing to the second bin on the histogram. When ACK-pairs have at least one data packet transmitted in between, the inter-ACK time increases to approximately $1100\mu s$ ($442\mu s + 658\mu s$), contributing for the third bin.

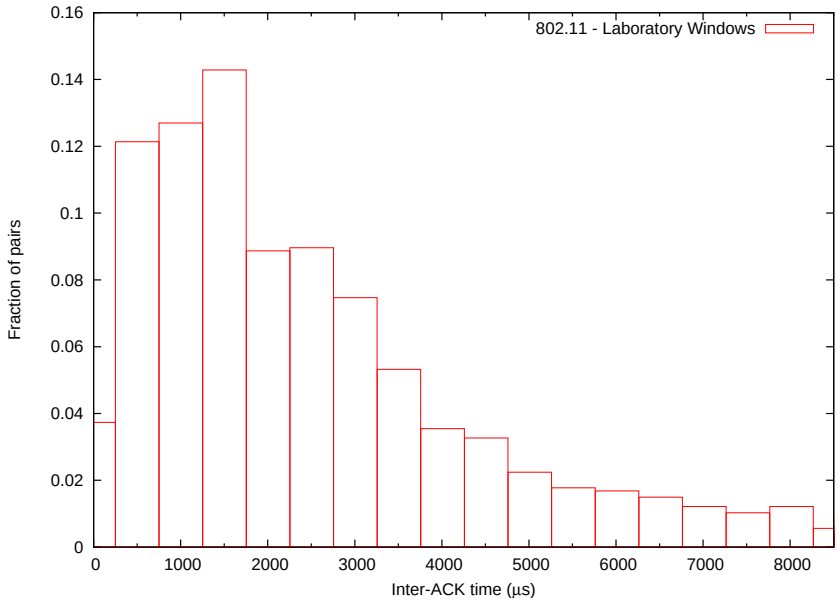


FIGURE 4.9: ACK-pair distribution histogram for a Windows 802.11 connection.

To show that the *peek-gap-peek* is still present on this trace, we plot the graph in a slightly different way. We reduce the size of the first bin to $125\mu s$, keeping the size of every other bin in $500\mu s$. The result is shown on Figure 4.10. In this graph, the back-to-back pairs still concentrate on the second bin, which now represents the interval $[62.5, 562.5)\mu s$. But pairs that have at least one data packet in between are now contributing to the forth bin, representing the interval $[1062.5, 1625)\mu s$. It is possible to observe that the gap in the inter-ACK time between back-to-back pairs and pairs with at least on data segment in between still exists, but in a different position. We come back to this issue on Chapter 6, where we discuss the observations made in the laboratory experiments in more details.

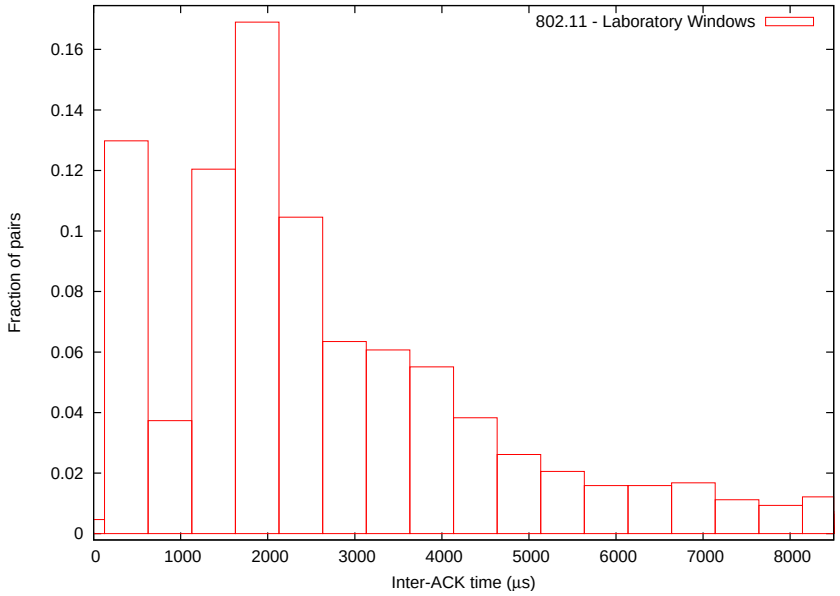


FIGURE 4.10: ACK-pair distribution histogram for a Windows 802.11 connection.

This study showed us the impact that the half-duplex nature of the wireless medium had in the inter-ACK time as seen from the server. As ACKs flowing from the client to the server compete for the medium with data segments flowing in the opposite direction we expect to see a gap on the distribution of inter-ACK times distinguishing the cases where ACKs are transmitted back-to-back or not.

Chapter 5

The Inter-ACK Time Distribution

The results of the ACK inter-arrivals study show that it is possible to observe characteristics from wireless connections on the passively captured traffic. In this chapter we describe in more details what characteristics of the access networks impact on the distribution of inter-ACK time, and it should be distributed from a theoretical point of view.

Throughout this chapter we assume the scenario described in Section 1.1. This scenario consists of a local network with both wired and wireless clients connected through a arbitrary topology. This network connects to external servers on the Internet. A monitoring point is deployed in a way in which it is capable of capturing traffic exchanged between local and external endpoints in both directions. We monitor TCP connections in which data flows downstream and acknowledgements upstream, with respect to the client. We assume that the access link of the clients is the communication bottleneck, i.e. these are the links with smaller transmission capacity, and the monitoring point is located near to them.

This chapter is organized as follows. In the first two sections we discuss factors that have large impact on the distribution of the inter-ACK times: the transmission capacities of links and the duplex capabilities of the medium. Based on this information we describe the theoretical inter-ACK distribution in Section 5.3. We close this chapter with some practical issues on the identification of ACK-pairs in Section 5.4.

5.1 Link Transmission Capacities

As discussed in Chapter 2 Ethernet and 802.11 protocols have different transmission rate capacities. We consider Ethernet links to be of 10Mbps or 100Mbps, and 802.11 protocols to range from 1 to 54Mbps. Clearly when observing the interval between consecutive ACKs the transmission capacity of the studied link will have a large impact on the distribution of such intervals. Here we make estimates of how long it takes to transmit the TCP data and ACKs segments over different link types. This analysis is based on [28].

For our analysis we assume that all TCP data segments are 1500 bytes long, which is a typical Maximum Transmission Unit (MTU) for the studied protocols. We also assume that ACKs have no user data, i.e. they consist of 40 bytes (20 bytes from IP header and 20 bytes from TCP header without options) plus link-layer encapsulation. The use of TCP options, such as *timestamp*, would not have a big impact on this analysis, as for TCP data segments the TCP options are included on the 1500 bytes MTU and for ACKs they would only add a few bytes of overhead, and thus a few microseconds on their transmission time.

5.1.1 Ethernet

We start our analysis with the simpler case, the Ethernet. This protocol adds 38 bytes per packet of overhead, consisting of inter frame gap (12 bytes), MAC preamble (8 bytes), MAC source and destination address (6 bytes each), type (2 bytes) and cyclic redundancy check (CRC - 4 bytes). This totalizes 1538 bytes for a TCP data segment. As Ethernet have requires that the minimum frame payload to be 46 bytes, the 40-byte long ACK segment is padded with 6 extra bytes, totalizing 84 bytes. The transmission of TCP data and ACK segments over a 10 Mbps link takes respectively 1230.4μs and 68.8μs, while the transmission over a 100Mbps link takes respectively 123.04μs and 6.88μs.

5.1.2 802.11

For 802.11 protocols 36 additional bytes of data are added during the encapsulation process to the payload data by two distinct headers. The first 28 bytes form the

MAC header, which consists of frame control (2 bytes), duration ID (2 bytes), addresses 1 to 3 (6 bytes each), sequence control (2 bytes) and CRC (4 bytes). Note that the field *address 4* defined on the standard is only used when packets are transmitted between two access points over a distribution system [1]. In every other situation this field is omitted, which is the case in our scenario. The next 8 bytes form the Subnetwork Access Protocol (SNAP) header [29]. With the extra encapsulation TCP data packets are 1536 bytes long and TCP ACKs are 76 bytes long.

We also take into account the transmission of control frames. 802.11 protocols define that always after a data transmission an ACK (14 bytes) needs to be sent. CTS (14 bytes) frames are used in our laboratory environment for the CTS-to-self protection mechanism discussed on 2.2, we come back to the usage of this frame later in Section 5.1.2.3. At this point it is important to make clear that the term ACK refers to both TCP and link-layer acknowledgements, although the meaning should be clear based on the context.

We assume that the following sequence is taken for data transmission: (1) the sender waits for a DIFS period then (2) sends the 802.11 data frame (containing the TCP data or ACK segment), (3) upon the receipt of the data frame, receiver waits for a SIFS period and then (4) transmits the 802.11 ACK frame, concluding the process. Note that we consider an ideal case where the station always senses the medium free during the DIFS period, as we are calculating a minimum transmission time. Also link-layer errors/retransmissions are not taken into account.

The transmission time T_{SEG} of a TCP segment over a 802.11 link operating at TX bytes per microsecond with no protection mechanism can be estimated as:

$$T_{SEG} = T_{PHY} + \frac{(S_{SEG} + S_{ACK}) \times 8}{TX}, \tag{5.1}$$

where T_{PHY} is the overhead introduced by the physical layer, which consists of the DIFS and SIFS periods and the time necessary to transmit the physical layer header in each frame, S_{SEG} is the payload size of the TCP segment in bytes, i.e. 1536 bytes for data segments and 76 bytes for ACK segments and S_{ACK} is a constant equal to the size of a link-layer ACK, i.e. 14 bytes.

Finally, we also consider the extra overhead caused by the physical layer which is dependent on the 802.11 technology used. In our estimate of the transmission time we ignore the encoding made by the physical layer, i.e. assuming that the physical layer transmits bytes instead of symbols. This allows expressing the transmission time for different transmission rates in a simple mathematical formula, at the expense of losing precision.

For 802.11b no error is introduced by ignoring the physical layer encoding, as any frame size can be divided into an integer number of symbols by the physical layer. However that is not the case for 802.11a and 802.11g, where some padding might be needed. In the worst case scenario, every frame transmitted causes a symbol to be sent with only one bit of data payload. In the transmission sequence described above, we estimate the error to be twice an OFDM symbol time, i.e. $2 \times 4\mu s = 8\mu s$ (in the tranmission sequence described above two frames are transmitted, the data the and ACK frames). As the error introduced is always less than the measurement accuracy for the traces present in the validation of our method in Chapter 6, thus it does not have a large impact in our study.

In the following we show how the T_{PHY} is calculated for 802.11a, 802.11b and 802.11g technologies. We then provide estimates for the segment transmission time for the different transmission rates defined in these tecnologies.

5.1.2.1 802.11b

802.11b uses DSSS for the physical layer and defines a preamble to be included in every frame before transmission over the wireless hop. Two types of preamble are defined, the *long preamble* which requires $192\mu s$ to be transmitted and the *short preamble* which requires $96\mu s$. We consider only the *long preamble*, which is mandatory. DIFS and SIFS periods are $50\mu s$ and $10\mu s$ long, respectively. In this situation $T_{PHY} = 444\mu s$ (the preamble is added twice, once for the TCP segment and once for the link-layer ACK).

For this technology the minimum contention window consists of 31 slots of $20\mu s$, leading to a random backoff in the interval $[0, 620]\mu s$. Remember that all segments transmitted back-to-back are separated by a random backoff time.

Table 5.1 summarize the transmission time for data and ACK segments at the different speeds defined for 802.11b, based on Equation 5.1.

Table 5.1: Transmission times for 802.11b

Transmission Rate (Mbps)	TCP Data Segment (μs)	TCP ACK segment (μs)
1	12844	1164
2	6444	804
5.5	2698	575
11	1571	509

5.1.2.2 802.11a/802.11g

The 802.11a technology support much higher transmission rates using OFDM modulation. For each frame transmitted over the wireless hop a physical header (formed by a preamble and the PLPC header) is added, which takes $20\mu s$ to be transmitted. In 802.11a, DIFS and SIFS are $34\mu s$ and $16\mu s$ long, respectively. The physical overhead in this situation is $T_{PHY} = 90\mu s$.

802.11g also makes use of OFDM modulation, defining a physical layer very similar to the one from 802.11a. For each frame transmitted also a $20\mu s$ physical header is added, however it adds a $6\mu s$ *signal extension*, needed for coding purposes that is not present on 802.11a. The DIFS and SIFS periods are slightly smaller $28\mu s$ and $10\mu s$ long. Ultimately, the physical overhead is the same as in 802.11a, that is $T_{PHY} = 90\mu s$.

For both technologies the minimum contention window consists of 15 slots of $9\mu s$, leading to a random backoff in the interval $[0, 135]\mu s$.

Table 5.2 summarize the transmission times for data and ACK segments at the different speeds defined for 802.11a and 802.11g using OFDM modulation, based on Equation 5.1. Note that 802.11g is backward compatible with 802.11b, i.e. it can use the same physical layer as 802.11b, thus the transmission time reported in Table 5.1 are also valid for 802.11g stations.

5.1.2.3 802.11g with CTS-to-Self

802.11b stations are not capable of understanding OFDM modulation, in fact 802.11b stations are not even capable to perceive the medium being used by an 802.11g station. So, for co-existence of 802.11b and 802.11g stations on the

Table 5.2: Transmission times for 802.11a/802.11g

Transmission Rate (Mbps)	TCP Data Segment (μs)	TCP ACK segment (μs)
6	2156	210
9	1467	170
12	1123	150
18	778	130
24	606	120
36	434	110
48	348	105
54	320	103

same access point, protection mechanisms are defined. These protection mechanisms basically consist of the exchange of reservation packets in a slower, 802.11b-compatible modulation, before the transmission of data at the higher rates defined by the 802.11g standard, using OFDM.

In our last scenario we calculate the transmission time considering that the CTS-to-self protection mechanism is being used, situation observed on the *Laboratory* traces. This mechanism requires that before an 802.11g station initiates a data transmission, one CTS frame needs to be sent using 802.11b-compatible modulation and data rate, reserving the medium for a certain amount of time. With the medium reserved, the following 802.11 data and ACK can be exchanged using OFDM. The use of CTS-to-self implies the use of the DIFS and SIFS periods defined by 802.11b, that is $50\mu s$ and $10\mu s$ respectively. Also the minimum contention window is the same as in 802.11b, that is, 31 slots of $20\mu s$.

In this situation, the following sequence is followed: (1) sender senses the medium free for a DIFS period and (2) sends the CTS frame. Then (3) it waits for a SIFS period before (4) it finally sends the 802.11 data. Upon the receipt of the data frame the receiver (5) waits for a SIFS period and then (6) sends the 802.11 ACK. Once again, note that we consider the case where no other station tries to get access to de medium during this time and no transmission error occurs.

The total transmission time, T_{CTS}, can now be estimated as:

$$T_{CTS} = T_{PROT} + T_{SEG}, \tag{5.2}$$

where T_{PROT} is the extra delay caused by the use of the protection mechanism and T_{SEG} can be calculated using Equation 5.1.

T_{PROT} can be estimated as:

$$T_{PROT} = PLCP_{CTS} + \frac{S_{CTS} \times 8}{TX_{CTS}} + SIFS, \tag{5.3}$$

where $PLCP_{CTS}$ is the time necessary to transmit the physical layer header for the CTS frame, i.e. 192 μs, S_{CTS} is the size in bytes of a CTS frame, i.e. 14 bytes, TX_{CTS} is the transmission rate in which the CTS frame is exchanged in bytes per microsecond and SIFS is equal to $10\mu s$. The possible T_{PROT} times are shown in Table 5.3.

TABLE 5.3: CTS-to-self overhead (T_{PROT})

Transmission Rate (Mbps)	CTS-to-self Overhead (μs)
1	314
2	258
5.5	222
11	212

For the T_{SEG} calculation, T_{PHY} includes the 802.11b DIFS and SIFS ($50\mu s$ and $10\mu s$, respectively) and the physical header from 802.11g ($2 \times 20\mu s$) plus the *signal extension* ($2 \times 6\mu s$), totalizing $112\mu s$. The possible T_{SEG} times are showed on Table 5.4.

TABLE 5.4: T_{SEG} for 802.11g with CTS-to-self

Transmission Rate (Mbps)	Data Segment (μs)	ACK segment (μs)
6	2179	232
9	1490	192
12	1145	172
18	801	152
24	629	142
36	456	132
48	370	127
54	342	125

Any combination of values in Table 5.3 and Table 5.4 gives a valid transmission time for data and ACK segments. The minimum transmission time is achieved

by exchanging frames at maximum transmission rate, i.e. CTS frames at 11Mbps and the following frames at 54Mbps. In this situation the TCP data transmission is completed after $554\mu s$ while the ACK transmission after $337\mu s$. The maximum transmission time (when exchanging CTS frames at 1Mbps and following frames at 6Mbps) is $2493\mu s$ for data segments and $546\mu s$ for ACKs.

5.2 Duplex Capabilities

The duplex capabilities of the studied protocols have a major impact on the distribution of the inter-ACK time interval. Intuitively one could expect that inter-ACK times for half-duplex connections would be fairly larger than the ones from full-duplex connections, as on the former is not possible to have transmission in both directions at the same time. In this section we discuss the effects of the duplex capabilities of Ethernet and 802.11 protocols in detail.

Consider the scenario where a client downloads a large amount of data over a TCP connection. At a certain point of the communication the TCP parameters, such as congestion window and advertised receive window, allow the server to transmit four back-to-back data segments. Furthermore assume that these segments are practically undisturbed while crossing the path from the server to the client, i.e. they are not lost, follow the same path, do not encounter any queues on this path, etc., and arrive nearly back-to-back at the access link of the client. Following the TCP delayed ACK mechanism, assume that the second and the fourth packets trigger the transmission of an ACK segment by the client. Depending on the duplex capabilities of the access link the order in which these segments are transmitted is different and affects inter-arrival times of the ACK segments.

If the link is full-duplex, as in the case of Ethernet, the ACKs can be sent as soon as the data segments are received and processed by the client. We use a sequence diagram to represent this situation on Figure 5.1. For ease of visualization we number the TCP data segment according to the send order, and we number the TCP ACKs using the same number of data segment they refer to. It is clear that the inter-ACK time is dependent on the transmission time of the amount of data acknowledged by the second ACK in the pair. In this scenario the inter-ACK time is equal to the transmission time of the two data and one ACK segments at the monitoring point (we consider the processing time of the segments to be

negligible). In Section 5.4 we show that it is not the case that an ACK is *always* triggered by the reception of two data segments.

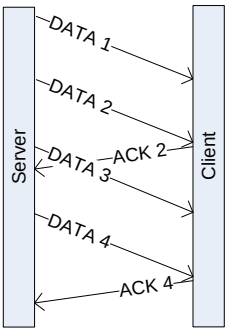


FIGURE 5.1: The full-duplex case.

Considering a half-duplex 802.11 network, the access link consists of an AP on one end and the client on the other. Note that we assume the access link to be the bottleneck, so data segments arrive on the access point faster than it can transmit them to the client. For our analysis this means that until the fourth data segment transmission is finished, the access point always tries to access the medium. The arrival of the second and fourth segments trigger the transmission of ACKs by the client, but, unlike the full-duplex case, the transmission might not take place immediately. On a half-duplex link when both AP and client have segments to send, they will compete for the medium. For simplicity assume that only the AP and the receiver client are trying to access the medium.

On this scenario the segments can be exchanged in any of the three orders depicted on Figure 5.2, depending on a number of factors like processing time of the segments on the client side and on the values chosen by the AP and client for the exponential backoff mechanism.

The receipt of *DATA 2* by client triggers the transmission of *ACK 2* making client and AP to contend for the medium. In Case *(a)* the client gets access to the medium in its first attempt, allowing the transmission of *ACK 2*. No more contention exists. If it was the AP to get access to the medium, segments *DATA 3* and *ACK 2* would contend for the medium. Case (b) shows the case where the client get access to the medium after receiving *Data 3*, delaying the transmission *ACK 2*. At last in Case *(c)* the AP get access at both times, causing the ACKs to be sent back-to-back.

Is easy to see that a similar behavior occurs if any bigger number of segments is sent by the server. The inter-ACK time on a half-duplex link is not dependent

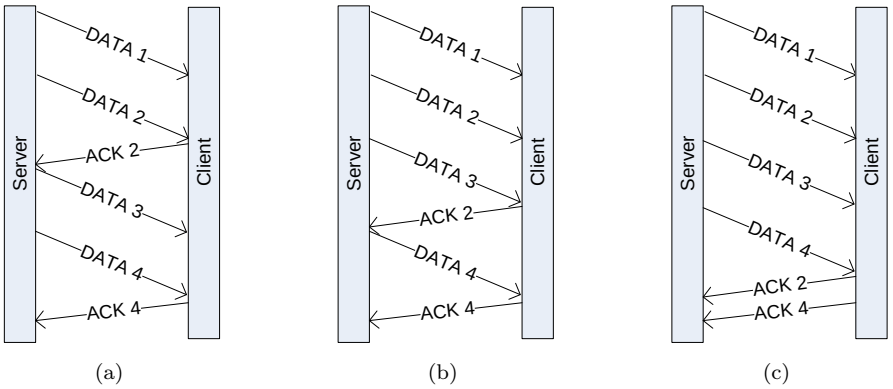


FIGURE 5.2: The half-duplex cases.

on the number of segments acknowledged by the second ACK of a pair, as in the full-duplex case, but on how these segments are actually transmitted over this link.

On a half-duplex link the inter-ACK time is composed by the transmission time of an ACK plus the transmission time of the n data packets transmitted between the ACKs in a pair. In our tests we observe n as large as 8.

5.3 The Inter-ACK Time Distribution

Based on the link transmission capacities and duplex capabilities discussion in the previous sections we propose a description of the inter-ACK distribution at the monitoring point. A basic assumption in our method is that the local network does not cause great perturbation to the inter-ACK times, i.e. segments crossing the local network will not face a large amount of cross traffic. Also, our analysis of the duplex capabilities needs data segments to arrive nearly back-to-back at the monitoring point. We want to be clear that we are not assuming that all data segments sent back-to-back by the server will not face cross-traffic on the path to the monitoring point. We simply discard ACK-pairs for which the data segments do not meet this criterion. In Section 5.4.2 we discuss other practical issues on the ACK-pair identification.

As discussed on the previous section, on Ethernet connections the inter-ACK time (ΔACK_{Eth}) is dependent on the number of data segments acknowledged by the second ACK in an ACK-pair. Considering that the delayed ACK mechanism is used, the inter-ACK time would be roughly equal to the transmission time of two data (T_{Data}) segments, which can be expected to be a nearly constant value.

The probability density function (PDF) of the inter-ACK time would then have a single dominant mode at:

$$\Delta ACK_{Eth} = 2 * T_{Data}. \tag{5.4}$$

A sketch of the expected inter-ACK time PDF for a Ethernet connection is depicted in Figure 5.3. The existence of cross traffic could either compress or expand the time between the ACKs, causing some noise before and after the dominant mode. Even if no cross traffic is present, some variation is expected due the time necessary to process the segments.

Based on the results from Section 5.1, for Ethernet 10Mbps the mode is located at $\Delta ACK_{Eth} = 2 * 1230.4 = 2460.8\mu s$ and for Ethernet 100 Mbps it is located at $\Delta ACK_{Eth} = 2 * 123.04 = 246.08\mu s$.

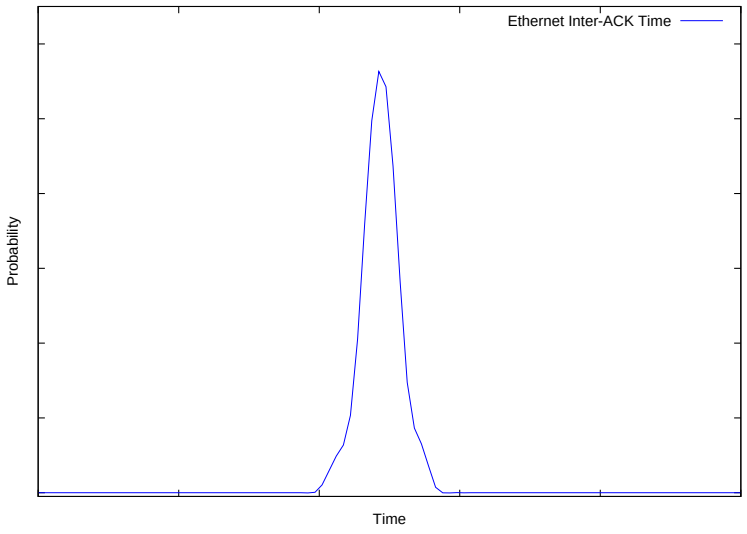


FIGURE 5.3: Sketch of the PDF for the inter-ACK time for Ethernet

For 802.11 connections the inter-ACK time ($\Delta ACK_{802.11}$) is dependent on the transmission order of the segments on the wireless link. More specifically it is roughly equal to the transmission time of n data segments plus the transmission time of an ACK (T_{ACK}) over this link, where n is the number of data segments transmitted in between the ACKs in a pair. The PDF for the inter-ACK time in this case can be expected to be multi-modal, with modes at:

$$\Delta ACK_{802.11} = n * T_{Data} + T_{ACK}. \tag{5.5}$$

While the segment transmission times give us an approximation of the position of the modes, it is important to note that the CSMA/CA protocol causes 802.11 nodes to always wait a random backoff time before a frame transmission to avoid channel capture (see Section 2.2). The size of this random interval is dependent on the technology used and on the number of transmission attempts. Assuming that all frames transmissions are successful, i.e. no collisions or errors occur during transmission, the random backoff is always chosen from the minimum contention window.

We expect the inter-ACK distribution to have broader and weaker modes as n increase, as more random times are added to the inter-ACK time, achieving a point where they are no longer discernible from noise.

On Figure 5.4 a sketch of the PDF for 802.11 connections is presented. The first mode represents the transmission of back-to-back ACKs on the wireless hop, and the following modes are separated by the time transmission of one TCP data segment. We expect to have some noise around the modes caused by cross-traffic and the time necessary to process the segments as in the Ethernet case.

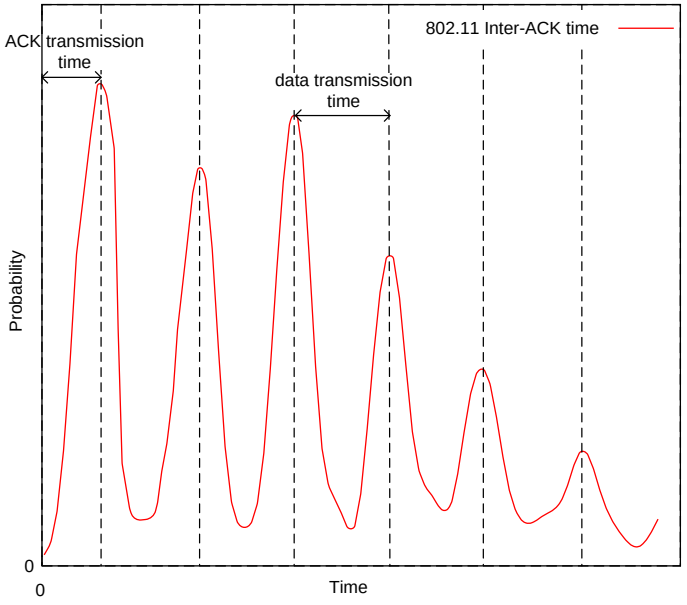


FIGURE 5.4: Sketch of the PDF for the inter-ACK time for 802.11

A nice *side effect* of our approach to differentiate Ethernet from 802.11 connections is that the position of the modes gives us information on the transmission rate that the protocol is operating in. This is especially interesting for 802.11 connections, as the transmission rates can vary depending on the quality of the

wireless link. The identification links operating at lower than expected transmission rates (54Mbps for 802.11a and 802.11g, and 11Mbps for 802.11b) provide useful information for network administrators, as repositioning the existing AP's or adding new ones could make the network perform better.

The position of the first mode in the inter-ACK time distribution can also give us some extra information. As it can be observed in on Section 5.1.2, the transmission time for an ACK segment varies considerably depending on which 802.11 technology is used. If the position of the first mode is at $210\mu s$ or before, it is an indication that OFDM is in use. If the first mode is in the interval $[337, 546]\mu s$, it is an indication of the use of CTS-to-self. If the first mode is at $509\mu s$ or after, it is an indication that DSSS is in use. Note that in the case that the first mode is present in the interval $[509, 546]\mu s$, both DSSS and CTS-to-self may be in use.

5.4 Practical Aspects

In this section we discuss practical problems we have to deal with for the identification of ACK-pairs in real world situations. We first comment on the implementation of TCP in different operating systems (OS's), as it is important to understand when ACKs are generated by them. This give us a base to discuss the identification of ACK-pairs *per se*, including issues like lost packets and reordering.

5.4.1 TCP ACK in Different OS's

Through our laboratory experiments we were able to observe that the implementation of TCP is slightly different in the three tested OS's: Linux kernel 2.6.22, Windows XP and Mac OS X Leopard. It turns out that the delayed ACK mechanism as described in RFC 1122 [11] allows different implementations. The first point is that [11] only establishes a maximum delay of $500ms$ for ACKs, when it states: *"... in particular, the delay MUST be less than 0.5 seconds, ..."*. This means that the actual value for the delayed ACK timer is implementation dependent and does not even need to be a constant.

Secondly it states: *"... in a stream of full-sized segments there SHOULD be an ACK for at least every second segment."*. Note the use of the word *should*, which

in RFC standards represents a recommendation rather than one requirement for compliance. This means that TCP implementations are allowed to delay an ACK for more than two segments, i.e. acknowledging only the third or fourth data segments are valid implementations.

In Windows XP the delayed ACK timer and the number of received data segments that trigger an ACK are controlled by two registry entries: *TcpDelAckTicks* and *TcpAckFrequency*. Both have *2* as default value, meaning that the delayed ACK timer is $200ms$ (2 periods of $100ms$) and every second data segment is acknowledged. To some extent we can verify these values on the laboratory traces. We see that ACKs are always sent in response to 1 or 2 data segments. When the second ACK of a pair is generated in response to a single data segment, the inter-ACK time tends to be in the order of hundreds of milliseconds, an indication that this ACK was generated by a delayed ACK timer expiration.

As *open source* software the Linux kernel 2.6 source code is freely available and can be found on the Internet [30]. This makes possible to study their TCP implementation. On this OS the TCP delayed ACK timer is variable and its value is constantly refined to the minimum of the interval between the arrival of data packets and the *sample round trip time* (srtt). Furthermore, the timer has a lower bound of $40ms$ and an upper bound of $200ms$.

In Linux, upon the receipt of a data segment an immediate ACK is sent in three situations:

1. More than one MSS worth of data is received and the receiver buffer has space for accepting advertised window worth of data;
2. TCP is in *quick ACK* mode, and;
3. Data received is out of order

If none of the situations is true, a delayed ACK is triggered. The first situation represents how the RFC 1122 statement, *"... in a stream of full-sized segments there SHOULD be an ACK for at least every second segment."*, is interpreted. The reception of a second full-sized segment causes the check "more than one MSS worth of data" to be true. But an immediate ACK is sent only if the receiver buffer has available space in buffer for accepting the last advertised window worth of data.

In practice, normally an ACK is sent for every second data segment, but in some cases is can be also triggered by any n number of data segments received before the delayed ACK timer expires, depending on the available space on the TCP buffer.

The *quick ACK* mode, mentioned as situation 2, is enabled every time the receiver infers that the sender is in *slow start*, e.g. on the beginning of a connection or after detecting a lost segment. When operating in this mode the receiver sends an ACK for every data segment received with the objective of making the congestion window on the sender side to grow as fast as possible. The third and last situation consists in sending an ACK every time an out of order is received, which is a recommendation of RFC 1122.

As we were not able to obtain official information about the Mac OS implementation of TCP is hard to say exactly how ACKs are generated. In our experiments we see ACKs after different quantities of data segments having a low inter-ACK time ($< 1ms$), which would indicate that they are not delayed ACKs. We also did not find any official information on the delayed ACK timer value, but it is reasonable to assume that it is at maximum $200ms$ as in the other OS's considered.

In summary when dealing with different OS's one should expect differences in the protocol implementations. On our analysis we assumed the number of received data segments that trigger an ACK to be exactly 2, as recommended by the RFC. Having ACKs being triggered by different number of segments can change the single dominant mode on the distribution of inter-ACK times for Ethernet connections.

This behavior is illustrated using a laboratory trace generated with a Linux client using a 100Mbps Ethernet connection. As discussed above, the Linux implementation of TCP can generate ACKs immediately after different number of data segments. On Figure 5.5 we plot the inter-ACK time discretized using $125\mu s$ bins, which is the accuracy of our measurements (see Section 4.1) versus the fraction of ACK-pairs in each of these bins. Each line represents the situation where the second ACK in the pair is sent in reply to 1, 2, 3 or 4 data segments. As expected, in the 1 data segment case most of the inter-ACK times are in the $125\mu s$ bin, and the following case's peeks are separated by $125\mu s$, approximately the time to transmit a full-sized TCP data segment over a 100Mbps link.

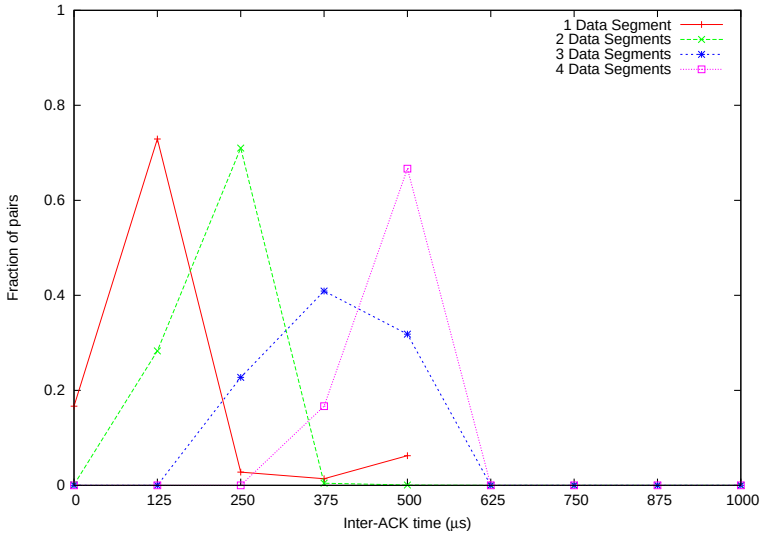


FIGURE 5.5: Inter-ACK time for an 100Mbps Ethernet connection where the second ACK in the pair is sent in reply to 1, 2, 3 and 4 data segments.

5.4.2 ACK-pair Detection

We use the fact that the interval between consecutive ACKs carries useful information about the characteristics of the access points when both data and ACK segments are carried undisturbed through the network. Unfortunately that is not always the case, it would actually be naive to assume that all segments would be undisturbed. Here we discuss some of the issues on identifying ACK-pairs on network traces.

TCP segments can be lost or reordered on the path from the server to the client. In situations where this happens the inter-packet time in this situation would not reflect aspects of the medium but TCP characteristics, so consecutive ACKs that have their respective data segments retransmitted or reordered are not considered in our analysis.

By the lessons learned while studying the TCP implementations discussed in 5.4.1 we discard consecutive ACKs that are more than $200ms$ apart, as this would be an indication of a delayed ACK expiration. Once again, this situation would not reveal the medium aspects. We also restrict our analysis to ACKs generated after 2 data segments to be consistent with Section 5.3.

Finally our analysis depends on the transmission of (nearly) back-to-back segments by the access network link. Thus we only consider ACKs which respective data segments are less than $500\mu s$ apart at the monitoring point.

Chapter 6

Validation

In this chapter we present the validation our method, using traces generated on semi-controlled laboratory experiments and *real-world* network traces collected at our university. We also present some unexpected results that are not part of the scope of this work.

6.1 Laboratory Traces

In this section we confirm our predictions about the inter-ACK time distribution using data gathered in the semi-controlled laboratory environment described in Section 2.5. We perform tests with different clients using the three operating systems that were used for the study of the TCP acknowledgements in Section 5.4.1: Windows XP, Linux kernel 2.6.22, and Mac OS Leopard.

For this analysis we use histograms with $125\mu s$ bins, which is the smallest bin allowed by our measurement accuracy. Following the reasoning presented in Section 4.2.3, our measurements are concentrated in clusters every $125\mu s$ and we want that all points in the same cluster contribute for the same bin of the histogram. For that we define the beginning of the histogram at -62.5 instead of zero, i.e. the first bin covers all inter-ACK times in the interval $[-62.5, 62.5)\mu s$, the second covers the interval $[62.5, 187.5)\mu s$, and so on. A last note is that we use lines instead of the traditional boxes to plot the histograms, this facilitates visualization for cases where more than one curve is depicted in the same graph.

The points over these lines represent the mean value of the histogram bin, e.g., 0 identifies the first histogram, 125 the second, and so on.

6.1.1 Ethernet results

The first set of results are from 100Mbps Ethernet connections from a client running Windows XP. In the graph plotted in Figure 6.1 is possible to observe a single peek in the distribution of the inter-ACK time at $250\mu s$. This matches the predicted behaviour by the analysis done in Section 5.3. The histograms for Linux and Mac OS are depicted in Figure 6.2 and Figure 6.3 respectively. Once again a single mode is present on the distribution as expected. All TCP flows used have at least 1000 ACK-pairs.

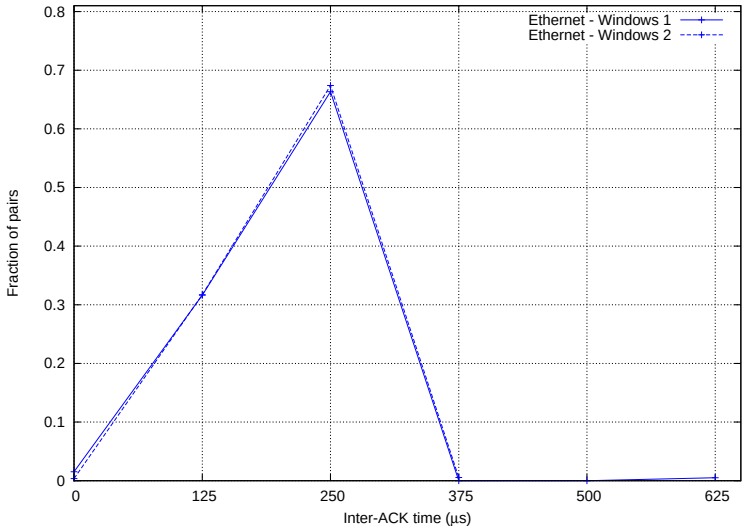


FIGURE 6.1: Inter-ACK time for an 100Mbps Ethernet connection in Windows XP.

We also performed tests 10Mbps Ethernet connections using a Mac OS client. In the results depicted in Figure 6.4 a single mode in the inter-ACK time distribution is present, although the value of the mode is slightly smaller then the one expected. We observe that most of the inter-ACK times concentrate at $2125\mu s$ and $2250\mu s$ while we expect this mode to be approximately at $2460.8\mu s$. This difference is mainly caused by the size of the TCP data segments, while we assumed 1538 bytes in our analysis, in these flows their size is 1342 bytes. Performing the calculations, one can check that the expected mode position for this segment size corresponds to the one observed in this graph.

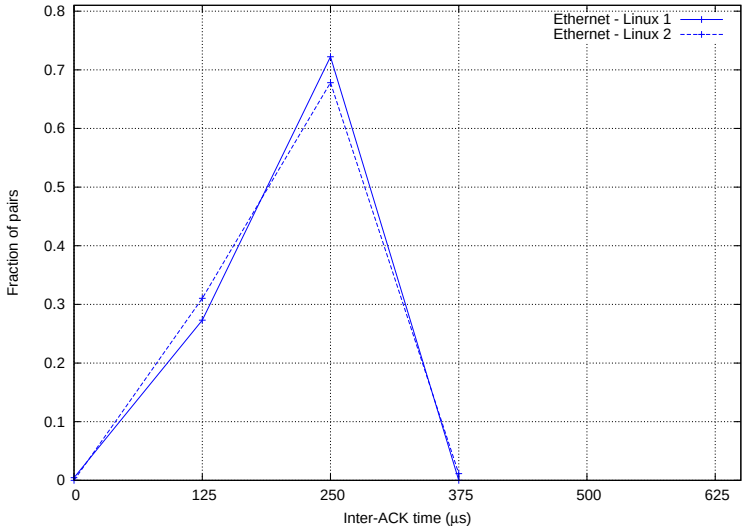


FIGURE 6.2: Inter-ACK time for an 100Mbps Ethernet connection in Linux kernel 2.6.

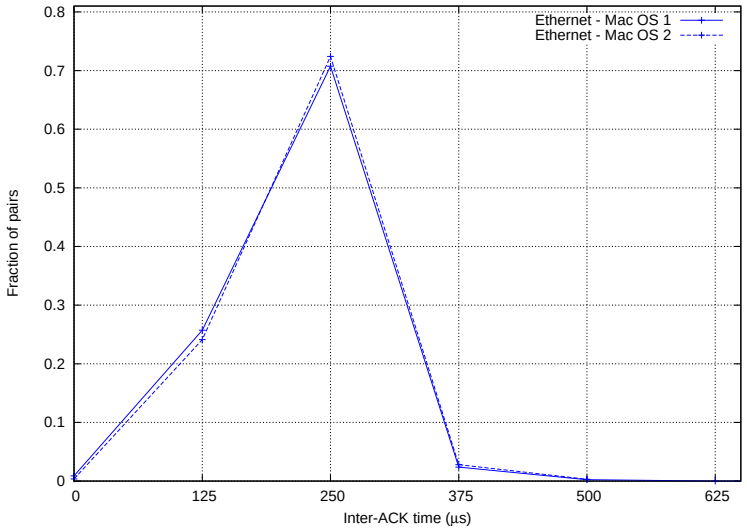


FIGURE 6.3: Inter-ACK time for an 100Mbps Ethernet connection in Mac OS Leopard.

For the half-duplex connection, plotted on Figure 6.5, we would expect to have multiple modes on the inter-ACK distribution as in 802.11 connections, but this behavior is not is not clear in this graph. The inter-ACK time for pairs transmitted back-to-back is approximately $68.8\mu s$, and, considering our accuracy, these pairs could contribute for either the first or the second bins. The next modes should be separated by the time of a full-sized TCP segment, which is $1230.4\mu s$ in this situation.

A possible explanation for the multiple mode behavior do not appear in this

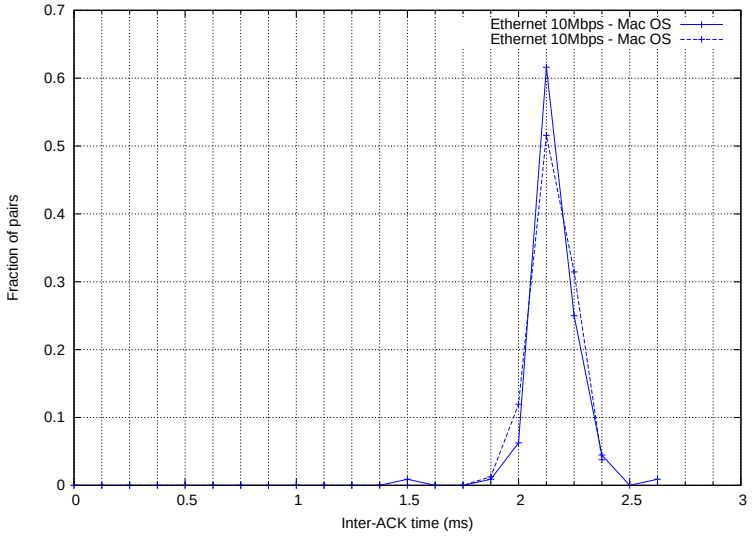


FIGURE 6.4: Inter-ACK time for an 10Mbps Ethernet connection in Mac OS Leopard and Linux.

graph is the *channel capture effect* described in Section 2.1. This effect would cause the TCP transmission to occur in large bursts. Once the sender captures the channel, it transmits all data segments it has. Following the delayed ACK mechanism, every second data segment received triggers an ACK transmission, however these segments are not transmitted by the link layer while the medium is captured by the sender. After the sender finishes it transmission, the client starts transmitting its delayed ACKs, possibly capturing the channel. When these ACKs are received by the sender, it updates its TCP window, sending a large amount of data again, and possibly capturing the channel and restarting the procedure. As ACKs are sent back-to-back most of the times, we have a large, dominant mode at the first two bins of the graph.

6.1.2 802.11 results

We now show the observable behavior of 802.11 connections. In all tests shown in this section, the clients are connected using an 802.11g interface. For compatibility reasons the CTS-to-self protection mechanism is used in our network, as reported in 5.1.2, and CTS frames are exchanged at 1Mbps. Considering that the stations are operating at maximum speed (54Mbps), the expected transmission times for a data and for an ACK segment in this situation are $656\mu s$ and $546\mu s$ (see Equation 5.2 and Tables 5.3 and 5.4).

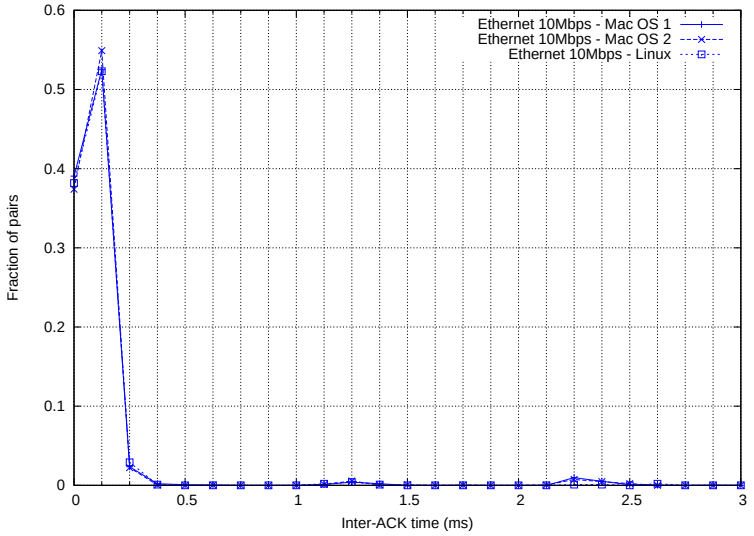


FIGURE 6.5: Inter-ACK time for an 10Mbps half-duplex Ethernet connection on Mac OS Leopard.

Figure 6.6 show the results for Windows XP clients. The flows represented by lines *Windows 1* and *Windows 2* are quite similar. These flows have 1157 and 1074 ACK-pairs, respectively. A first mode is present at $500\mu s$ and the three following modes are separated by 5 or 6 bins ($625\mu s$ or $750\mu s$). Considering the data and ACK segments transmission for this case and the accuracy of our measurements, the expected behavior is confirmed.

For the flow represented by line *Windows 3* (681 ACK-pairs) the first mode is also present at $500\mu s$, but the following modes are further apart when comparing to the other flows. In this case modes are separated by 6 or 7 bins ($750\mu s$ or $875\mu s$). This indicates that the wireless network was operating in a slower transmission rate (36Mbps), probably caused by a lower link quality in the moment of the measurement.

A closing remark about this experiment related to flow *Windows 2*. This was the same flow used to plot the $500\mu s$-bin histograms in Figure 4.9 and Figure 4.10. Observing Figure 6.7, where this flow is plotted using both $500\mu s$ and $125\mu s$ bins, it becomes clear why the *peek-gap-peek* behavior was not visible at first. It was simply hidden by the position of the bins. When we included a small bin of $125\mu s$ in Figure 4.10, the behavior appears as the peeks become separated by one bin.

In Figure 6.8 we present the results for Mac OS Leopard clients. The flows Mac OS 1, 2 and 3 have respectively, 1471, 2664 and 3144 ACK-pairs. At these traces the first mode has considerably more pairs than the others. Another unexpected

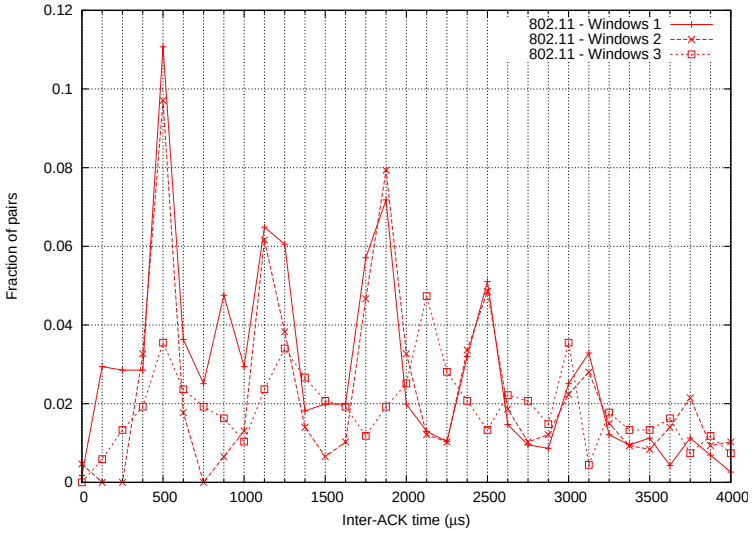


FIGURE 6.6: Inter-ACK time for 802.11 connections in Windows XP.

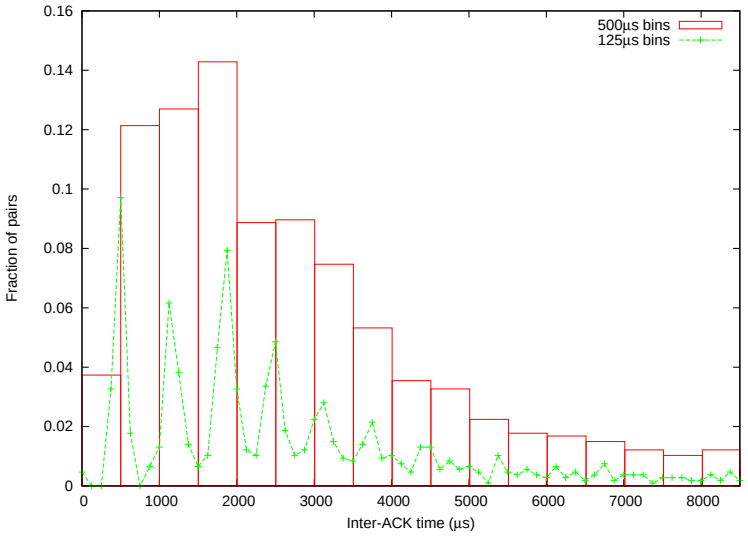


FIGURE 6.7: Inter-ACK time for an 802.11 connection in Windows XP using $125\mu s$ and $500\mu s$ bins.

behavior is the large concentration of pairs with inter-ACK in the first 3 bins (smaller than $250\mu s$), differently from what is observed in the Windows XP traces. We address this issue later on.

The other modes, caused by the transmission of data segments in between the ACK-pairs, still exist. As in the Windows traces, the position of the first of these modes is approximately at $1250\mu s$ and they are separated by $625\mu s$ or $750\mu s$, fitting our analysis. The flow Mac OS 1 does not present the first of these modes. This shows that some flows may not follow the inter-ACK distribution described in Chapter 4.

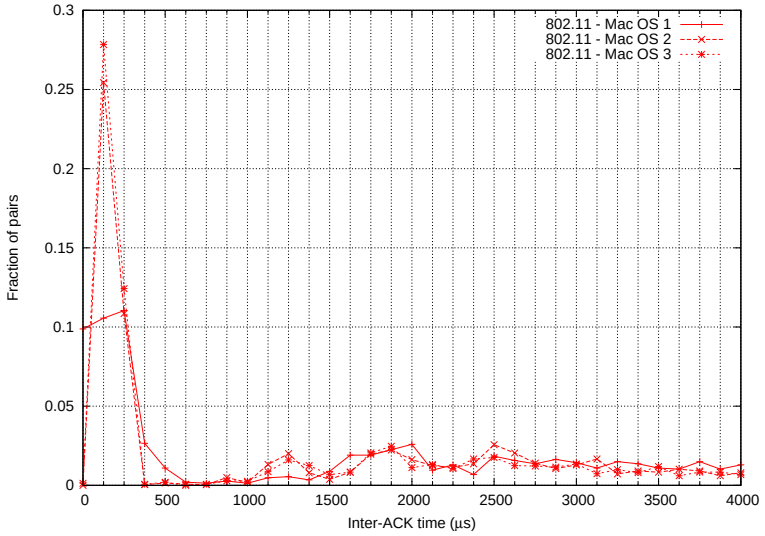
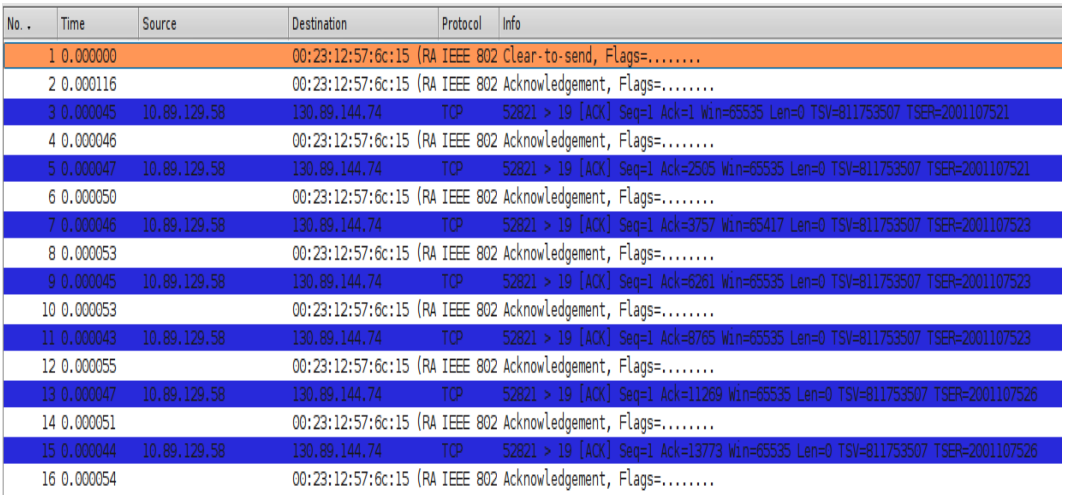


FIGURE 6.8: Inter-ACK time for 802.11 connections in Mac OS Leopard.

To understand why so many inter-ACK times are smaller than $250\mu s$ we study these flows with the traces made in the wireless hop. The 802.11 standard [1] defines that transmitted frames from a station are always separated by at least one random backoff interval, to avoid channel capture. Moreover, when monitoring a TCP connection in a wireless hop where the CTS-to-self mechanism is being used, one expects that every TCP ACK sent by a station is preceded by a CTS frame and followed by an ACK frame. However this does not seem to be the case for the monitored Mac OS client.

In Figure 6.9 we show an excerpt from the traffic captured on the wireless hop. The *time* column specifies the interval in seconds since the previous captured frame. This excerpt contains seven consecutive ACKs sent by the Mac OS client, displayed in blue. Note that the time between two ACKs is nearly constant, approximately ranging from $91\mu s$ to $98\mu s$. We would expect a much larger variation due to the backoff timer, which is defined in multiples of $9\mu s$. It is also possible to observe in this excerpt that all TCP ACKs are followed by link-layer ACKs, but none is preceded by a CTS frame.

The only CTS frame showed is followed by an ACK, with no data frame in between. It is likely that the data frame was not capture by the monitoring point, but it reached the intended receiver. This ACK contains the MAC address (see field *Destination* in the figure) of the monitored Mac OS client, so possibly the lost frame is an eighth TCP ACK.



No. .	Time	Source	Destination	Protocol	Info
1	0.000000		00:23:12:57:6c:15 (RA	IEEE 802	Clear-to-send, Flags=........
2	0.000116		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
3	0.000045	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSV=811753507 TSER=2001107521
4	0.000046		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
5	0.000047	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=2505 Win=65535 Len=0 TSV=811753507 TSER=2001107521
6	0.000050		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
7	0.000046	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=3757 Win=65417 Len=0 TSV=811753507 TSER=2001107523
8	0.000053		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
9	0.000045	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=6261 Win=65535 Len=0 TSV=811753507 TSER=2001107523
10	0.000053		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
11	0.000043	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=8765 Win=65535 Len=0 TSV=811753507 TSER=2001107523
12	0.000055		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
13	0.000047	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=11269 Win=65535 Len=0 TSV=811753507 TSER=2001107526
14	0.000051		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........
15	0.000044	10.89.129.58	130.89.144.74	TCP	52821 > 19 [ACK] Seq=1 Ack=13773 Win=65535 Len=0 TSV=811753507 TSER=2001107526
16	0.000054		00:23:12:57:6c:15 (RA	IEEE 802	Acknowledgement, Flags=........

FIGURE 6.9: Excerpt from a TCP connection captured on the wireless hop generated using a Mac OS client.

The same behavior can be observed throughout the trace, consecutive TCP ACKs being sent with no visible backoff timer in between, and only the first of the ACKs is preceded by a CTS frame. While this explains why so many ACK-pairs have such small inter-ACK times, it is not an expected behavior from an 802.11-compliant station. As these traces were all generated by the same client, we cannot be sure that this is a general behavior in Mac OS implementations or an isolated case due to hardware/software malfunctioning.

6.2 UT Traces

We further validate our work with traces collected in our university campus gateway (see Section 2.5). In this data set we are able to distinguish the access network type of the hosts based on their IP address, as the Ethernet and 802.11 are different subnets. However we do not have detailed information on the exact technology used, i.e. Ethernet hosts can be operating at 10Mbps, 100Mbps or 1Gbps transmission rates, while 802.11 hosts can be using 802.11a, 802.11b or 802.11g technologies. The consequence is that we are not able to confirm our results for transmission rates.

Following the reasoning from Section 4.1.2, we estimate the precision of our measurements based on the observed time resolution of the inter-ACK and inter-data times. In Figure 6.10 is plotted the inter-ACK time versus the inter-data time calculated on randomly selected TCP flow from this data set. It is clear that

all data points are concentrated in clusters spaced $50\mu s$ from each other in both axis. This behavior is recurrent in all TCP flows used in this analysis.

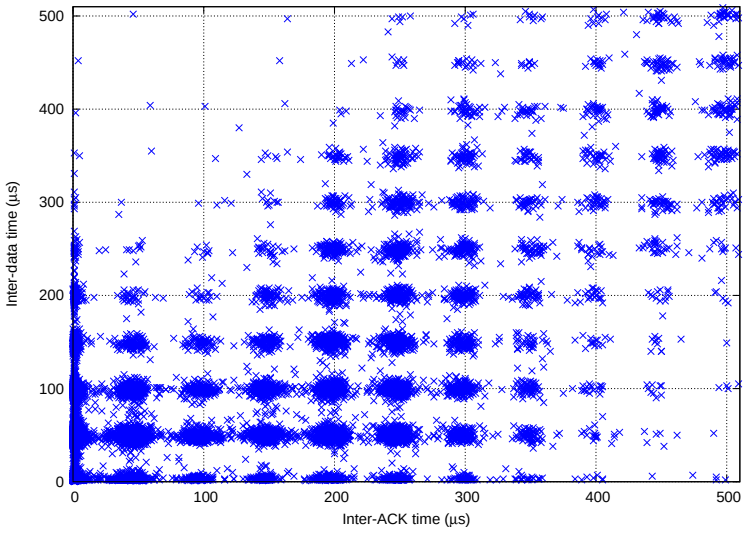


FIGURE 6.10: Data points concentrated in multiple of $50\mu s$ indicating the precision of the measurements.

This accuracy allows us to generate histograms a $50\mu s$. We make all clusters to contribute to the same histogram bin by making the first histogram bin to start at $-25\mu s$ instead of zero. This way, the first bin includes all points in the interval $[-25, 25)\mu s$, the second covers the interval $[25, 75)\mu s$ and so on. As before, we use lines instead of the traditional boxes to plot the histograms, and the points present in the lines represent the mean value of a bin.

6.2.1 Ethernet results

Figure 6.11 shows the results for some random selected Ethernet flows present in the data set. As expected, the inter-ACK distribution for these flows contains a single mode, which is present at $250\mu s$ indicating that all hosts are connected to 100 Mbps access links.

Two observations are made about these results. First, in comparison with the results from the experiments done in our laboratory environment, the flows *Ethernet 1* and *Ethernet 4* present a considerably broader mode. For instance, Ethernet 4 hast almost 10% of the inter-ACK times in the $400\mu s$ bin, while in all laboratory experiments have less than 5% of the inter-ACK times bigger than $375\mu s$.

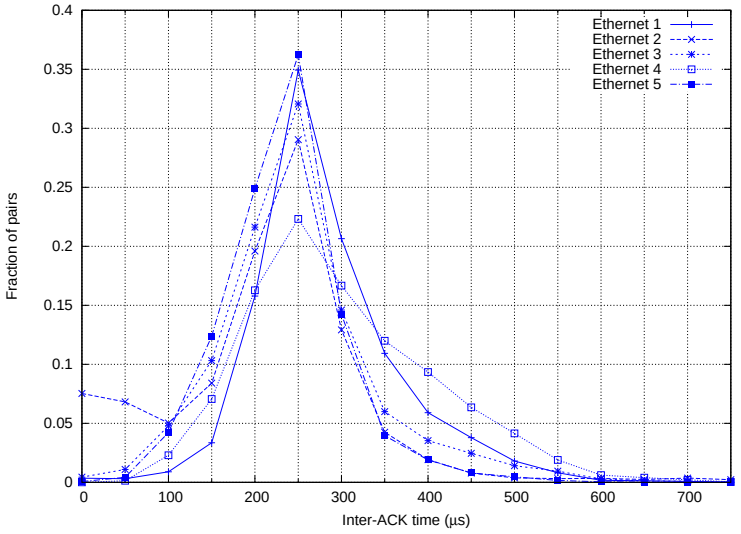


FIGURE 6.11: Inter-ACK time for 100Mbps Ethernet connections.

Second, the flow *Ethernet 2* presents an unexpectedly large amount of intervals in the first two bins. Both the larger (present in flows Ethernet 1 and Ethernet 4) and the shorter (present in flow Ethernet 2) inter-ACK times can be caused by cross-traffic faced by the ACKs in a pair in the path from the client to the monitoring point.

Packets from some other flow can be inserted between the ACKs in a pair by some router in this path, leading to a larger inter-ACK time observation. A shorter inter-ACK time can be observed if the first ACK in a pair faces queuing in a router in this path. When this happens it may be the case that the ACK pair will be transmitted back-to-back by this router. If the router is connected to a link with higher transmission rate than the access link the inter-ACK time is shorter than the one expected by our method.

6.2.2 802.11 results

When examining the UT traces for wireless flows, we realize that many of the flows did not contain sufficient ACK-pairs for our analysis. We notice that the TCP flows from wireless hosts are normally smaller than the wired ones. Also due to the large amount of packet loss in this data set, some large TCP flows do not contain sufficient ACK-pairs for our analysis.

In our experiments we see that the multiple modes behavior for 802.11 connections is not always clear when fewer than 1000 ACK-pairs are present. However for large traces (more than 10000 ACK-pairs) the behavior is as described in Section 5.3. For this analysis we only use traces that have at least 700 ACK-pairs. In Figure 6.12 show the effect of the number of observations in our analysis. In this figure we plot the same flow, with different number of pairs. Note that the multiple mode behavior is clear when all 11143 ACK-pairs are used. When 700 pairs are used it is possible to observe the three first modes, however when fewer packets are used, is impossible to distinguish the modes from noise.

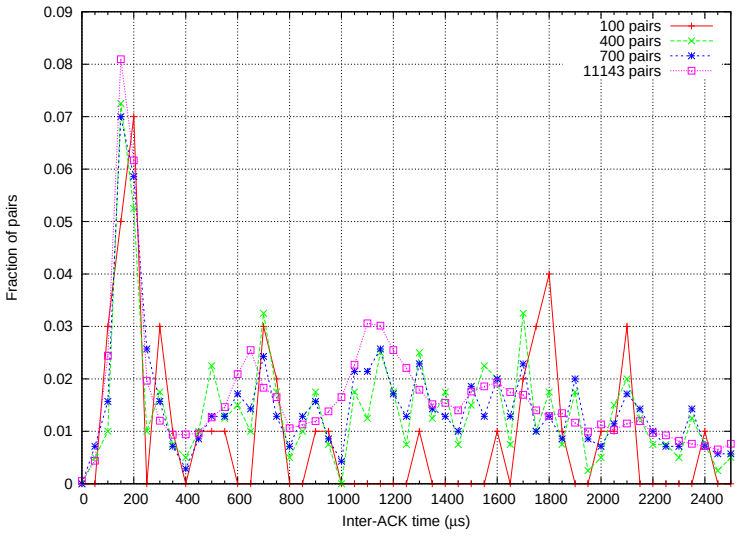


FIGURE 6.12: Inter-ACK time distribution for different number of observations.

The histograms for the 802.11 flows with the largest number of ACK pairs are plotted in Figure 6.13. The 802.11 Flows 1, 2 and 3 are from the client and contain respectively 11143, 12823 and 12905 ACK pairs. All flows present the same behavior, with four clear modes in the inter-ACK distribution, characterizing the half-duplex nature of the 802.11 hop. It is also possible to observe that the modes get broader as the inter-ACK time increases, what can be explained by the random backoff times introduced by the CSMA/CA protocol, as described in 5.3.

In this graph, the first mode is present approximately at $150\mu s$ indicating the use of OFDM and the space of approximately $500\mu s$ between the modes indicates that the transmission rate is 36Mbps (the transmission time for a TCP data segment at this speed is $434\mu s$, and the random backoff is in the interval $[0, 135]\mu s$).

Figure 6.14 shows two other flows, from distinct clients, with a similar behavior. The 802.11 Flows 4 and 5 contain 729 and 713 ACK pairs, respectively. As these

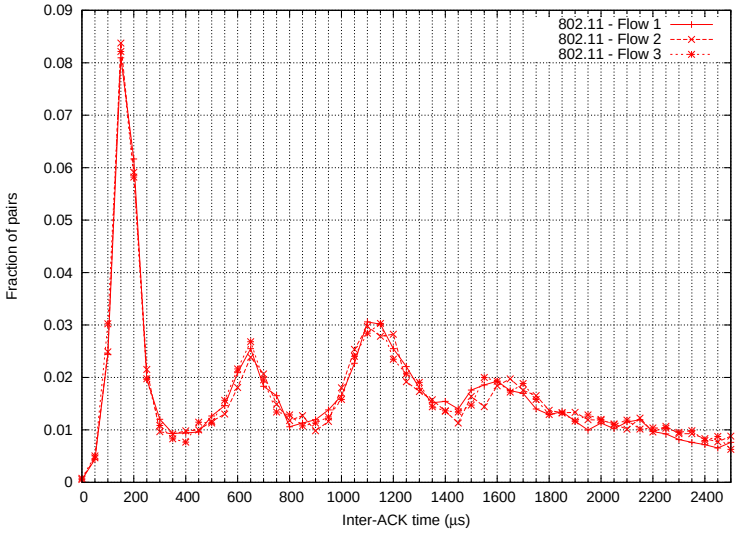


FIGURE 6.13: Inter-ACK time distribution suggesting a 802.11 station using OFDM operating at 36Mbps.

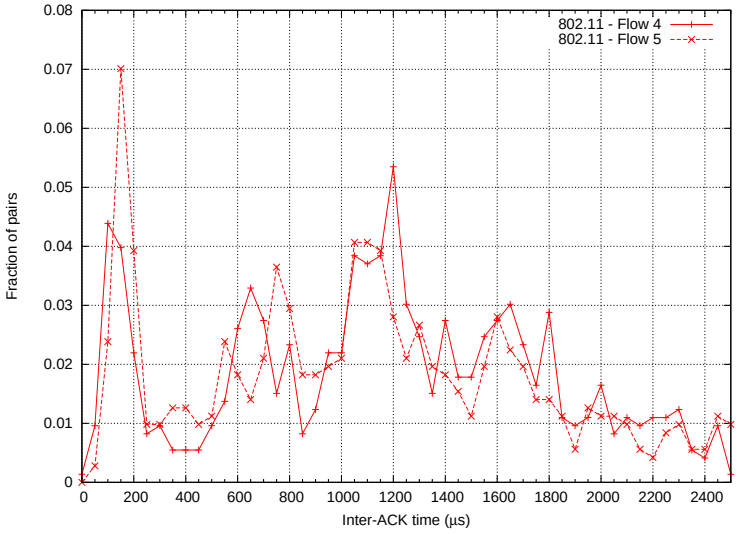
flows contain considerably fewer data points than Flows 1, 2 and 3, when plotting the histogram with 50μs bins (Figure 6.14(a)) a large amount of noise is present in the graph. However when making the same plot using 100μs bins (Figure 6.14(b)) the modes become more visible.

In this graph the first mode is located before 210μs, indicating the use of OFDM. Modes are separated by 500μs or 600μs, still in accord with the transmission rate of 36Mbps, as the flows in Figure 6.13.

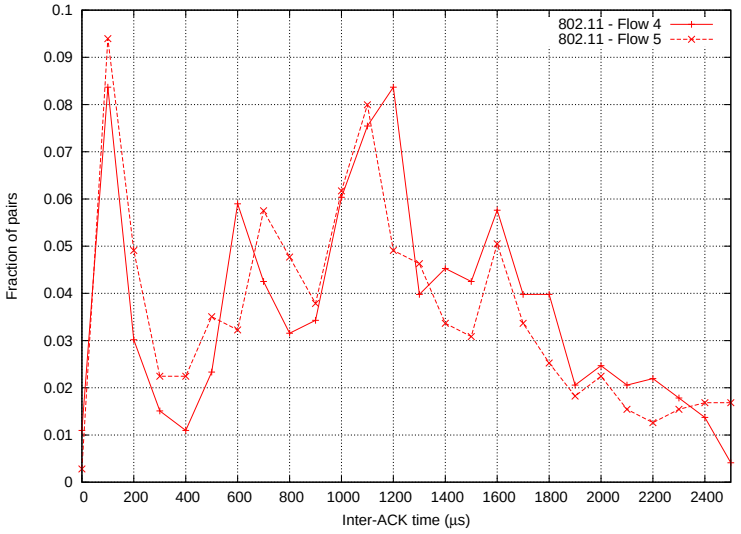
Figure 6.15 shows two flows with different behaviors. The Flows 6 and 7 are from different clients and contain 982 and 785 ACK pairs, respectively. As in the case of Flows 4 and 5, the modes are better visualized using 100μs bins. Flow 6 present the first mode at 100μs, with two visible following modes spaced by roughly 400μs or 500μs, describing the behavior an 802.11 station using OFDM operating at 48Mbps.

For Flow 7, the first mode is present approximately at 300μs, indicating the use of CTS-to-self mechanism. The modes separated by roughly 600μs. Various combinations of T_{SEG} and T_{PHY} (see Section 5.1.2.3) can fit this values, thus is not possible to infer the transmission rate for this case.

The last 802.11 flows used for this analysis are present in Figure 6.16. The Flows 8, 9 and 10 are from the client and contain respectively 694, 1004 and 1059 ACK pairs. The inter-ACK times present in these flows are substantially bigger



(a) $50\mu s$ bins



(b) $100\mu s$ bins

FIGURE 6.14: Inter-ACK time distribution suggesting 802.11 stations using OFDM operating at 36Mbps with fewer ACK pairs.

than the ones analyzed before, as it can be observed in this figure. While multiple modes can be visualized in this distribution using $50\mu s$ bins (Figure 6.16(a)), the large amount of noise makes hard to determine the location of the modes.

Figure 6.16(b) presents the same graph with $200\mu s$ bins, which makes the behavior to become clearer. The first peek is locate at the interval $[400, 600]\mu s$, characterizing the use of DSSS for the physical layer. The following peeks separated by roughly $1600\mu s$ or $1800\mu s$. Taking into account the large random backoff introduced by this technology, the minimum contention window can introduce an extra delay up to $620\mu s$, this behavior fits a transmission rate of 11Mbps.

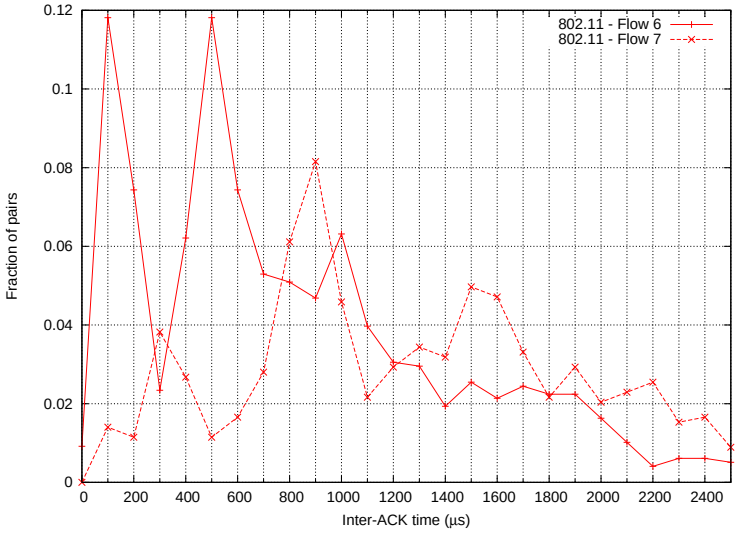


FIGURE 6.15: Inter-ACK time distribution suggesting 802.11 stations transmitting at different data rates.

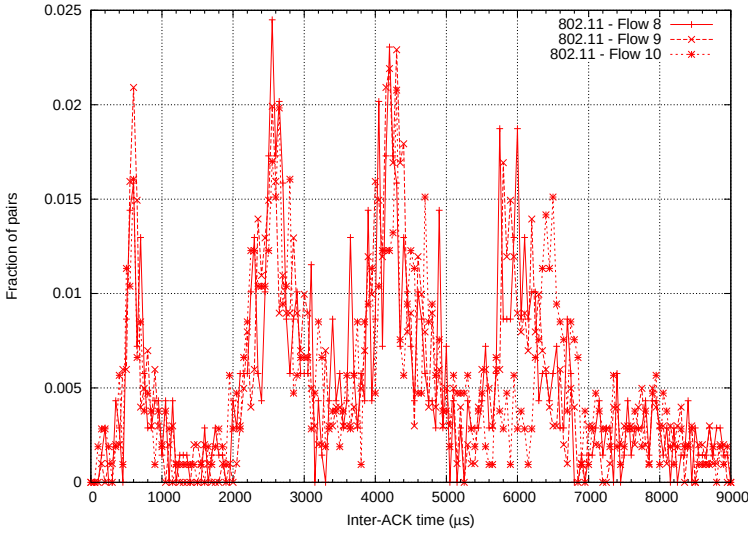
6.2.3 Unexpected Results

In this section we discuss some results that are not part of the scope of this work. Figure 6.17 shows the first unexpected behavior. This flow is generated by a host with IP address in the range reserved for Ethernet connections, however, the behavior showed on this graph does not fit neither the 10Mbps or 100Mbps Ethernet inter-ACK distribution described on Section 5.3.

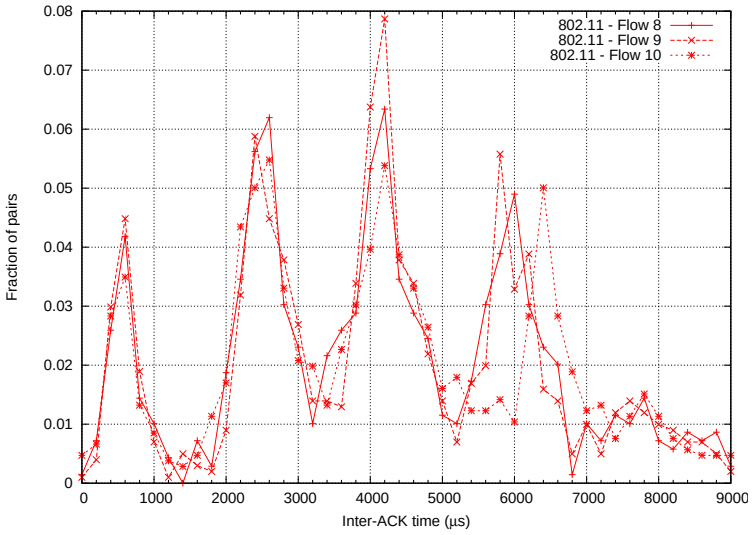
We believe this to be the behavior of a client connected to a 1Gbps Ethernet. The transmission times for TCP data and ACK segments are respectively, $12.304\mu s$ and $0.688\mu s$. Assuming that this is the bottleneck link, following Equation 5.4, we would expect a single mode in the inter-ACK distribution at approximately $25.296\mu s$.

Two curves are present in Figure 6.17, one using the same threshold used to identify ACK-pairs as the one described in Section 5.4.2, i.e. $500\mu s$, and one using a smaller threshold of $50\mu s$. In both curves most of the inter-ACK times are on the first bin (i.e. are smaller than $25\mu s$), as expected. However a considerable amount of inter-ACK times exist in the interval $[150, 300]\mu s$, which does not fit our reasoning.

Although $500\mu s$ is a reasonable threshold for the considered technologies so far, segment transmission times are much smaller for 1Gbps connections. Thus we need a smaller threshold to discard segments that are not transmitted (nearly)



(a) $50\mu s$ bins



(b) $200\mu s$ bins

FIGURE 6.16: Inter-ACK time distribution suggesting 802.11 stations using DSSS operating at 11Mbps

back-to-back. As it can be observed in Figure 6.17, the number of inter-ACK times in the $[150, 300]\mu s$ interval is reduced when reducing the threshold used to identify ACK-pairs.

Figure 6.18 shows two TCP flows from the same client, which is in the IP range reserved to VPN connections. The presence of a single mode in the inter-ACK time distribution suggests a full-duplex connection. The position of this mode indicates a transmission rate of approximately 25Mbps (it takes roughly $1000\mu s$ to transmit two full-sized TCP segments at this speed).

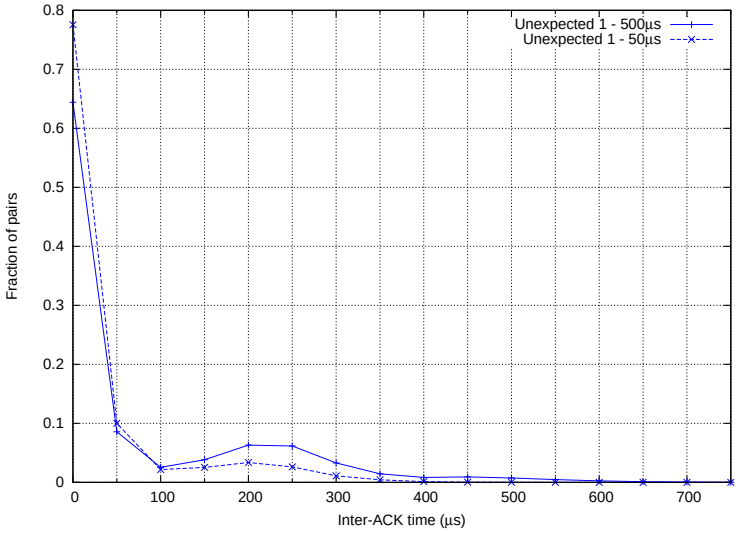


FIGURE 6.17: Inter-ACK time distribution for a possible 1Gbps Ethernet host.

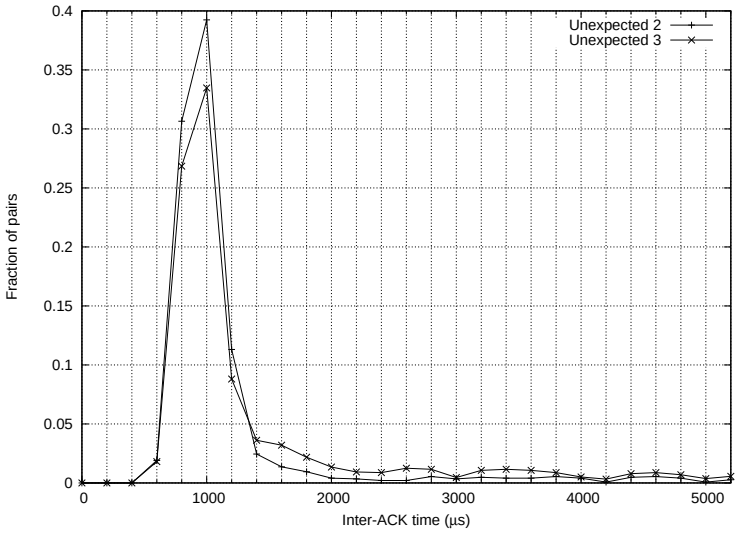


FIGURE 6.18: Inter-ACK time distribution for a VPN host.

The last unexpected behavior detected in this data set can be visualized in Figure 6.19. In this graph the inter-ACK times for two TCP flows from a client in the IP range reserved to ADSL connections are plotted. Three modes separated by roughly $800\mu s$ can be observed, however further research to infer the reasons behind this behavior is needed.

Although we are not able to draw any further conclusions from these flows, the fact that the inter-ACK distribution is similar for different flows from the same hosts suggests that our method could be extended to infer information about other types of access networks.

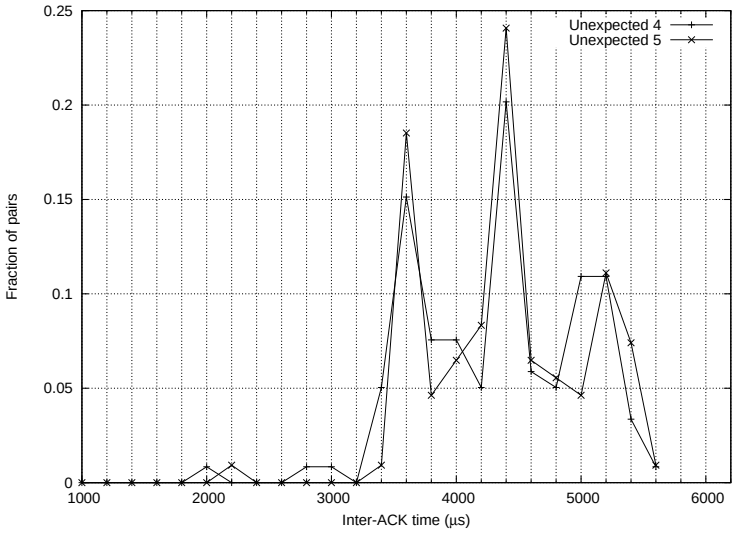


FIGURE 6.19: Inter-ACK time distribution for an ADSL host.

Chapter 7

Conclusion and Future Work

In this report we propose a novel approach for the identification of access network types. Based on basic characteristics of Ethernet and 802.11 protocols, like transmission rates of the links and duplex capabilities of the medium, we show that it is possible to distinguish TCP flows that cross these types of networks.

Our main contribution is the description of the inter-ACK time distribution of TCP flows crossing full or half-duplex links, which is used to differentiate Ethernet and 802.11 access networks. Our method can also be used to infer the transmission rates of the access network links based on the position of the modes in the inter-ACK time distribution. It is important to note though, that with this objective one should use more accurate measurement methods than the ones reported in this work, possibly using specialized hardware for network capture, such as *DAG* cards [31]. With the $125\mu s$ precision of our laboratory measurements it is not possible to capture small changes in the transmission rates.

Based on our semi-controlled laboratory experiments we see that this precision is sufficient to distinguish full-duplex Ethernet connections from the half-duplex 802.11 connections. Our tests were carried out in a variety of OS's, showing that the behavior we describe is intrinsic from these protocols and not particular to one implementation. The behavior can also be observed in traces collected in the link that connect part of the student houses to our university network, showing the applicability of our method in *real-world* environments. A downside of our method is that it requires a large number of ACK-pairs (i.e. large TCP flows) to be applied.

In this work, our tests were limited by our network infrastructure. Future work includes the realization of new experiments in different situations, e.g. using 802.11b, 802.11a and the new 802.11n technologies. Some of the unexpected results presented in our validation suggest that our method could be extended to other access network types such as ADSL. Being able to derive traffic characteristics from wireless connections from passive measurements performed in the wired network can be especially interesting for technologies such as *General Packet Radio Service* (GPRS) and *Universal Mobile Telecommunications System* (UMTS), as network operators are often not willing to share traffic information.

In this work we also do not perform the access network identification automatically, which is left for future work. One possibility to perform this task is to adapt the algorithm proposed in [20] to perform automatic detection of capacity bottlenecks. This algorithm constructs a *kernel density estimate* of the PDF of packet inter-arrivals and then scans it for modes. By adapting this algorithm to precisely detecting modes in the inter-ACK distribution it could be used to perform the access network identification. However, as the inter-ACK distribution differs considerably for 802.11 and Ethernet, it might be possible to perform the identification with fewer ACK-pairs (i.e. smaller TCP flows), where the behavior described in this work is not yet visible.

Another point that deserves further research is the impact of cross-traffic in our method. One of our basic requirements is that the monitoring point is close to the client, which decreases the amount of cross-traffic seen on the path from the client to the monitoring point. Through simulation models it would be possible to carefully study the performance of our method under different network loads, and thus, different quantities of cross-traffic.

Bibliography

[1] IEEE 802.11-2007, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 2007.

[2] D. Kotz, T. Henderson, and I. Abyzov. CRAWDAD data set dartmouth/campus (v. 2007-02-08). Downloaded from http://crawdad.cs.dartmouth.edu/dartmouth/campus, February 2007.

[3] Simpleweb / University of Twente - Traffic Measurement Data Repository. http://traces.simpleweb.org/.

[4] V. Baiamonte, K. Papagiannaki, and G. Iannaccone. Detecting 802.11 Wireless Hosts from Remote Passive Observations. In *Networking 2007 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 6th International IFIP-TC6 Networking Conference Atlanta, GA, USA, May 14-18, 2007 Proceedings*, page 356. Springer, 2007.

[5] W. Wei, S. Jaiswal, J. Kurose, and D. Towsley. Identifying 802.11 traffic from passive measurements using iterative Bayesian inference. In *Proc. IEEE INFOCOM*, 2006.

[6] W. Wei, K. Suh, B. Wang, Y. Gu, J. Kurose, and D. Towsley. Passive online rogue access point detection using sequential hypothesis testing with TCP ACK-pairs. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 365–378. ACM New York, NY, USA, 2007.

[7] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (4th Edition)*. Addison Wesley, March 2007. ISBN 0321497708. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0321497708.

[8] KK Ramakrishnan and H. Yang. The Ethernet capture effect: analysis and solution. In *Conference on Local Computer Networks*, volume 19, pages 228–228. IEEE COMPUTER SOCIETY PRESS, 1994.

[9] Jochen Schiller. *Mobile Communications*. Addison Wesley, second edition, May 2003.

[10] M. Duke, R. Braden, W. Eddy, and E. Blanton. A Roadmap for Transmission Control Protocol (TCP) Specification Documents. RFC 4614 (Informational), September 2006. URL http://www.ietf.org/rfc/rfc4614.txt.

[11] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. URL http://www.ietf.org/rfc/rfc1122.txt. Updated by RFCs 1349, 4379.

[12] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997. URL http://www.ietf.org/rfc/rfc2119.txt.

[13] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. In *In Proceedings of ACM Mobicom*, pages 107–118. ACM Press, 2002.

[14] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan. Characterizing user behavior and network performance in a public wireless lan. *SIGMETRICS Perform. Eval. Rev.*, 30(1):195–205, 2002. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/511399.511359.

[15] Felix Hernandez-Campos and Maria Papadopouli. Assessing the real impact of 802.11 wlans: A large-scale comparison of wired and wireless traffic. In *in 14th IEEE Workshop on Local and Metropolitan Area Networks, Chania*, 2005.

[16] Liang Cheng and Ivan Marsic. Fuzzy reasoning for wireless awareness. *International Journal of Wireless Information Networks*, 8:2001, 2001.

[17] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *IN PROCEEDINGS OF IEEE INFOCOM*, pages 905–914, 2001.

[18] R. Kapoor, L.J. Chen, A. Nandan, M. Gerla, and M. Y. Sanadidi. Capprobe: a simple and accurate capacity estimation technique for wired and wireless

environments. *SIGMETRICS Perform. Eval. Rev.*, 32(1):390–391, 2004. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/1012888.1005732.

[19] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. pages 123–134. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.9081.

[20] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Strauss. MultiQ: Automated Detection of Multiple Bottleneck Capacities Along a Path. In *In IMC 04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 245–250. ACM Press, 2004.

[21] T. En-Najjary and G. Urvoy-Keller. Pprate: A Passive Capacity Estimation Tool. *In practice*, 3:98.

[22] W. Wei, B. Wang, C. Zhang, J. Kurose, and D. Towsley. Classification of access network types: Ethernet wireless LAN, ADSL, cable modem or dialup? In *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, 2005.

[23] A. Wland. *Sequential Analysis*. J. Wiley & Sons, 1947.

[24] tcpdump/libpcap. http://www.tcpdump.org/.

[25] J. Postel (ed.). RFC 864: Character generator protocol. http://tools.ietf.org/html/rfc864, May 1983.

[26] Wireshark. http://wiki.wireshark.org/CaptureSetup/WLAN.

[27] Kismet. http://www.kismetwireless.net/.

[28] M. Gast. When Is 54 Not Equal to 54? A Look at 802.11a, b, and g Throughput. http://www.oreillynet.com/pub/a/wireless/2003/08/08/wireless_throughput.html.

[29] M. Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O'Reilly Media, Inc., April 2005. ISBN 0596100523.

[30] The Linux Kernel Archives. http://www.kernel.org/.

[31] DAG Network Cards. http://www.endace.com/dag-network-monitoring-cards.html.