wypy: An Extensible, Online Interference Detection Tool for Wireless Networks

by

Reza M. E. Lotun

Hon. B.Sc., The University of Toronto, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

February, 2008

Abstract

WiFi networks have become ubiquitous. However, due to the nature of the radio-wave medium, the performance of 802.11 is unpredictable and highly dependent on the environment. This problem is fundamental to 802.11's decentralized, signal-based airspace arbitration mechanism. When devices have incomplete and inconsistent channel conditions for an overlapping interference domain, their signals alone cannot ensure a fair competition for airspace. As a result, competing flows may suffer from unfair bandwidth distribution if the shared airspace is congested.

A useful tool to visualize and diagnose problematic wireless networks is the set of devices interfering with each other at a given time. We say two devices a and b *interfere* when one of two possible situations occur. First, a is able to sense b's radio signals, though not necessarily decode them, resulting in a unable to send data. Second, a and b *aren't* in radio range, but their destination devices *are*, resulting in packet collisions. We call such a set of mutually interfering devices the *interference neighbourhood*.

We present **wypy**, an *online* system which merges trace-files and produces a map of interfering devices contained within the trace. **wypy** is able to identify pairs of devices exhibiting either *hidden* or *exposed* terminal interference using a pipeline that consists of trace merging and reconstruction, filtering of simultaneously sending devices, throughput and delay signal calculations, and a test for *interference correlation*. We evaluate **wypy** using an in-lab testbed set up in known interference scenarios.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I could not have done this alone. Thanks to my supervisor Mike Feeley for stimulating meetings - I was always motivated to get things done after them. I'd also like to thank Buck Krasic for being my second reader, and for offering his ideas and advice at different stages of this work. Buck's PhD student Mike Blackstock faithfully attended our meetings and provided feedback and encouragement during some important points of this project. DSG Lab in general was a great place to work - in particular I'd like to thank Brad Penoff for providing an external perspective, and also always being open for a beer after a day at work. Finally, and most importantly, I'd like to thank Kan Cai for being my research partner - being there to bounce ideas off of, pair program and debug with, and to provide the experimental muscle that brought the code to life.

Dedication

For Kavitha and Mariam.

Chapter 1

Introduction

The future is wireless. The tethers that tie information devices to specific locations are slowly being severed. Wires connecting our peripherals such as mouse and keyboard are incrementally being replaced by wireless interconnects, as exemplified by Bluetooth. In particular, wireless LAN is clearly the foremost success in this trend. Broadband internet connections using the 802.11 family have exploded on the market to become the *de facto* home, office and school wireless networking technology of choice. The 802.11 suite of physical layers, given the various designations of *a, b, g, n*, have been dominant for a number of years, and likely will stay dominant for the forseeable future.

There are two common deployments of 802.11. The first, which we'll call *hotspot* deployment, is typical of internet cafes, or even home deployments. This usually consists of single inexpensive wireless routers, or *access points (AP)*, arrayed haphazardly in relation to each other. Each household, or small business, runs their own access point, oblivious to their neighbours and their environment, each with independent network identities and authentication schemes. Another, probably more apt, term for such deployments is *ad-hoc*.

The other common deployment of 802.11 is exemplified by large campus or business networks. Usually a large geographic space has been surveyed for optimal placement of a homogeneous set of *centrally administered* APs. Such networks present a single unified network identity and authentication scheme to its users, and are operated as extensions of an existing wired network backbone. These networks are fast becoming the norm.

The fundamental issue with any wireless technology concerns the network medium - radio waves. A new set of distinct challenges emerge when deploying a number of devices sharing airspace, most commonly manifesting in the phenomenon of *wireless interference*. The situation is undoubtedly inevitable - a huge set of devices competing with each other to broadcast over the airspace. Every device within range can "hear" the other. Even still further devices can have their own transmissions distorted by local transmissions in seemingly arbitrary ways (not at all "arbitrary" *per se*, but so

difficult to model accurately, that it might as well be). Thus wireless interference has the potential to lead to poor performance, as viewed by the user. The 802.11 *medium access control (MAC)* protocol handles only the very basic problems that wireless interference raises, as we'll show.

In this work, we'll tackle the problem of *detecting* wireless interference in a centrally administered wireless network. Ad-hoc deployments raise their own set of problems, however many of the detection ideas presented in this thesis can be translated to those deployment situations as well. In particular, we'll assume that given a centrally administered wireless network deployment, that we can either augment APs or deploy an overlay network of sensor nodes, whose job it is to listen to live traffic. We'll discuss a variation of this theme - where clients participate in the "listening" process. We'll also suggest various ways of *managing* wireless interference and its by-product, *unfairness*.

1.1 Radio Signals and Noise

Communication over a radio medium requires distinguishing a *signal* over background *noise*. The quality of a signal is usually measured using the power ratio of the signal and background nosie, which is termed the *signal to noise ratio (SNR)*. One recourse to compensate for a high noise floor is to raise the signal power, however due to 802.11 operating on the unlicensed ISM band, there is are strict regulations on how much power can be pumped into a signal. Thus, to increase capacity on a link we have two options [14]. One option is to increase the amount of spectrum to a device - however, this is unlikely as spectrum allocation is tightly controlled by government. The other option (and the one employed by successive 802.11 physical layers), is to modify the encoding on the link so more data per unit time is transmitted. Employing more dense coding has the downside that larger SNR values are needed - since receivers have to disambiguate more subtle differences in the signals - and thus overall conspire to reduce the range of communication.

To illustrate the challenges, we can consider characterizing bounds on data capacity for a physical channel, and see how SNR varies as we approach these bounds. The capacity of a communication channel is called the *Shannon limit* and as shown by the Shannon-Hartley theorem,

$$C \leq W log_2 \left(1 + \frac{S}{N}\right) \tag{1.1}$$

where C is the maximum capacity in bits per second, W is the bandwidth of the channel measured in Hz, and $\frac{S}{N}$ is the SNR. By inspection we can

see that to increase the upper bound on C linearly for W in a fixed range would necessarily impose an exponential increase on the SNR (as we'd have to differentiate between a large number of signal levels). If we solve for SNR we get,

$$\frac{S}{N} \geq 2^{\frac{C}{W}} - 1 \tag{1.2}$$

which gives us the minimum theoretical SNR to achieve data rate C, and illustrates the exponential nature of SNR for fixed ranges of W. It should be noted however that these are *theoretical* limits, and in practice the real world is far more unforgiving and messy.

In free space, radio signals decay with distance, partially due to the inverse-square decay of electro-magnetic radiation (as it spreads out in a growing sphere around the transmitter), and partially because of modulation (the encoding used on the signal). This phenomenon is called *path loss*. Higher modulation signals will suffer quicker signal decay, since they require higher SNRs. Thus, the general rule of thumb is that as the sending rate increases, the communication range decreases.

Indoors, the situation is far more complicated. In addition to straight line path loss, waves can bounce of obstacles and end up destructively interfering with each other in a phenomenon called *multipath fading*. Multipath fading is a special case of the general *inter-symbol interference (ISI)* [14], which describes the different sorts of delay a signal can incur because of the different paths it can take, and the resultant echoes that conspire to garble a signal.

1.2 A Whirlwind Tour of 802.11

In relation to the OSI stack, 802.11 has both link- (MAC) and physical-layer (PHY) components.

1.2.1 The PHY Layer

The 802.11 physical layers are given letter designations, of which the most popular are *b*, which supports data rates up to 11Mbps, and *g*, which extends these rates to 54 Mbps. The details of the various physical layer variants (a, b, g, n), aren't crucial to the presentation of this work, though a few basic notions are necessary to fully appreciate the interference problem.

Fundamentally, every device that participates in an 802.11 network is a *transceiver* - capable of both transmission, and reception of radio waves. 802.11 devices operate on the license-free *industrial, scientific, and medical*

(ISM) S-Band of 2.4-2.4835 GHz[14], which is shared by other devices such as microwave ovens, cordless phones, and Bluetooth interconnects. The ISM band which 802.11 b and g operate on, is divided into 13 channels each of width 22 MHz but spaced only 5 MHz apart, with channel 1 centered on 2412 MHz and 13 on 2472 (to which Japan adds a 14th channel 12 MHz above channel 13)[1]. However, because of constraints of how much power can be transmitted on each channel, only channels 1, 6, and 11 (in the Americas) are usable, and there may even be interference *between* channels [8].

When a device wants to transmit a frame, the medium is first tested to see if it is busy using using a combination of "physical" and "virtual" (MAC layer) carrier sensing. Physical carrier sensing is carried about through a procedure called *Clear Channel Assessment (CCA)*. The CCA module can work in 3 modes [14], where the medium can be declared to be busy if either:

1. It detects any signal energy above a certain threshold, called the *Energy Detect (ED)* threshold. The ED threshold depends on transmit power.

2. Any valid 802.11-modulated signal is detected (this does not mean the signal can be necessarily *decoded*).

3. A combination of mode 1 and 2.

Normally mode 2 is used [15], though in 802.11g Mode 3 is used, where the ED is set to a specific value [1].

Reception at the physical layer can be demonstrated by the *Physical Layer Convergence Protocol (PLCP)* header which encapsulates MAC frames (and consequently are only seen by the wireless driver). The PLCP is divided into two parts:

1. **Preamble (144 bits)** A specific bit pattern alerts the receiving circuitry of an incoming frame, and is always transmitted at 1 Mbps.

2. **PLCP Header (48 bits)** Identifies incoming packet length, the transmission rate, as well as a 16-bit CRC over the header. This can be transmitted at 1 or 2 Mbps for 802.11b or 6 Mbps for 802.11g [9].

If the PLCP header fails the CRC check, we say a **physical error** has occured.

[1] In 802.11g the ED is -76 dBm. Also, a physical layer "virtual" carrier sense is employed for interoperability with other vendors and physical layers.

1.2.2 MAC Layer

The protocol underlying the MAC is the *Distributed Coordination Function (DCF)* protocol. The algorithm used by DCF is a variant of the Ethernet MAC algorithm, *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)*. Basically stated, when a device wants to transmit a frame it runs physical carrier sensing via CCA. If it can decode the signal using the medium, the device enacts "virtual carrier sensing". Most 802.11 frames carry a duration field which tells all listening frames how long it is reserving the medium for. All devices that can decode their signal will set a counter called the *Network Allocation Vector (NAV)*, to count down for that duration.

Once the NAV counts down, the station waits for a period of time known as the *Distributed Interframe Spacing (DIFS)*. After this time has ended, a period called the *backoff window (BO)* begins. The BO window is divided into slot times, where each slot interval is medium dependent (10 μs for b). If the medium is busy, a device will enter random back off, exponentially increasing the number of slots in the BO window until it reaches a maximum of 1023 slots, at which point it remains constant for seven retransmission attempts, at which point the packet is dropped.

There are three types frames in 802.11 - DATA, CONTROL, and MANAGEMENT frames. Beacon messages, sent by APs and probe requests sent by clients in search of APs are examples of MANAGEMENT packets. When a device sends a unicast DATA frame, the receiver has to acknowledge immediately with an ACK packet, an example of a CONTROL packet. If an ACK is not received, the frame is retransmitted. Each DATA frame contains a *Frame Check Sequence (FCS)* field which is a 32 bit CRC over the whole frame - when frames fail the FCS, a situation we call a **CRC error**, those DATA frames aren't ACKed and the sender is expected to retransmit them. Frames from the same transmitter include as 12-bit monotonically increasing (0 - 4095) sequence number.

Delay	Time (μs)
DIFS	50
SIFS	10

Table 1.1: Various delays.

The MAC contains some recourse to deal with certain hidden terminal effects (described below), namely the *Request-to-Send (RTS)*, *Clear-to-send*

(CTS) messages, which can optionally be employed. When a sender wants to send a DATA frame, the sender can first sends an RTS frame with the length of time it wants to reserve the medium for, and the receiver immediately (using the *Short Interframe Spacing (SIFS)* instead of the DFS) responds with a CTS indicating their vicinity is clear for reception. Any terminals *in communication range* of either party will decode the duration time and withhold transmissions for that duration. In practice though RTS/CTS are rarely used because of the overhead, and in some cases aren't even implemented in wireless cards [11].

1.2.3 802.11 b and g Inter-operability

A wide number of legacy 802.11 b devices exist still. Both b and g use incompatible modulations - that is, during the CCA, a b device will be unable the modulation of a g signal, and will incorrectly sense the medium as idle. It is up to the access point to determine if it has any b clients. If so, the AP enables "802.11g protection mode" where each g frame is preceded by a lower-rate b-modulation compatible "CTS-to-self" frame that reserves the channel for the time needed to transmit and ACK a DATA frame.

1.3 Characterizing Interference in 802.11

1.3.1 Communication Range vs. Carrier Sense Range

We can view a single device in space to have two boundaries around it. The basic boundary is its *communication range*, the range at which devices within it can hear and decode its communications. Beyond that, a more fuzzy and irregularly shaped region can be defined - the *carrier sense (CS)* range. Other devices within this range cannot decode a signal, but can sense it. The sending device's signal has decayed so much at this point that variations within it cannot be discerned, or multipath effects begin to dominate. The end result is that any devices within this region will sense the medium as busy and enter the random backoff stage. It should be noted that both communication and CS range can be asymmetrical - it's not necessarily the case that a device B within A's communication range will be able to have it's communication decodable by A. The CS region is dynamic and irregular due to the presence of obstacles such as walls, furniture, and people.

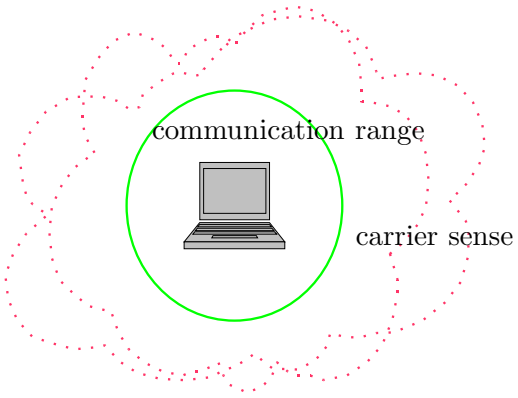

Figure 1.1: Communication vs. Carrier Sense

1.3.2 Interference Range

Consider a set of devices A, B and C arranged in space. Let A be sending to B (that is, A is in communication range of B). Let C be sending to some recipient (it is unimportant exactly where this recipient is situated), but let C be out of CS range of A (thus, A and C can be sending packets simultaneously). There is a possibility that, depending on where C is situated, its packets may corrupt (that is, collide), with packets at B. We define the *receiving interference* range as the set of all positions for C (the *interferer*) where A's (the *victim*) packets to B get corrupted by C's transmissions. Another name for those devices C are *hidden terminals*.

So far we've defined two very important topology scenarios wireless devices can be deployed in. In the CS range, devices can hear each other, and will back off when one is sending. To a device in the CS range of another device, the medium is noisy - this noise cannot be differentiated from normal background noise. If a sending devices is extremely busy (as is the case during times of network congestion), then it can certainly be the case that certain devices within CS range of others may rarely get the chance to send anything at all, leading to an unfair distribution of network bandwidth. Likewise, for receiving interference (or "hidden terminal scenarios"), corrupted packets will necessarily lead to the retransmission of those packets - thus airtime given to the victim device has been wasted, and the medium resource has been used inefficiently (in addition to the denial of service the interferer has caused to the victim). We thus term the *interference range* to

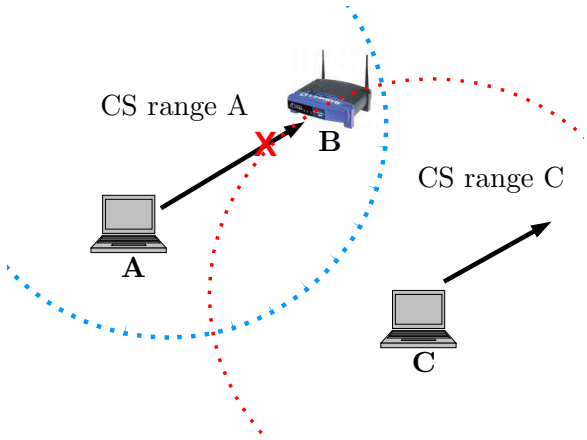

Figure 1.2: Possible manifestations of interference.

refer both to the CS range and receiving interference range. We see that in both cases one device is always interfering with the operation of the other, causing denial of service either by not letting the victim send, or corrupting the victim's packets in flight.

1. **DATA-DATA Collision:** C's transmission may generate enough noise at B to interfere with A's transmission, that is both data packets *collide*.

2. **DATA-ACK Collision:** If node C receives a DATA packet from a sender, its corresponding ACK may interfere with a Data packet at A

3. **ACK-ACK Collision:** C's ACK may interfere with the reception of A's ACK to B.

1.4 Large-Scale Wireless Networks

Given an understanding of how 802.11 operates, and the various types of interfering situations devices can be in, we turn now to a brief study of "enterprise" (large-scale campus or business, wide-area centrally administered) wireless networks and some of the challenges they pose.

In a simplified sense, a wide-are wireless LAN (WLAN) can be viewed as an extension of an existing wired backbone network, and serves as a "last-hop" for users of that network.

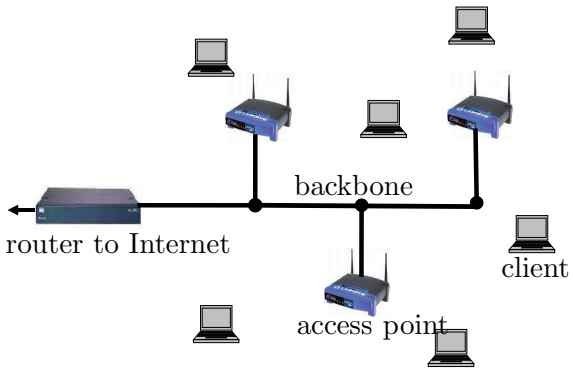

Figure 1.3: This is a typical deployment of an enterprise wireless network.

We have a natural choke-point of the router. The first observation we should make is that during some time interval the number of packets on the wired side is much less than the number of packets on during the same amount of time on the wireless side. This is because, in addition to the IP packets being pumped into the wireless airspace, there exist a number of "background noise" MANAGEMENT and CONTROL packets, in addition to retransmitted DATA packets, that occur. That is, a single TCP packet on the wired side my correspond to many packets on the wireless side, because of possible retransmissions and wireless ACKS, as is especially the case during periods of high network usage. The second observation is that of the bandwidth disparity, in the backbone (which can typically be in the Gigabit Ethernet range or greater), and that of the wireless link (being no more thant 54 Mbps).

1.4.1 Seeing Interference

As we will show, wireless interference and congestion are the culprits of unfairness. In general, knowledge of interfering devices is *useful*, since, if we factor out the vagaries of the environment, it is the main cause of any sort of performance degradation in a wireless network. So, the question is, can we "see" or detect its impact?

A number of challenges exist on this front:

1. How do we best come up with a metric to judge the impact of inter-

ference? In the hidden terminal case (receiving interference), we have an interfering device garbling the reception of packets at another device. However, in the exposed terminal (CS range) interference case, we have the victim *silenced* whenever the interferer is sending.

2. How do we differentiate interference-induced traffic from normal changes in device behaviour. For example, is the sudden silencing of throughput for a device because of a hidden terminal, or because a user closed their laptop and stopped using the network?

3. If we are to base our criteria of judging interference on network traffic, how do we best get at the data? As mentioned previously and as we will soon shown, fully describing wireless side of the network is not trivial. Because of the changing conditions of the environment and the limited locality of wireless signals, to fully "see" wireless activity in a large airspace would require a whole other network of listening sensors to record and aggregate the information they see in the airspace.

Our approach to these problems is to ensure we have as complete a packet-trace as possible of the airspace. We search for interfering devices by detecting *correlated patterns* in devices throughput and delay signature. Our definition of "signature" in this context are the representation of *throughput* and *delay* as 1-D signals in time. Operating over small-enough timescales (where "small-enough" will be quantified below), we assume we can extract meaningful correlations lasting for the life of actual network communications, and thus reduce false-positive rates of interference instead of natural changes in the network flows themselves.

We hold off on a more detailed presentation of our approach until later, however, and instead consider some applications of knowledge of interfering devices. Given a the set of mutually interfering devices, which we call the *interference domains (ID)* how can we use them?

1.4.2 Congestion and Unfairness

In general, *congestion* can be defined as the situation where the *offered load* on a link (equivalently, channel) approaches the *capacity* of a link [20]. In hidden-terminal scenarios, when the aggregate traffic load exceeds the 802.11 capacity and dominating MAC-level flows request more bandwidth than their fair share, network utilization is so high that there is not enough room in the airspace for losing flows to recover from packet loss using TCP or MAC-layer retransmissions. Similarly, in exposed-terminal scenarios, the

exposed (victim) terminal is not given fair airspace access to send its packets. Frequent packet losses and long delays cause the TCP sender in a losing flow to push fewer packets into the network, at which point the flow is doomed to lose its fair network share to others [5].

Wireless traffic has also grown faster than the substantial increases in bandwidth. Studies [17] have shown that wireless users use bandwidth-greedy applications just as they would on wired connections. As the capacity of the network backbone that connects APs to the Internet increases to Gigabit ethernet, congestion is even more likely to occur on the last-hop wireless links. Recent collected traces have shown that, even in well planned wireless networks such as hotels [19] and university buildings [8, 9], wireless users often suffer from performance degradation caused by packet collisions. As the airspace congestion problems increase, unfairness is more likely to happen [5].

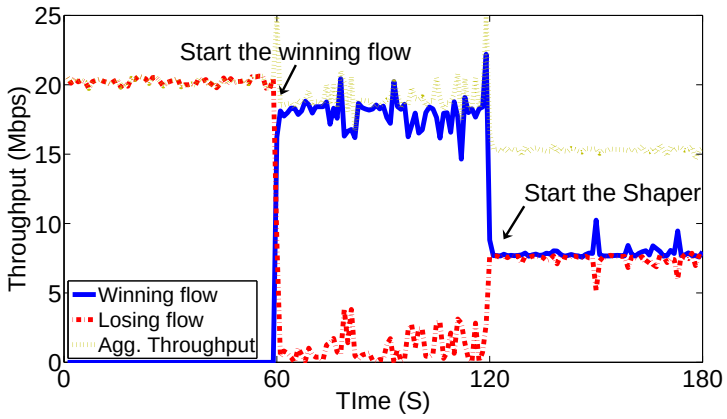

Figure 1.4: Throughput between an interferer and a victim.

1.4.3 Shaper

In [5] we presented a system which uses the knowledge of interference domains to tackle MAC-layer congestion and tackle unfairness. Shaper works by throttling traffic *above* the 802.11 MAC layer, at an upstream router. In [5] we argue that as long as fewer packets are pushed into the airspace than 802.11's capacity, standard TCP and fair queuing will together allow 802.11 to achieve fairness.

This cross-layer approach is effective because throttling the aggregated throughput below the network capacity slows winning flows and thus grants losing flows the extra airspace they need to deliver packets and recover from packet loss. Successfully delivered packets in turn open up the TCP sending window for a losing flow. Consequently, a losing flow is now able to push more packets to its receiver. Packets from all flows are queued at the upstream router and are treated equally by the fair queue management mechanism. This feed-back loop continues until a max-min fairness is achieved between flows.

Shaper operates on the set of mutually interfering APs. This is slightly different from the ID map since we cascade interfering relationships between two devices a and b one level up to interference relationships between their access points AP_a and AP_b. Given these interference domains, Shaper throttles throughput on each independently in an attempt to alleviate congestion and improve fairness for all the connected devices within each domain.

Interfering Access Points

Is the number of such interfering subsets small enough to be tried exhaustively? Suppose the area in question has n APs and suppose the area covered is large enough that we can reliably pre-partion the set of access points into j subsets. Perhaps these j subsets can be defined so that we care only about sets of access points that are, say, r meters within each other, or within 3 wireless hops of each other. So we have j subsets of n, where we can safely assume (that since we're dealing with a large enough area) $j \ll n$. Then we can have 2^j such assignments of accesspoints for each time-step.

For example, if $n = 1700$ (as is the case on the UBC campus), then if j is 10, that means we'd have 1024 sets of accesspoints we could possibly assign at some time t. Assuming a time granularity of 1 second, the space of possible mappings we'd have to search for to find the "right" one would be 86400 seconds/day to one possible subset of j access points, or 1024^{86400}. So, given this extremely detailed clustering function, we have an enormous space of possible clustering functions, namely on the order of $\sim 10^{259200}$. Even if we assume more reasonable requirements, if it turns out $j = 5$ and we're only looking for mappings on a granularity of 1 minute, we'd have $32^{1440} \sim 10^{1500}$ possible mappings! The key idea is that the set of possible clusterings is large enough to motivate various ways of approaching this problem.

1.4.4 User-Assisted Fault Diagnosis

Another application of knowledge of interference domains can be user-initiated fault diagnosis. So far we have assumed that the interference domain map is constantly being recalculated at every time step, to be used in some some central manner to take certain actions.

Instead consider the scenario where interference detection is offered as an *on-demand service*. Each AP, either augmented or with corresponding sensor node, can be taking constant measurements of the environment, but stored in some AP log repository. A user suffering from poor performance can then initiate a "help request" for diagnosis. A short period of logging can take place by the *user's device*, as part of some fault diagnosis protocol. The resultant trace can then be merged with the network side logs to gain a better picture of the user's local environment. The calculation of interference domains would be an integral part of this fault diagnoser. The information could be distilled into a variety of suggestions: "your local area is suffering from interference, please move", or "nothing wrong, check your hardware", are some Spartan ideas on the output of such a service, though more useful suggestions can undoubtedly be made.

We now turn to examining some related work in this problem space to get a better feel of the challenges, and some potential solutions, to detecting interference from network data in an enterprise network.

Chapter 2

Related Work

In much of the following reviewed work, a notion of *wireless link* is often used roughly to mean an existing communication channel between devices. Many use the notion of ETX to quantify the existence of link. ETX is presented in [12] as a path metric for multi-hop wireless networks which explicitly takes measured forward and reversed delivery ratios into account. The ETX of a link is the predicted number of data transmissions required to send a packet over the link, including retransmissions. If we consider a link defined by a sender and receiver A and B, we define the *forward delivery ratio* d_f as the measured probability that a data packet successfully arrives at r, and the *reverse delivery ratio* d_r as the probability that the ACK has been successfully received. We can view the the successful transmission of a packet over a link as the outcome of a weighted coin flip. Given that the expected probability that a transmission is $d_f d_r$, and that we are describing a Bernoulli process,

$$ETX = \frac{1}{d_f d_r} \tag{2.1}$$

An often-used definition for a wireless link, then, is having the ETX of a communication channel below some threshold.

2.1 Wireless Interference

RF Interference

In [15] the impact of external radio-frequency interference on 802.11 networks is studied. A range of devices that share the 2.4 Ghz band are noted:

- 2.4 GHz cordless phones
- Bluetooth headsets
- Zigbee (IEEE 802.15.4) embedded devices
- 2.4 GHz RFID tags

- "wireless USB" devices
- microwave ovens

The authors [15] set up an experiment consisting of an AP and client laptop using a mix of various commodity wireless NICs, and a selfish/malicious interferer. Various devices at different powers and ranges were tried as an interferer; a wireless NIC, a 2.4 GHz jammer, a Zigbee and a cordless phone. The authors measured interference effects at various parts of processing a packet at the PHY layer, on b, g and n links. They found that weak, narrow-band interferers can effectively disrupt 802.11 links, and persist even with tuning of CCA detection thresholds, sending rate and packet size. They suggest rapid channel hopping as a way to withstand RF interference.

Multi-Way Interference

In the interference works considered so far, a *single* interferer exists, and the attempted simultaneous transmissions of a *(victim, interferer)* pair is used as the basis of defining and quantifying the extent of interference. In [13], higher order interference effects were considered - that is, instead of the interferer bing a single device, it instead now can be a *set* of simultaneously sending devices. However, they found that, though when it occurs it causes significant throughput degradation, multi-way interference is rare.

Theoretical Models of Interference

The work of [18] extended [16] to computing of throughput in *multi-hop* wireless networks given node locations and ranges, as well as traffic routing information is considered. The authors use a *conflict graph* to model the effects of wireless interference. The vertices in the conflict graph correspond to active links in the network, and weighted edges between vertices correspond to the degree of noise infringement that active links inflict on each other. The authors use this graph as constraints on a linear program to solve a flow optimization problem.

The Case Against Simulation

Heavy-duty models such as those listed above are NP-hard to compute in the general case, and at best give an upper and lower bound on throughput, assuming accurate inputs and *existing* knowledge of interference. Simple rules of thumb for estimating interference, and RF properties in general have

been variously proposed in used in the literature over the years. In [22], the authors evaluate a number of models and axioms used by simulators such as:

- Signal range and interference as a function of cartesian X-Y distance
- Radio transmission area/volume is circular/spherical
- All radios have equal range
- Symmetry of radio links - that is if A can send to B then B must be able to send to A.
- The binary existence of radio links - that is, the non-existence of *signal quality*.
- Signal strength is some function of distance

They find that such simple axioms are *too* simple to model real wireless networks, and necessitate the use of real testbeds for evaluation of models.

2.2 Measuring Interference

Using Throughput

Given a *static multihop wireless network* [27] addresses the problem of estimating link interference. Specifically [27] defines interference as the aggregate throughput drop over a set of links, when each link the set is active when all others are silent, versus when all are active simultaneously. The following challenges for determining such a measure are given:

1. Physical models of radio signal propagation. As mentioned above, given the various environment and hardware specific factors is extremely difficult.
2. Brute force testing of all lists are impractical, since given n nodes there are $O(n^2)$ links and simple pair-wise comparison of links amounts to $O(n^4)$ tests.
3. Interference estimation is *not* a one time task as interference patterns can change with the environment. Thus, any interference estimation tests must be performed periodically.

In an 802.11 network, with parameters such as transmit power and data rate held fixed, a scheme to measure interference between pairs of wireless links is proposed. A *wireless link* L_{AB} from node A to B is defined using ETX.

Consider a pair of links L_{AB} and L_{CD}. For a fixed packet size, let U_{AB} be the unicast throughput of link L_{AB} when *no other links are active*, and likewise U_{CD} for L_{CD}. Let $U_{AB}^{AB,CD}$ be the unicast throughput of L_{AB} when L_{AB} and L_{CD} are active *simultaneously*.

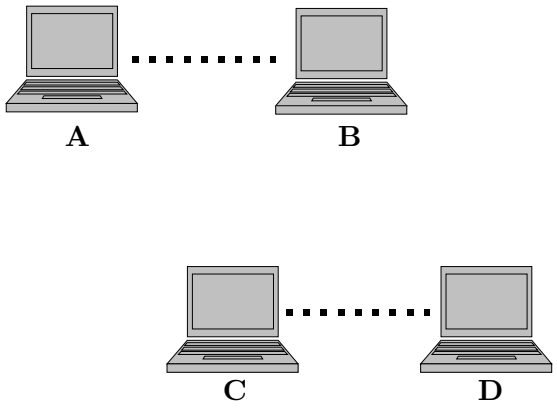

Figure 2.1: We seek to estimate the interference caused between two links.

The *Link Interference Ratio* (LIR) is defined as the aggregate throughput of the links when they are active simultaneously, to their aggregate throughput when they are active individually.

$$LIR_{AB,CD} = \frac{U_{AB}^{AB,CD} + U_{CD}^{AB,CD}}{U_{AB} + U_{CD}} \tag{2.2}$$

Some properties of LIR:

1. $LIR \in [0, 1]$
2. $LIR = 1$ implies no interference.
3. $LIR < 1$ implies interference.
4. $LIR = 0.5$ implies the aggregate of the links is halved when they are active simultaneously. This is a good indication that two nodes are within carrier sense of each other, and are backing off during each others transmission.

A method is proposed to reduce the complexity of manually estimating interference between pairs of links, using broadcast messages. Denote the send rate of a broadcasting node A as S_A. The delivery rate of A's broadcast at some node B will be denoted by R_{AB}. All pairs of nodes A, C are then set to broadcast simultaneously, their send rates denoted by S_A^{AC} and S_C^{AC}, with all nodes B denoting their receive rates as R_{AB}^{AC}. The *Broadcast Interference Ratio* (BIR) is analogous to the LIR but with defined with respect to broadcast messages instead:

$$BIR_{AB,CD} = \frac{R_{AB}^{AC} + U_{CD}^{AC}}{R_{AB} + R_{CD}} \tag{2.3}$$

Building off [27], [25] proposes the idea of an *interference map* to model interference in a wireless network. The interference map of n nodes on the same channel is a tuple (D, CS, RI), where:

- D (Channel Quality) are the $O(n^2)$ delivery ratios (percent capacity of a channel), from node i to node k *without any interference*. To collect these measurements each node broadcasts all all the other nodes record, for $O(n)$ measurements.
- CS (Carrier Sense) are the fractions of maximum capacity i can *send when j is sending at maximum capacity*. A value of $cs_{ij} = 1$ implies i and j aren't in CS range. If two nodes i and j sense each other and share the medium completely $cs_{ij} = cs_{ji} = 0.5$. An example of an exposed sender could be $cs_{ij} = 1$ and $cs_{ji} = 0.5$.
- RI (Receiving Interference) are the *delivery ratios* from i to k in the case of an interferer j. This is equivalent to the conflict graph [18] described earlier. To collect these measurements, nodes i and j broadcast simultaneously and all other nodes k record, for $O(n^2)$ measurements.

The effects of multiple interferers were shown in 802.11a networks to be independent in that their effects could be measured independently in isolation of one another to predict their cumulative effect on a link. The major assumptions made in [27] and [25] are constant bit rate traffic, complete control over all nodes in the network, and the ability to effectively shut down the network to initiate measurement phases.

Combining Measurements with RF Models

Noting that, though measurements using real hardware in real environments are more accurate, they're not necessarily reproduceable, are time-

consuming, and not always amenable to generalization, [28] combining measurements and a model of RF profile of the environment. The authors use measurements of *received signal strength indicators (RSSI)* values to seed a *signal to interference-plus-noise ration (SINR)* RF model, which estimates packet delivery probability as a function of the RF interference.

The 802.11 standard defines the RSSI value as an internal 1 byte value (thus, being able to differentiate between 256 signal levels) that measures the RF energy during packet reception as seen by a receiver NIC [3]. The drawbacks are that the definition of such an interal RSSI value is optional, and if implemented each vendor can implement it in a different manner. Specifically, the measurements they need for n nodes are the $O(n^2)$ RSSI values and pairwise packet delivery counts − as each nodes broadcasts and all others record.

Using two variants of their RSSI-seeded SINR model, *a receiver model* for packet reception in the presence of an interferer, and a *carrier-sense model* for modelling deferrals to another sender. Evaluation on 802.11 a and b testbeds showed their model to hold predictive power for packet reception probabilities. Both [21, 35] extend [28]to an arbitrary number of interferers using a model of the MAC as discrete state Markov chain, driven by deferral and reception PHY models, again seeded by $O(n)$ measurements.

2.3 A Unified Link-layer View

There are a number of challenges to obtaining a complete link-layer view of the wireless side of the network [9, 23, 30, 34]. The general problem is, given a number of devices communicating in some geographic area, how do we capture a complete link-layer view of the airspace. The requirements tacitly imply some sort of synchronized time scale amongst a set of monitoring devices. Because of the RF medium, signal propagation cannot be predicted so to guarantee near total coverage of every sent packet the problem of *where to place monitor nodes* also exists. Also, how can we be sure we've seen every packet? The wireless side injects its own "background noise" of MANAGEMENT and CONTROL packets, the most important being retransmitted DATA packets, and means to *quantify* coverage are also important.

2.3.1 Trace Merging

Assuming n monitors exist in a wireless network, each in promiscuous mode with their own trace file and *their own clock*, [23, 30, 34] tackle the problem in an offline mannner, while [9] presents a method to do it *online*. The

process of merging involves a *bootstrap synchronization phase*, where given no knowledge of the monitors clocks, their initial offset need be obtained via the existence of *reference packets* seen by both monitors. A further complication that 802.11 operates on μs granularity, so the merge process is particularly sensitive to clock drift.

A means of continuous resynchronization is required. In [34] linear regression is performed on reference packets on a pair of trace files over the whole time-frame, and clock drift is then inferred as a linear phenomenon. Noting that clock skew manifests itself in a non-linear fashion, [23] divide the merge files into sections, and do linear regression for each section, thus arriving at a piece-wise linear approximation to the unknown non-linear clock skew. Merging over a large set of trace files in real-time, [9] employs a similar method, but augments it with direct modelling of clock skew for each monitor.

Time Granularity

Given two reference frames r_1 and r_2 from two sniffers, we define the *time synchronization error*, as the average timestamp difference between r_1 and r_2 (to an unknown "global clock"). To place an upper bound on this error requires us to quantify the minimum possible time difference between two valid 802.11 frames. The minimum delay is

$$d_{min} = pr_d + SIFS + pkt_{min} = 192\mu s + 10\mu s + 10\mu s = 212\mu s \qquad (2.4)$$

Where pr_d is the time to delay to transmit the preamble, $SIFS$ is the short interframe spacing, and pkt_{min} is the time it takes to transmit the smallest possible packet (a 14 byte ACK). So, given a $212\mu s$ delay, we see that the upper bound on the time synchronization error is $\frac{d_{min}}{2}$ or $106\mu s$.

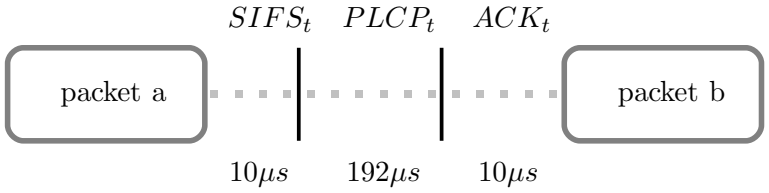

Figure 2.2: The largest possible time synchronization we can tolerate, as recorded in a packet trace.

Unique Packets

A remarkable number of 802.11 packets are impossible to disambiguate [9, 23]. For example, a retransmitted packet heard by multiple monitors that is continually rebroadcast will look identical during the merge process, and thus correct time synchronization of the traces are the only way to reliably disambiguate them [23].

During the *bootstrap synchronization* process, where initial clock skew is determined without any prior knowledge, a method of determining proper unique packets is essential. Starting with an upper bound of the possible clock variation amongst the clocks on the different trace files, an initial set of common packets need to be found amongst a set of highly probably unique packets. Certainty cannot be guaranteed for the following reasons:

1. The sequence numbers of non-retransmitted packets goes from 0 to 4095, after which it is reset. If the upper bound on the clock skew is high, we may scan forward too far in time in a trace file and mistakenly mark the *next* phase of sequence numbering as an "identical packet"

2. Beacon packets are one type of reference packets used in the literature [34] because of their unique 64 bit timestamp files counting up from an epoch defined as when the access points they belong to are first turned on. However, in [23] it was shown that a high number of access points were power cycled and thus resetting their timestamp values.

2.3.2 Inferring Missing Packets

Even given a reasonable coverage of monitors and an accurate merge, it is still very likely that packets could be missing from the trace file. However, given that a packet trace is essentially a record of *transactions* or negotiations, missing packets can be inferred within certain sets of packets. For example, a DATA is always followed by an ACK in a successful transmission, or other retransmissions, an RTS is followed by a CTS, etc. In addition to these high-level protocol interactions, sequence numbers on DATA packets can tell us directly which packets are most likely missing (given that the trace is off sufficient granularity to capture *at least a few* numbered packets within a 0-4095 range). To implement link-layer protocol inference, [9] uses a finite state machine of packet interactions to identify transmission attempts and exchanges to infer missing packets, supplementing it with information at the transport layer using TCP sequence numbers and ACKS − though in general transport level information may not be available due to

the use of encryption. In [23] the inference problem is treated as a *language recognition task*. Packet exchanges are treated as sentences with certain structure (defined by the protocol). The trace then can be viewed as interleaved partial sentences in the language. Treating the language as regular allows the authors to leverage regular expressions and finite state machines, and the authors use these FSM to generate missing segments of some sentences. Analyzing the effectiveness of their inference component versus the use of additional monitors, they find that inference and additional coverage are complimentary − good coverage of monitors is needed, but there are diminishing returns after covering a dense set, and inference is effective at recovering missed packets.

2.4 Large-Scale Analyses

Distributed Fault Diagnosis

In [7], a distributed fault-diagnosis system called WifiProfiler is presented. The system comprised an "information plane" orthogonal to the actual 802.11 network devices were connected. Each device runs a sensor program that scans the host machine to detect any problems − these problems can run the gamut from detecting network connectivity problems to misconfiguration issues. The sensors issue a help request when a problem is detected, and a P2P ad-hoc help network is formed with its peers to pool and aggregate information to perform automated diagnosis.

In [2, 6] a system called DAIR is presented which leveraged the stationary PCs in an office building by equipping them with wireless USB dongles. These dongles perform light-weight RF measurements of the airspace, as well as other measurements of the wired side. The measurements are forwarded to a central inference engine elsewhere on the network for further computation, and was used to detect rogue APs and perform localization of mobile clients.

In [31] a system for diagnosing and and differentiating physical-layer anomalies using wirelss sniffers is presented. Four PHY layer anomalies are detected: *i*) noise levels *ii*) hidden terminals *iii*) capture effects and *iv*) long-term signal strength fluctuations. The *capture effect* refers to the phenomenon where, given two incoming simultaneous wireless signals, a receiver locks on and captures the stronger of the two signals. This can be a problem if because of distance, RF fluctuation, or vary signal strength, a node not associated with a receiver may have its transmission stomp out another who is genuinely associated with that receiver. Each client logs

air traffic, and the data is sent to a central server for analysis, though each independent view is not merged together to get a global view of the airspace. Simultaneous transmissions within an AP domain is used as the basis to detect and differentiate hidden terminals and the capture effect, and long term signal strength changes between pairs of APs are compared using a measure of linear correlation (Pearson's correlation score).

Performance Measures

The Jigsaw [8, 9] system at UCSD is the only large-scale monitoring and merge system we are aware of. The goal of the project is to systematically capture all link-layer packets over a large area (the computer science building) using a large number of sniffers (192) simultaneously monitoring three channels. An online merge phase followed by link-layer reconstruction is performed, and the resultant trace [10] are stored offline for further analyses. Given the merge trace file, the Jigsaw authors performed a number of interesting analyses. Over the course of the day 47% of recorded frames were found to be physical (PLCP 16-bit CRC failures) or CRC errors (MAC frame 32-bit CRC failures). Broadcast traffic was found to consume 10% of the airtime, since broadcast packets are sent out at the lowest rate and thus make inefficient use fo the airtime.

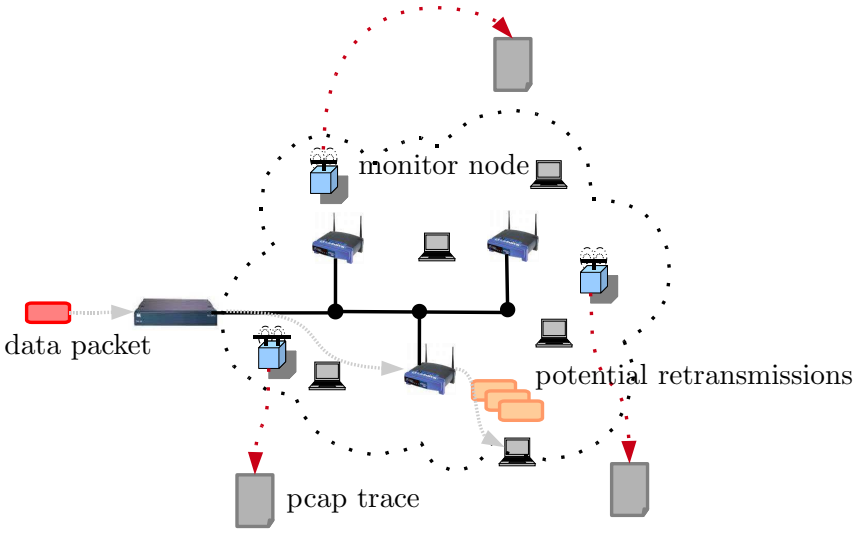

Figure 2.3: An overlaid monitor network over a wireless domain

The authors of [9] also attempts to *quantify* how often receiving interference (hidden terminals) occur. Given a sender s and a receiver r. Let:

- n be the number of transmissions from s to r
- n_0 be the number of transmissions from s to r where no simultaneous node is sending
- n_0^l be the n_0 transmissions that are lost. By lost we mean that no corresponding ACK was observed.
- n_x be the number of transmissions from s to r *with* a simultaneous transmission
- n_x^l be the number of n_x transmissions that are lost

Assuming that the trace completely sees all packets, notice that the fraction (or probability) of $T = \frac{n_x^l}{n_x}$ gives the percentage packets that *may* have been lost to hidden terminal effects (or some unknown environmental cause). Also notice $B = \frac{n_0^l}{n_0}$ gives a rough "background loss probability" caused by the environment. However, we can say that the probability of loss due to *receiving interference only* is $\frac{T-B}{1-B}$. These are rough values that are only asymptotically correct, so the authors chose 536 (s, r) pairs (82% of all pairs in the trace) that exchanged at least 100 packets, they found that 88% of the pairs experience lost packets due to hidden terminal effects.

To complement the hidden terminal interference loss show above, [23, 24] presents a method of determining the number of contenders of the medium, using trace data. Contention time is defined as the time when a device's MAC layer receives a packet to to the time of successful transmission (if it's a DATA packet, "successful" means the corresponding ACK is also received). The scan a trace in reverse chronological order and maintain a set of *contender* stations along with their corresponding *idle-wait-time* − the amount of idle time they must have waited to acquire the channel before their last observed transmission. For example, the time between a device's DATA and DATA-retry packets give an indication of the amount of time a device was waiting to acquire the channel. Time delays between packets from stations within the set are computed, and stations are removed from the set when their delay difference to the idle-wait-time has reached zero, and new devices are added to the set when DATA-retry packets are seen. Original packets (DATA without the retry bit), reset the idle-wait-time for that station. Thus at any given instant, the set of mutually contending devices can be enumerated, giving some indication of carrier sense interference.

Chapter 3

The *wypy* Framework

Given large datasets of trace data, the ability to process and analyze the data in easy and extensible ways is of paramount importance for the development of the detection schemes outlined later. Starting with packet-level logs and a vague idea of patterns contained within the logs, how do we *start* to test our ideas, and models?

The *wypy* ("why pie") system was designed foremost to aid in the experimental process. The name is derived from from *why* – reflecting the primary goal to detect interference – and *Python* – reflecting the language it was written in (not to mention that it rhymes with *Wifi*). *wypy* is a lightweight framework comprising selective packet parsing, tagging, merging of tracefiles, and a general pipelined system where analysis components called *analyzers* can be chained together to promote code reuse and modularity.

3.1 Architecture

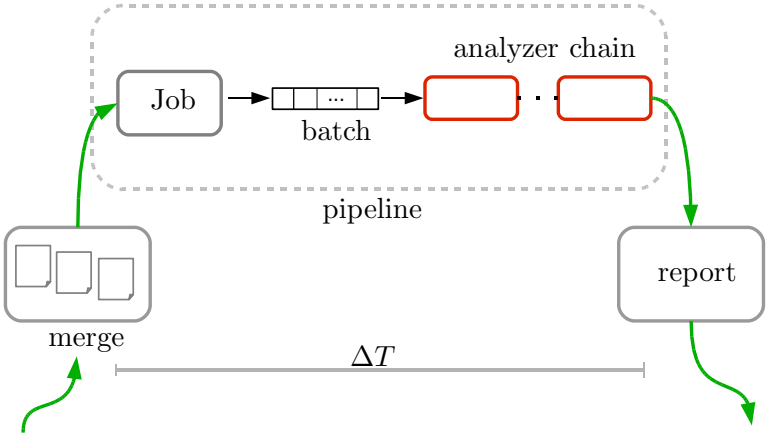

Figure 3.1: The *wypy* architecture

In Figure 3.1 we see an overview of the *wypy* system. The major steps corresponding to *i*) input of packet level traces, *ii*) optional merging of trace files to time synchronize them and remove duplicates, *iii*) input into a pipeline process which buffers packets by *timebucket* ΔT and feeds this buffer through a chain of analyzers at which *iv*) information is aggregated and reported.

3.1.1 Input

The the input to *wypy* involves one or more sniffer or monitor nodes running in promiscuous mode, which collect and record all 802.11 frames they are able to decode in the local airspace. *wypy* works on the level of pcap [32] format tracefiles. The pcap format is simple − a flat, binary file with two main parts: a file header, and a sequence of records (corresponding to captured frames). The purpose of the file header is to establish byte ordering, state the type of link layer frame present in the file, and declare any global time zone offsets as used by the timestamps of the contained records. A 16 byte header precedes every record giving a timestamp for the recorded frame and its length in the trace file.

Field	Bytes	Description
timestamp	8	4 byte seconds, 4 byte μs
snaplen	4	Length of packet in tracefile
caplen	4	Length of actual packet (in the air)

Table 3.1: Pcap record structure.

Directly following the contents of the pcap header is the link layer header. In practice, the two most common headers are the Prism (144 bytes) and RadioTap (variable size, at minimum 26 byte) variants. These physical layer headers are generated by wireless NICs when they are in promiscuous mode, and contain information such as channel, rate, signal quality, and CRC errors (if exposed).

3.1.2 Parsing

Beyond the link layer frame headers the actual 802.11 packet header begins, and each packet is passed to the variable packet parser. "Variable" here simply means that the degree of parsing for 802.11 packets can be determined. Its format is quite complex, and depending on the application, not

every part of the header need be parsed.

An 802.11 packet comes in three flavours (*types*) − DATA, MANAGEMENT, and CONTROL. Each type has a 8-16 packet *subtypes*. Depending on their type 802.11 can very greatly in size, and format. This is by careful design − for example ACK packets (a type of CONTROL packet), need to be extremely short to decrease their probability of being corrupted while in the air.

Type	Description
DATA	Regular data traffic
CONTROL	ACKs, RTS, CTS
MANAGEMENT	beacons, authentication, probing for APs

Table 3.2: Types of 802.11 packets and some representative subtypes.

In *wypy* packets are viewed as concatenated blocks of bytes. This analogy is especially useful for protocols like 802.11 that have different formats depending on their type. To make this notion concreate, the `Packet` object of *wypy* subclasses Python's built-in `list` type. Each protocol header then is subclassed from the *wypy* `HeaderBlock` object, which simply contains attributes and information on how to parse them from a block of bytes. Protocols such as TCP would be a single headerblock, whereas 802.11 comprises 7 different `HeaderBlock`'s. A `Packet` is created by instantiating it (really just an empty list, with some extra scaffolding), passing in a sequence of anonymous bytes representing the packet payload, and `append`-ing various `HeaderBlock`'s. The `append` triggers parsing behind the scenes, making it easy to use and write parses for new protocols.

Every 802.11 packet begins with a 2 byte FrameControl field, which states the packet type and subtype, whether the packet has been retried, and what mode the device that sent it is operating in (ad hoc vs. managed vs. a bridge between networks, etc.). Other key fields are the SequenceControl (2 bytes) field, which gives sequence numbers for data packets, DurationID (2 bytes) field which states the time at which listening stations should back off for, and address (6 byte ethernet style) fields giving sender, destination and SSID (network name) information.

Packet Tagging

Given that there are a variety of types of packets that can be had from a 802.11 session, a quick and transparent way of determining attributes on

Frame Subtype	Fields
DATA	FrameControl, DurationID, 3 Addresses, SequenceControl
ACK	FrameControl, DurationID, 1 Address (destination)
MANAGEMENT	FrameControl, DurationID, 1 Address, SequenceControl
RTS	FrameControl, DurationID, 2 Addresses

Table 3.3: A example of how field structure can vary for a few representative 802.11 packets.

packets is needed. This is most useful when testing and setting attributes during analyzer program. For example, if a packet `p` was a corrupted (via CRC-error) data packet, in *wypy* the simple operation of:

```
p += "data"
p += "corrupted"
```

"tags" the packets with various informational attributes. Later processing steps can check for tags, take appropriate steps or set their own tags. A checking of state can be performed via:

```
if "corrupted" in p:
    ...do something...
```

3.1.3 Merging

The merging component takes n trace files representing a set of local monitors in some active airspace. Currently merging all n files is performed pairwise for simplicity, similar to the "waterfall process" as done in [23], though *wypy*'s merge can be easily be extended to an n-way online merge similar to that in [9, 30].

The algorithm is divided into a number of phases. The run-time analysis will be based on n trace files with $O(k)$ records each:

1. **bootstrap synchronization** - Since the clocks which produced the trace files are assumed to be wildly varying at worst, and at best synced with NTP (which provides millisecond resolution), a suitable *reference packet* heard by both monitors must be searched for. The time difference between two reference packets represents the initial time offset between the two clocks. An arbitrary trace file clock is used as a global time. As mentioned earlier, not all 802.11 packets are

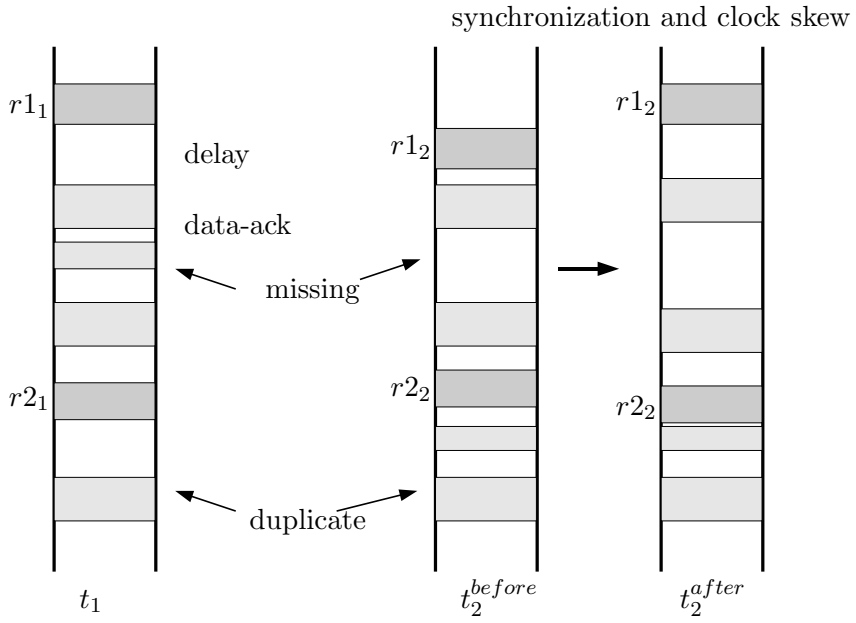

Figure 3.2: An overview of merging. Two tracefiles are presented for simplicity, t_1 is the master trace file. Trace file t_2 contains the same reference packets, so the region in between both reference packets are linearly scaled to match the masters. Notice we also have to merge packets that may be unique to both traces, and also make sure not to include duplicates.

suitable to be used as reference packets, since they are not distinguishable from each other. *wypy* uses DATA packets *without* the retransmit bit set, and uses the Address fields and SequenceControl information as unique markers. Since the clocks may not necessarily be synced via NTP (or at least, it is prudent to have the merge process robust against this possibility), the issue of wrapped sequence numbers need be dealt with − that is, the possibility that an earlier "actual-time" version of the packet wasn't recorded in one trace, but later on in time a "wrapped around" version of the packet was encountered and *that* is used as a reference packet. *wypy* takes care of this by searching for a set of reference packets, and choosing the minimum time offset for its initial offset.

2. **merging** - After an initial offset is determined (assuming no clock drift), all timestamps on the trace files can be corrected by the additive

offset, and the process can be treated as merging sorted lists, which can be done in $O(k)$ time. For merging n trace files, the heads of each trace file can be placed in a priority queue, resulting in a total time to merge of $O(k \log n)$.

3. **resynchronizing** - To deal with inevitable clock drift, between two trace files *wypy* finds pairs of reference packets, and linearly interpolates ("stretches") the time spans of these sections so that they match. This resynchronization procedure is invoked via callbacks triggered by positions in the file corresponding to the last round of resychronization plus some offset set heuristically (to avoid unneeded constant resynchronization between monitors that have a large overlap between what they hear). The total process results in a very fine grained piece-wise linear approximation to the unknown non-linear clock skew.

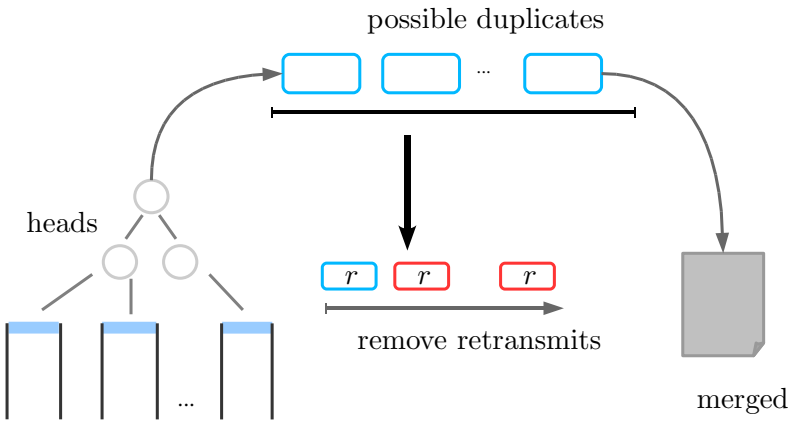

Figure 3.3: Assuming the clocks are reliably synced, we must merge the heads together, checking for duplicates, and making sure not to remove *too man* retransmitted packets.

3.1.4 Job and State Management

The job and state management functionality of *wypy* serve to buffer incoming merged and parsed packets into a unit of time called a *timebucket* ΔT. This is the large-scale granularity that later analyzers will operate on, or where decisions of interference will be made. Later, we'll discuss how best

to choose this timebucket. The duration can be for any amount of time, though it's usually set on the time granularity of seconds. The *wypy* `Job` is instantiated with with a list of analyzers and exports a batch of packets that share the same timebucket to be sequentially passed through the analyzer chain. Each analyzer subclasses *wypy*'s `Analyzer` class, which defines a clear API for interacting with the *wypy* pipeline − a packet batch is processed through an `process` method which contains two subsystems defined below.

3.1.5 Analyzer Chain

The analyzer chain is simply a list of instantiated analyzer objects. The idea is that each analyzer continually works on a batch of packets and computes various statistics or values, either for immediate visual feedback or logging purposes, or as a means of *preprocessing* the packet batch. Analyzers have the option of working either on timebucket granularity, or subdivisions of a timebucket called *time windows*. For example, a throughput analyzer would extract all devices flows and calculate throughput over some timewindow, say 200 ms, and add the *throughput time series* signal as an attributed to the batch to by used by an downstream analyzer, such as the *correlation coefficient* analyzer described later. This allows for a highly modular analyses, as a specific analyzer would have a set of dependencies on upstream analyzers and need not recalculate everything itself. For example, a type of analyzer callled a `Filter` processes the packet batch and filters out simultaneous sending devices, and annotes the batch with a set of MAC addresses, which can be used by downstream analyzers to calculate things like throughput more efficiently.

Every analyzer has its own state manager which issues a `State` object each timewindow. The `State` object is just a container object for attributes that are calculated each timewindow. In the throughput example each state would hold the number of bytes transmitted by a device during the time window. An end of timewindow `aggregate` method is called on the resultant linst of state objects and a final calculations over the whole timewindow are made. In the throughput case it may just be culling out the information in each timewindow and normalizing it.

3.2 Implementation

The *wypy* system was written in the Python [33] programming language − a high level, dynamic interpreted language, making extensive use of the SciPy and NumPy [26] projects, which provide fast resizeable multidimensional

arrays and math functions. The advantage of using Python are speed of development and code clarity. It excels as a prototyping language, has a vast standard library, and integrates well with the underlying operating system.

The disadvantage of using Python are speed and memory usage. Since the language is interpreted it suffers a distinct performance disadvantage compared to compiled languages such as C. Its memory is garbage collected, and its internal datastructures are primarily based on hashtables (dict and set), which can lead to increased in memory usage.

Though the initial reaction when dealing with performance constraints such as these is to reimplement it in a faster, compiled language, the situation is not so hopeless. Being careful to avoid the dangers of premature optimization, there are a number of standard methods to improving speed in Python. The laguage ships with a number of profilers that can be used to find hotspots within the code-base to be optimized. There also exist a number add on packages and modules which are designed for Python optimization tasks. Though the aim of this work is to provide an extensible toolkit for quick iterations of data analysis, some optimization steps were taken. As a first pass, we tried **psyco** ("Python Specializing Compiler") [29], which is a "Just In Time" (JIT) extension module to Python. The psyco compiler generates machine code at runtime, and does real-time profiling to find hotspots. With very little tuning "out of the box", psyco leant 30-50% improvements to running time, with minor increase of memory usage.

3.3 Analyzers

To give a flavour of the type of processing done by analyzers, a few exemplar ones will be overviewed, and we'll show how to chain them together to reuse code and potentially boost efficiency.

Throughput

This is perhaps the simplest analyzer. Our `analyze` step just creates a dictionary per state of bytes sent from a sender, receiver pair. The `aggregate` state simply produces a timeseries of these throughputs (putting a 0 in the time series if it wasn't seen during a time window, as shown below.

```
def analyze(packets_tw, current_state):
    """ Given the packets falling in our
```

```
        timewindow, keep track of number of
        bytes sent for a sender, receiver pair
        """
        for packet in packets_tw:
            pair = (packet.sender, packet.destination)
            current_state.counts[pair] += len(packet)

    def aggregate(states):
        """ Given a list of states (one per
        timewindow), aggregate statistics
        over the whole time bucket (largest
        scale time granularity)
        """
        final_throughputs = {}
        for state in states:
            for pair, throughput in state.counts:
                final_throughputs[state] = throughput \
                                or 0 if pair not seen
```

Simultaneous Senders

The set of simultaneously sending devices provides useful information about the airspace. For example in the hidden terminal interference scenario, all interfering terminals to some victim *will* be in its simultaneous sender set. In the case of the exposed sender (CS range) scenario, where only one device is active at one time, any exposed devices *can't* be in the simultaneous sending set.

Given a set of packets, how do we determine the simultaneous senders. This is most easily done with any packet that contains a DurationID field, such as DATA packets. When a DATA packet is sent, this implies the sender has direct control of the local airspace − that is, it won the contention period. The time spent in control of the airspace is determined by the size of the transmitted frame, and the rate it which is sent at. Within DATA packets, a NAV is set for the duration of the packet's ACK, a duration which all listening devices must back off for.

Thus, for each packet in our set, we can take the packet's time stamp as the time just before the packet was sent (we only care about the relative time in between packets), and knowing its size and rate (from the link layer header), we can calculate how long it was in the air for. Adding on to this

time, gives a total duration, which can be checked against other durations for overlap.

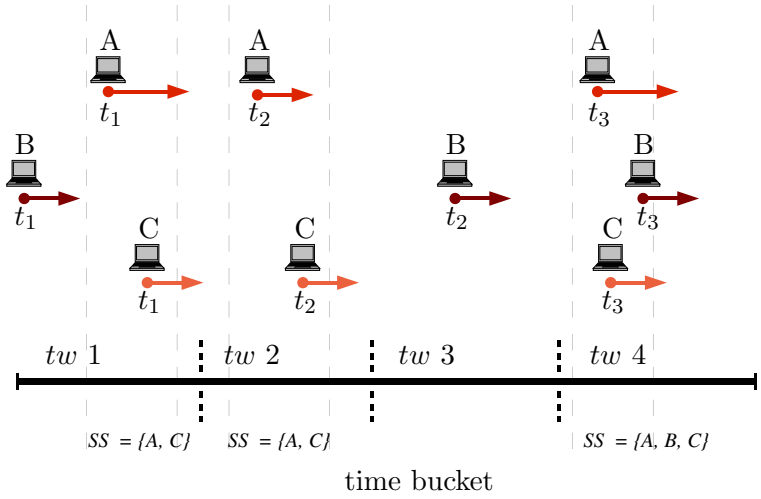

Figure 3.4: A number of devices active within sub time windows of a time bucket. A device is active from the beginning of their timestamp, to how long it would have taken to transmit their packet, plus their duration.

```
def analyze(packets_tw, current_state):
    """ Given the packets falling in
    our timewindow, keep track of which
    devices are overlapping
    """
    for packet in packets_tw:
        time_in_air = pkt.timestamp + pkt.nav + \
                      + pkt.length / pkt.rate
        check for overlaps in all other devices

def aggregate(states):
    """ Given a list of states (one per
    timewindow), aggregate statistics
    over the whole time bucket (largest
    scale time granularity)
    """
```

```
reverse dictioanry for each state to produce
for each device, the set of simultaneous senders
(optionally filter by degree - for example, only
keep simultaneous senders that sent more than 50%
of the time)
```

Chapter 4

Detecting Interference

In an interfering scenario, when one device sends, the other device is unable to get its packets through to its recipient. Let's assume that both devices are streaming traffic, and making full use of the available airspace, for argument's sake. If two devices a *victim* and a *interferer* are both sending the interferer always wins and "drowns out" the victim.

Thus, the throughput of the two devices follow a pattern − when the interferer's throughput goes up, the victim's throughput goes down. Likewise when the interferer's throughput goes down the victim's throughput goes up. Any deviation by either device will result in the other device immediately picking up the slack.

Our detection method seeks to exploit this throughput correlation by observing it in live network data.

4.0.1 Correlation

If we let the throughput for one device be X and the throughput for the other be Y, we will make an assumption that fluctuations in throughput between pairs of devices *considered in isolation from others* is a *linear relationship*. What this means is that:

$$\Delta Y = \alpha \Delta X + b \tag{4.1}$$

That is throughput changes between X and Y are linear. In fact, for interfering scenarios we say that the two throughputs are *negatively correlated* - "one goes up, the other goes down".

The correlation coefficient, given by:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{4.2}$$

$$= \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \sigma_Y} \tag{4.3}$$

$$= \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E^2(X)}\sqrt{E(Y^2) - E^2(Y)}} \tag{4.4}$$

and is defined to give a value in $[-1, 1]$, where a 1 imples strong positive linear correlation, a 0 implies no linear correlation, and a -1 implies strong negative linear correlation.

Any time we calculate the correlation coefficient (CC) we do so on the samples of throughput, represented by a timeseries, as given below:

$$r_{xy} = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} \tag{4.5}$$

$$= \frac{\Sigma x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} \tag{4.6}$$

$$= \frac{n\Sigma x_i y_i - \Sigma x_i \Sigma y_i}{\sqrt{n\Sigma x_i^2 - (\Sigma x_i)^2}\sqrt{n\Sigma y_i^2 - (\Sigma y_i)^2}} \tag{4.7}$$

which is called *Pearsons' correlation*.

4.1 The Algorithm

The algorithm is as follows. Given n devices in a timebucket, for all pairs of devices find all the simultaneous senders. Consider only the set of simultaneou senders (in the hidden terminal scenario). Calculate the throughput of each device, and calculate the correlation coefficient value betweeen each pair. As a filtering step, we remove any section of a timeseries between devices that vary more than 10% of the other, that is we only consider parts that could *possibly* be caused by the other signal. This is to avoid associating external causes in a signal to a device it is being testing for interference against. If the determined correlation is below a *threshold* of -0.7 (explained below), we classify the pair as *interfering*, else we say non-interfering.

4.2 Experimental Setup

Our testbed consists of seven nodes, two LinkSys APs, two clients, and three monitor nodes in a $8m$ x $8m$ room. The wireless chipsets for each client are Ralink RT2500 1.1.0, the APs use Broadcom and the monitor nodes use Atheros Madwifi 0.9.33. We use altered antennas to attenuate signal strength, and change the network topology by adjusting transmission-power settings. The clients and monitors run Linux kernel 2.6.22 with the high-resolution timers enabled; the system clock frequency is set to 1000 Hz, and the APs run OpenWRT WhiteRussian. The transmitting rate is fixed at 54

Mbps. The autorate function is disabled so that topology does *not* change during the runs.

We generate traffic on the testbed using the "Distributed Internet Traffic Generator (D-ITG)" [4] using a gaussian distributed inter-packet time, given in ms. We use UDP traffic for the following experiments.

4.3 Experiments

We set up and test correlation values on three topologies:

1. No interference
2. Hidden Terminal Interference
3. Exposed Terminal (CS Range) Interference

   In each topology we run generate 10 minutes of streaming traffic and calculate the correlation coefficient on time buckets of 10 s. Each 10 s time interval has a throughput time series calculated on the resolution of 10 ms.

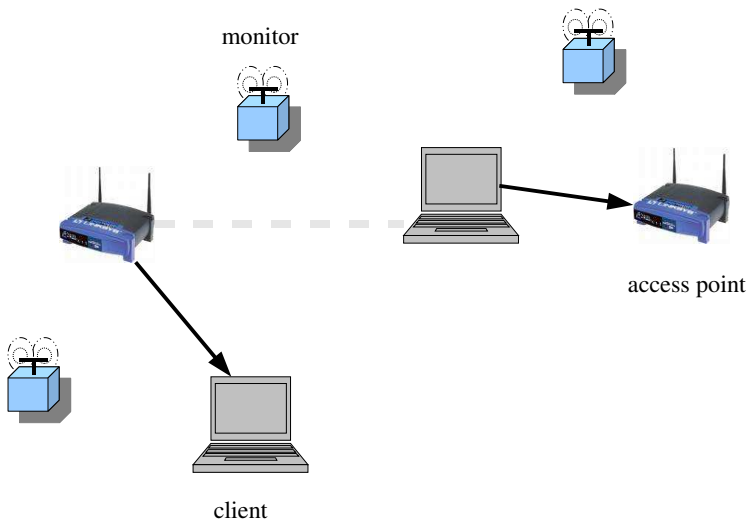

Figure 4.1: Our testbed. A hidden terminal interference scenario with strong interference (larger than -60 dBm)

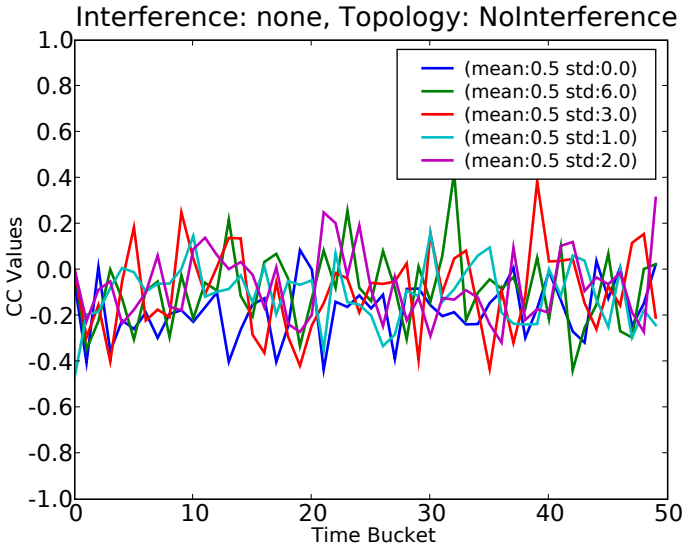

Figure 4.2: Results for normally distributed inter packet time in a no interference scenario. Correlation results closer to 0.5 imply no linear correlation.

4.4 Determining the threshold

Examining the CDF of the correlation coefficient values allow us to determine the best value to set our threshold. Ideally, we'd like to choose a threshold at the "bend" in the graph that allows us to classify most instances of interference as seen in our testbed. We can see that using a threshold of -0.7 allows us to classify more than 90% of interfering scenarios. Also note that in examining the CDF of a non-interference scenario, we generally get no correlation. This demonstrates the discriminative power of the correlation coefficient.

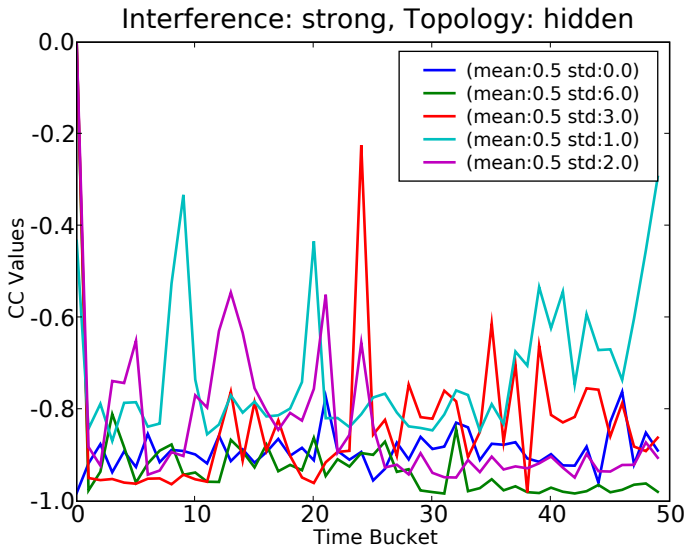

Figure 4.3: Results for normally distributed inter packet time in a strong interference scenario. Correlation results reports, closer to -1 means a strong negative correlation.

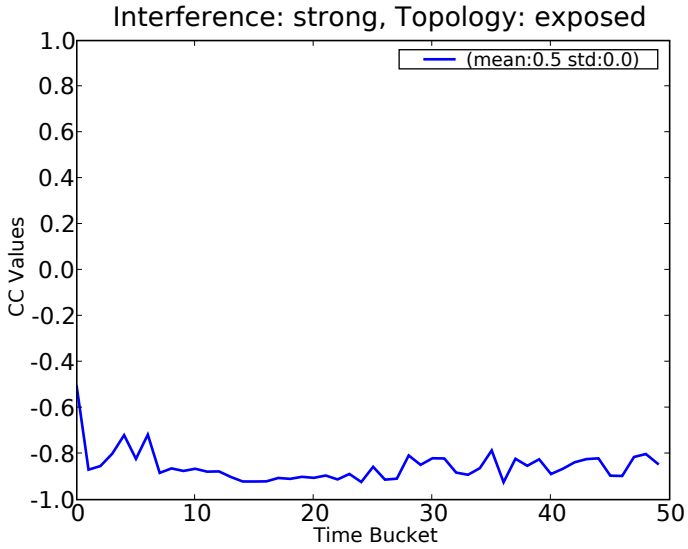

Figure 4.4: CDF for CC values in the exposed sender interference scenario.

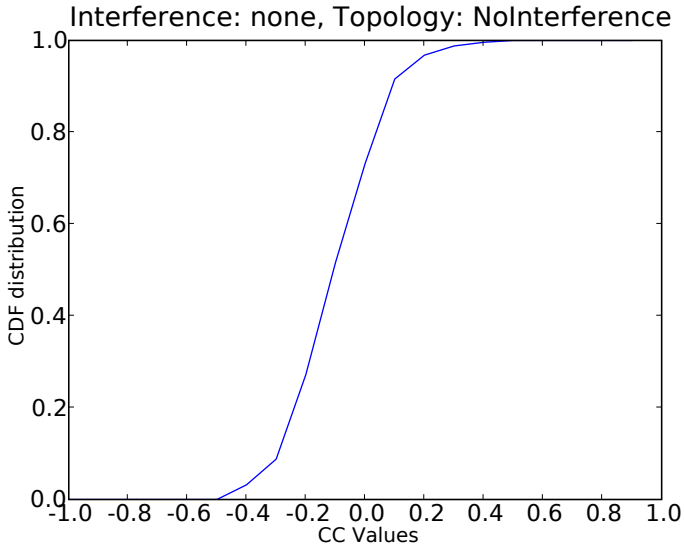

Figure 4.5: CDF for CC values in the no interference scenario.

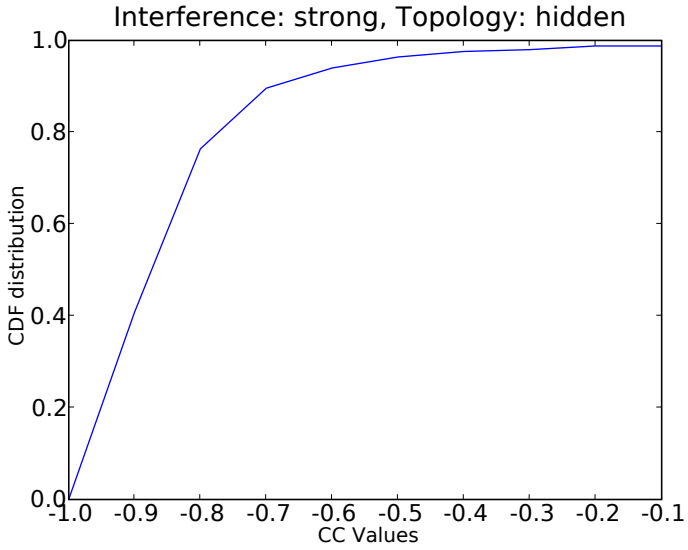

Figure 4.6: CDF for CC values in strong interference scenario (hidden terminal).

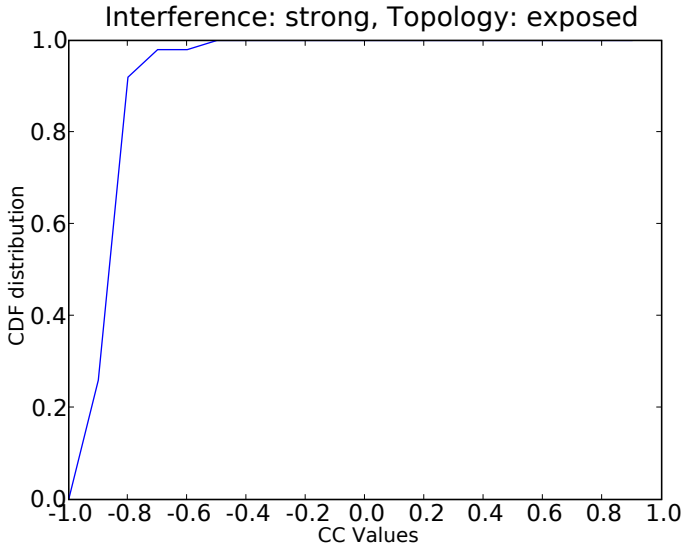

Figure 4.7: CDF for CC values in strong interference scenario (exposed terminal).

Chapter 5

Conclusion and Future Work

We conclude that using correlation of throughputs to detect interference is a promising line of research. Evaluation on hidden and exposed terminal scenarios under different types of bursty traffic showcased its ability to discriminate between interfering and non-interfering situations. Future work would entail more thorough testing of the interfering scenarios under different traffic loads, using TCP traffic, and with varying sending rates. Testing scalability and discriminative power in a large-scale real network environment such as the Jigsaw [9] system is also needed to show that the detection scheme outlined here is appropriate for a real network.

Bibliography

[1] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1–1184, June 12 2007.

[2] Paramvir Bahl, Jitendra Padhye, Lenin Ravindranath, Manpreet Singh, Alec Wolman, and Brian Zill. Dair: A framework for managing enterprise wireless networks using desktop infrastructure. In *Hotnets IV: In Proceedings of the Annual ACM Workshop on Hot Topics in Networks*. ACM, 2005.

[3] Joe Bardwell. Converting signal strength percentage to dbm values. Downloaded from `http://www.wildpackets.com/elements/whitepapers/Converting_Signal_Strength.pdf`, 2002.

[4] Alessia Botta, Alberto Dainotti, and Antonio Pescap. Multi-protocol and multi-platform traffic generation and measurement. Downloaded from `http://www.grid.unina.it/software/ITG/index.php`, May 2007.

[5] Kan Cai, Michael Blackstock, Reza Lotun, Michael J. Feeley, Charles Krasic, and Junfang Wang. Wireless unfairness: alleviate mac congestion first! In *WinTECH '07: Proceedings of the the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 43–50, New York, NY, USA, 2007. ACM.

[6] Ranveer Chandra, Jitendra Padhye, Alec Wolman, and Brian Zill. A location-based management system for enterprise wireless lans. In *NSDI '07: 4th USENIX Symposium on Networked Systems Design and Implementation*. Usenix, 2007.

[7] Ranveer Chandra, Venkata N. Padmanabhan, and Ming Zhang. Wifiprofiler: cooperative diagnosis in wireless lans. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 205–219, New York, NY, USA, 2006. ACM.

[8] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C. Snoeren, Stefan Savage, and Geoffrey M. Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 25–36, New York, NY, USA, 2007. ACM.

[9] Yu-Chung Cheng, John Bellardo, Péter Benkö, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Jigsaw: solving the puzzle of enterprise 802.11 analysis. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 39–50, New York, NY, USA, 2006. ACM.

[10] Yu-Chung Cheng, John Bellardo, Péter Benkö, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Jigsaw trace for january 11, 2007 (sigcomm 2007). Downloaded from `http://sysnet.ucsd.edu/wireless/`, 2007.

[11] Chun cheng Chen and Haiyun Luo. The case for heterogeneous wireless macs. In *Hotnets IV: In Proceedings of the Annual ACM Workshop on Hot Topics in Networks*. ACM, 2005.

[12] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.

[13] Saumitra M. Das, Dimitrios Koutsonikolas, Y. Charlie Hu, and Dimitrios Peroulis. Characterizing multi-way interference in wireless mesh networks. In *WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pages 57–64, New York, NY, USA, 2006. ACM.

[14] Matthew S. Gast. *802.11 Wireless Networks The Definitive Guide*. O'Reilly Media Inc., 2nd edition, 2005.

[15] Ramakrishna Gummadi, David Wetherall, Ben Greenstein, and Srinivasan Seshan. Understanding and mitigating the impact of rf interference on 802.11 networks. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 385–396, New York, NY, USA, 2007. ACM.

[16] P.R. Gupta, P.; Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, Mar 2000.

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

[18] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 66–80, New York, NY, USA, 2003. ACM.

[19] Kyle Jamieson, Bret Hull, Allen Miu, and Hari Balakrishnan. Understanding the real-world performance of carrier sense. In *E-WIND '05: Proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pages 52–57, New York, NY, USA, 2005. ACM.

[20] Amit P. Jardosh, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Understanding congestion in ieee 802.11b wireless networks. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, pages 1–14, New York, NY, USA, 2005. ACM.

[21] Anand Kashyap, Samrat Ganguly, and Samir R. Das. A measurement-based approach to modeling link capacity in 802.11-based wireless networks. In *MobiCom '07: Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 242–253, New York, NY, USA, 2007. ACM.

[22] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *MSWiM '04: Proceedings of the 7th ACM international*

symposium on Modeling, analysis and simulation of wireless and mobile systems, pages 78–82, New York, NY, USA, 2004. ACM.

[23] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Analyzing the mac-level behavior of wireless networks in the wild. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, New York, NY, USA, 2006. ACM.

[24] Ratul Mahajan, Maya Rodrig, and John Zahorjan. CRAWDAD tool tools/analyze/802.11/wit (v. 2006-09-29). Downloaded from `http://crawdad.cs.dartmouth.edu/tools/analyze/802.11/Wit`, sep 2006.

[25] Dragoş Niculescu. Interference map for 802.11 networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 339–350, New York, NY, USA, 2007. ACM.

[26] Travis Oliphant, Robert Kern, and SciPy Dev Team. Scipy 0.60 and numpy 1.0.4. `http://www.scipy.org/`, 2007.

[27] Jitendra Padhye, Sharad Agarwal, Venkata N. Padmanabhan, Lili Qiu, Ananth Rao, and Brian Zill. Estimation of link interference in static multi-hop wireless networks. In *IMC'05: Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference*, pages 28–28, Berkeley, CA, USA, 2005. USENIX Association.

[28] Charles Reis, Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Measurement-based models of delivery and interference in static wireless networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):51–62, 2006.

[29] Armin Rigo. Psyco 1.6. `http://psyco.sourceforge.net/`, 2007.

[30] Bor rong Chen, Geoffrey Peterson, Geoff Mainland, and Matt Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. Technical Report TR-11-07, School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusettes, 2007.

[31] Anmol Sheth, Christian Doerr, Dirk Grunwald, Richard Han, and Douglas Sicker. Mojo: a distributed physical layer anomaly detection system for 802.11 wlans. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 191–204, New York, NY, USA, 2006. ACM.

[32] Lawrence Berkeley National Laborator Berkeley CA USA. libpcap. `http://www.tcpdump.org`, 2007.

[33] Guido von Rossum and Python Software Foundation. Python 2.5.1. `http://www.python.org/`, 2007.

[34] Jihwang Yeo, Moustafa Youssef, and Ashok Agrawala. A framework for wireless lan monitoring and its applications. In *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security*, pages 70–79, New York, NY, USA, 2004. ACM.

[35] Lili QiuYin Zhang, Feng Wang, Mi Kyung Han, and Ratul Mahajan. A general model of wireless interference. In *MobiCom '07: Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 171–182, New York, NY, USA, 2007. ACM.