UNIVERSITY OF CALIFORNIA
RIVERSIDE

Performance and Security Problems in Today's Networks

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Ahmed Osama Fathy Mahmoud Atya

August 2016

Dissertation Committee:

Dr. Srikanth V. Krishnamurthy, Chairperson
Dr. Eamonn Keogh
Dr. K.K. Ramakrishnan
Dr. Vassilis Tsotras

The Dissertation of Ahmed Osama Fathy Mahmoud Atya is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my deepest gratitude to my advisor Professor Srikanth V. Krishnamurthy. I thank you for your continuing guidance and support during my years of research. Your strong belief in the potential of this research has been a tremendous force for the completion of this work. Most importantly, I thank you for encouraging me in each step of my growing path.

Quite often, Im reminded of how amazing my family, teachers and mentors are. Thank you for your support for me that never ceases when I need it. You played an essential rule in feeding my natural interest in science and my tendency to learn. I would never get this far without the support of my parents. Thank you for always believing in me and supporting me. Your love and encouragement have been and will always be a great source of inspiration in my life.

There are many other people and collaborators whose names are not mentioned here. It does not mean that I have forgotten you or your help. It is a privilege for me to work with you. I sincerly thank you for all your efforts.

Chapters two, three and four of this dissertation were published in CoNEXT 2015, Transaction on Networking 2015 and Transaction on Mobile Computing 2015.

To my parents, wife and my son for all your love and support.

ABSTRACT OF THE DISSERTATION

Performance and Security Problems in Today's Networks

by

Ahmed Osama Fathy Mahmoud Atya

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, August 2016
Dr. Srikanth V. Krishnamurthy, Chairperson

The demand for bandwidth is elevating in today's networks. However, security threats and attacks are increasing as well. We study how to maximize throughput and maintain security in (I) wireless and (II) wired communication networks. For throughput maximization, (I) we investigate the cases where applying Network Coding (NC) in a careless manner, could cause significant throughput degradation in multi-rate environments. Via extensive experiments and an analysis, we characterize the regimes where NC offers throughput benefits and those where it does not. We design PACE, a policy-aware coding enforcement logic, which allows a router to switch between NC and store-and-forward modes depending on link qualities. Our evaluations show that PACE could potentially offer network-wide throughput improvements of up to 350%. (II) we propose a standards agnostic framework BOLT, that helps realize the throughput potential of Power Line Communication (PLC) to serve as a viable backhaul for local network connectivity. The design of BOLT is based on a comprehensive measurement study that provides many insights with regards to PLC network characteristics and dynamics. We implement BOLT on three different testbeds using

off-the-shelf PLC adapters and showcase its ability to effectively manage flows, delivering several folds throughput improvement over state-of-the-art solutions.

To maintain security, (I) We propose JIMS, a jamming interference mitigation scheme, using which, transceivers can identify subcarriers that are relatively unaffected by jamming and utilize them for communications. Prior approaches can only alleviate jamming interference to a limited extent; they are especially vulnerable to a reactive jammer, i.e., a jammer that injects noise upon sensing a legitimate transmission or wideband jamming. We show that JIMS restores throughput up to 75% on our WARP testbed. (II) We study VM migration as an effective countermeasure against attempt at malicious co-residency. We first undertake an experimental study on Amazon EC2 to obtain an in-depth understanding of the side-channels attack. Here, we identify a new set of stealthy side-channel attacks and show that migration limits the co-residency time with a victim VM to about 1% of the time with bandwidth costs of a few MB.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The demand for bandwidth is elevating in todays networks. However, security threats and attacks are increasing as well. We study how to maximize throughput and maintain security in (I) wireless and (II) wired communication networks. For throughput maximization, we investigate the problem of probably tuning the used of coding to maximize the throughput. Network coding has been shown to offer significant throughput benefits over certain wireless network topologies. However, the application of network coding may not always improve the network performance. In this work, we first provide an analytical study, which helps in assessing when network coding is preferable to a traditional store-and-forward approach. Interestingly, our study reveals that in many topological scenarios, network coding can in fact hurt the throughput performance; in such scenarios, applying the store-and-forward approach leads to higher network throughput. We validate our analytical findings via extensive testbed experiments. Guided by our findings as our primary contribution, we design and implement PACE, a Policy Aware Coding Enforcement logic

that enables network coding only when it is expected to offer performance benefits. Specifically, PACE leverages a minimal set of periodic link quality measurements in order to make per-flow online decisions with regards to when network coding should be activated, and when store-and-forward is preferable. It can be easily embedded into network coding aware routers as a user-level or kernel-level software utility. We evaluate the efficacy of PACE via (a) ns-3 simulations, and (b) experiments on a wireless testbed. We observe that our scheme wisely activates network coding only when appropriate, thereby improving the total network throughput by as much as 350% in some scenarios. In the wired front, we study powerline communication (PLC). Power line communications offer an immediate means of providing high bandwidth connectivity in settings where there is no in-built network infrastructure. While there is recent work on understanding physical and MAC layer artifacts of PLC, its applicability and performance in multi-flow settings is not well understood. We first undertake an extensive measurement study that sheds light on the properties of PLC that significantly affect performance in multi-flow settings. Using the understanding gained, we design *BOLT*, a framework that adopts a learning-based approach to effectively manage and orchestrate flows in a PLC network. *BOLT* is flexible and is agnostic to standards; it can be used to implement scheduling algorithms that target different performance goals. We implement *BOLT* on three different testbeds using off-the-shelf PLC adapters and showcase its ability to effectively manage flows, delivering several folds throughput improvement over state-of-the-art solutions.

For security, we first investigate the problem of jamming in wireless networks. Malicious interference injection or jamming is one of the simplest ways to disrupt wireless

communications. Prior approaches can alleviate jamming interference to a limited extent; they are especially vulnerable to a reactive jammer i.e., a jammer that injects noise upon sensing a legitimate transmission or wideband jamming. In this paper, we leverage the inherent features of OFDM (Orthogonal Frequency Division Multiplexing) to cope with such attacks. Specifically, via extensive experiments, we observe that the jamming signal experiences differing levels of fading across the composite sub-carriers in its transmission bandwidth. Thus, if the legitimate transmitter were to somehow exploit the relatively unaffected sub-carriers to transmit data to the receiver, it could achieve reasonable throughputs, even in the presence of the active jammer. We design and implement *JIMS*, a Jamming Interference Mitigation Scheme that exploits the above characteristic by overcoming key practical challenges. Via extensive testbed experiments and simulations we show that JIMS achieves a throughput restoration of up to 75% in the presence of an active jammer.

For wired networks, we focus on studying co-residency attacks in today's cloud providers. Attacker VMs try to co-reside with victim VMs on the same physical infrastructure as a precursor to launching attacks that target information leakage. VM migration is an effective countermeasure against attempts at malicious co-residency. In this paper, our overarching objectives are to (a) get an in-depth understanding of the ways and effectiveness with which an attacker can launch attacks towards achieving co-residency and (b) to design migration policies that are very effective in thwarting malicious co-residency, but are thrifty in terms of the bandwidth and downtime costs that are incurred with live migration.

Towards achieving our goals, we first undertake an experimental study on Amazon EC2 to obtain an in-depth understanding of the side-channels an attacker can use to ascer-

tain co-residency with a victim. Here, we identify a new set of stealthy side-channel attacks which, we show to be more effective than currently available attacks towards verifying co-residency. We also build a simple model that can be used for estimating co-residency times based on very few measurements on a given cloud platform, to account for varying attacker capabilities. Based on the study, we develop a set of guidelines to determine under what conditions victim VM migrations should be triggered given performance costs in terms of bandwidth and downtime, that a user is willing to bear. Via extensive experiments on our private in-house cloud, we show that migrations using our guidelines can limit the fraction of the time that an attacker VM co-resides with a victim VM to about 1 % of the time with bandwidth costs of a few MB and downtimes of a few seconds, per day per VM migrated.

Chapter 2

A Policy Aware Enforcement Logic for Appropriately Invoking Network Coding

2.1 Introduction

Wireless Network Coding (NC) exploits the broadcast nature of the wireless medium towards increasing the capacity of the network, by encoding the information contained in multiple packets into a set of fewer packets at intermediate wireless routers [139]. With this, in conducive topologies, NC has been shown to offer significant throughput benefits, compared to a traditional store-and-forward router approach. On the other hand, studies suggest that when NC is blindly applied, it can cause severe degradation of the network throughput, especially in multi-rate environments [65]. In this paper, we show via analysis

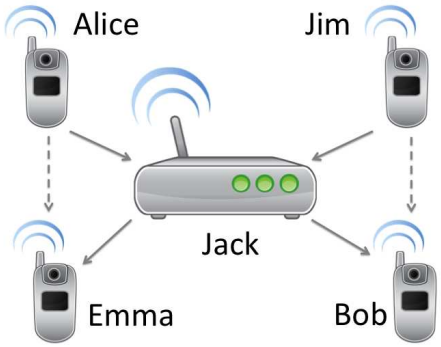

Figure 2.1: A five-node topology that may potentially benefit from NC.

as well as measurements that while network coding is not a magical solution for all wireless network topologies, regulating the use of network coding and store-and-forward together can result in improved long-term throughput benefits.

Performance improvement due to NC: now you see it, now you don't: Wireless network coding has been examined both theoretically and experimentally over the past decade, under many different deployment and traffic scenarios [139, 65, 148, 153, 219, 96, 206, 137]. Two primarily identified factors affect the performance improvements due to wireless NC:

(1) The network topology, which determines the ability of neighboring devices to successfully overhear each other's transmissions in order to decode encoded packets; and, **(2)** The traffic patterns of the different users, which dictate the number of packet encoding opportunities at the routers.

Let us consider the simple topology of Fig. 2.1, where Alice sends data to Bob, while Jim sends data to Emma, all routed via Jack. From among the above two factors, it is easy to see that #2 dictates the network coding gain: if Alice has a much higher application data

rate than Jim, then the router (Jack) will rarely be able to encode Alice's and Jim's packets together. However, the application of NC here, will never degrade the overall throughput[1] due to factor #2. On the other hand, factor #1 can be the reason for significant throughput degradation in the presence of NC. In particular, let us assume that both links *Alice-Jack* and *Jim-Jack* have a Packet Delivery Ratio (PDR) equal to 1 at 54 Mbps, while the overhearing link *Alice-Emma* has a PDR equal to 0.2 at this rate. In this topological scenario, Jack will receive packets from Alice and Jim at similar bit rates. But, if Jack decides to constantly apply NC given the high availability of candidate packets, this will cause a tremendous degradation in the overall network throughput (compared to simply applying store-and-forward). This is because Emma will not be able to decode 80% of the delivered encoded packets by Jack, regardless of the coding gain, due to the poor link quality that she maintains with Alice. In other words, the throughput achieved with store-and-forward would be higher than that achieved with NC here. If Jack greedily prefers NC to store-and-forward whenever native packets from Alice and Jim are available in Jack's queues, his strategy will backfire and hurt the network performance. Clearly, if Alice adapts her transmission rate in order for the PDR on the link *Alice-Emma* to increase, then the long-term throughput due to NC may end up being higher than that with store-and-forward, or it may not! Note that recent studies have proposed novel bit rate adaptation algorithms that take into consideration the existence of network coding [148, 153]. However, they have not examined whether a store-and-forward strategy would still be more beneficial than NC in diverse channel environments, even when such NC-aware rate control protocols are applied,

[1]We assume here for the sake of the discussion that the system overheads imposed due to NC do not affect the performance.

as we discuss in Section 4.2.

Designing an adaptive decision making engine: Given the above discussion, in this paper we design and implement PACE, a Policy Aware Coding Enforcement logic for NC-capable wireless routers. PACE leverages a small set of periodic link quality measurements in order to decide whether NC or store-and-forward must be applied, for each particular data flow that traverses the router, in real time. The logic of PACE is guided by our analytical study, which considers different topological scenarios, and provides insights on which approach (NC or store-and-forward) is expected to offer higher long-term network throughput. More specifically, our contributions in this paper are the following:

1. ***Analysis of the gains from network coding:*** We perform an analytical assessment of the achieved network throughput with NC and store-and-forward, for a general topological class. We verify the accuracy of our analytical assessments via extensive testbed experiments, using a novel network coding software platform [58]. Our analysis, in conjunction with our testbed measurements, provides recommendations on when it is preferable to apply NC, and when store-and-forward is a wiser choice, in multi-rate settings. With this, we construct a concrete set of NC application guidelines for each considered topological scenario.

2. ***Design of PACE:*** We use our guidelines to design PACE, our Policy Aware Coding Enforcement logic. PACE performs periodic link quality measurements to assess the potential effectiveness of NC. It leverages our analytical model to make online router decisions regarding when and how NC or store-and-forward should be enabled and what transmission bit rates are to be used in conjunction.

3. ***Measurements and simulations:*** We implement PACE on our wireless testbed, and we evaluate its efficacy via measurements over numerous different topologies and diverse link qualities. Moreover, we develop PACE in ns-3 and perform extensive simulations over large-scale single and multi-hop network deployments. Our simulations and experiments demonstrate that PACE always follows the right strategy with regards to the application of NC; with this, it results in throughput improvements of up to 350% in some cases, i.e, by switching on NC when appropriate.

Our work in perspective: In this paper we primarily focus on local NC topologies where an encoded packet by a router is decoded at the next hop. As we discuss in Section 2.6, this has direct applications in wireless LANs. We do not consider problems such as topology discovery, selection of which packets to code together or aggregated ACK/NACK packets. Such issues have been addressed by other studies such as [139]. Our proposed scheme is directly applicable in previously proposed NC frameworks, such as [139, 206, 89], etc. To show the applicability of our approach in more generic settings, we rely on simulations in scenarios wherein packets traverse multiple wireless hops with potential opportunities to use network coding at each such hop (as in mesh networks).

The rest of the paper is organized as follows. In Section 4.2, we discuss related work. In section 2.3, we present our generic analytical model and its validation. In Section 4.4 we develop the PACE algorithmic logic, which we evaluate in section 4.5. In Section 2.6 we discuss the scope of our study. Finally, our conclusions form Section 5.9.

2.2 Related Work

In this section, we discuss previous relevant NC studies. The idea of NC first appeared in [39] where Ahlswede et al. showed that by performing algebraic bit-wise operations in routers and by forwarding mixtures of packets, one can tremendously increase the transfer capacity of a multicast network. This work motivated the subsequent generation of a plurality of efforts to understand this concept and further exploit it in improving the network performance. As examples, Li et al. [159] show that linear codes are sufficient for achieving the optimal capacity in multicast traffic scenarios, although they are not sufficient for arbitrary network demands. Gkantsidis and Rodriguez [107] propose a platform for peer-to-peer content distribution, which depends on application-layer NC operations. Koetter and Medard [150] present polynomial-time algorithms for realizing network operations of encoding and decoding packets. Ho et al. [122] extend the algorithms of [150] and discuss distributed NC schemes. In addition, [121], [261] and [160] show that for many specific scenarios, NC results in better throughput than pure forwarding. In [146], the authors study the capacity of 2-hop relay networks and propose a near optimal coding scheme that can make a linear number of decisions in terms of which packets to combine using network coding. In the paper by Sharma et al., the authors examine if network coding can benefit cooperative communications (CC) with multiple simultaneous flows. They consider analog network coding (as opposed to simple XOR operations as we do here) and quantify the noise at a node that aggregates packets. They demonstrate analytically that this noise can diminish the benefits of analog network coding. In contrast, our work examines the impact of link qualities and traffic load on the benefits achievable with XOR based network

coding (NC). Furthermore, we do not consider either cooperative communications or analog aggregation. Finally, unlike in [226] which focuses on theoretical analysis of NC with CC, we focus on experimentation, and the design of a policy to only apply network coding in conducive conditions.

2.2.1 NC applicability assessment

The study that is mostly relevant to our work is by Chaporkar and Proutiere [65]. Similar to our work, they show that in multi-rate settings, systems with NC may have smaller throughputs than without coding. They argue that unless appropriate scheduling is applied, NC may lead to performance degradation in many scenarios. They further propose a generic framework that characterizes the throughput region with NC and enables the design of adaptive joint NC and scheduling schemes. However, they do not provide any generic guidelines or an online method for adaptively activating NC when it is expected to increase throughput. Moreover, [65] does not involve real network experimentation and/or measurements.

In [256], the authors analytically studied the capacity region of M unicast sessions in single hop relay networks by modelling the broadcast packet erasure stationary channels. This work does not consider store-and-forward or involve any real network experimentation and/or measurements.

In [139], COPE is proposed for practical wireless coding. COPE does turn off network coding if packet loss rates are higher than 20%. However, this value is empirical.

Unlike in the above efforts, our contribution is in quantifying the degradation in the performance when the overhearing links PDR is low. Specifically, unlike in other efforts

[139], we make a determination on at what point the quality is detrimental to network coding.

2.2.2 Analytical and simulation studies on wireless NC

Liu and Xue in [163] analytically characterize the achievable rate regions with NC for a basic 3-node topology wherein no overhearing is involved. Vieira et al. [249] examine how the combination of NC and bit rate diversity affects the performance of broadcasting protocols. Scheuermann et al. [217] propose noCoCo, a deterministic scheduling scheme for NC to operate on two-way multihop traffic flows. Seferoglu et al. [218] propose code selection schemes that consider the properties of video traffic. Le et al. [96] provide an upper bound on the number of packets that can be coded together. Lun et al. [88] show that the problem of minimizing the communication cost can be formulated as a linear program and solved in a distributed manner. There has also been some work on NC-aware data rate control at the transport layer [219, 220]. However, *these studies do not address the problem of choosing between NC and store-and-forward towards improving the long-term network throughput.*

2.2.3 Experimental work on wireless coding

Katti et al. [139] propose COPE, the first, seminal implementation of wireless NC. Since one of the goals of COPE is to increase the number of encoding opportunities, low transmission rates are favored in order for native packets to be overheard by as many neighbors as possible. Their experiments with COPE show that even with very simple encoding operations, NC can provide significant capacity gains. However, they do not

study cases where store-and-forward is preferable to NC. Rozner et al. in [89] present ER, a scheme that adopts the design of COPE and employs NC to perform efficient packet retransmissions. Rayanchu et al. [206] propose CLONE, a suite of algorithms for NC that take into account channel losses. Both [89] and [206] follow COPE's logic regarding the application of NC; they do not propose any policies for multi-rate settings.

Srinivasan et al., [233], propose a network metric that takes into consideration the inter-link interference on packet reception probability. However, they do not consider transmissions at multiple bit rates or rate adaptation.

MORE [93] is a routing protocol that performs a random mixing of packets, right before they are forwarded. With this, routers that overhear a transmission can decide not to forward the same overhear packets. However, no decisions on NC versus store-and-forward are made. MIXIT [137] encodes symbols rather than packets. Relays use hints from the PHY layer in order to infer which symbols within a packet are correctly received with high probability. Note here that all of these studies are transmission rate unaware.

Kim et al. [148] study the performance of NC in multi-rate settings. They show that unless rate adaptation is NC-aware, NC may not offer significant performance benefits. They further design a NC-aware rate control algorithm for local topologies. Kumar et al. [153] take the same path but propose a different NC-aware rate control algorithm.

Hulya et al., in [221], proposed I^2NC to overcome the non-negligible loss rates by combining inter-session and intra-session network coding. In their work, intra-session specifies the amount of redundancy required to compensate for errors and inter-session chooses the number of flows to code together. In [207], the authors proposed CLONE to improve

the performance of NC by being aware of potential losses and introducing redundancy to cope with these losses. Neither [221] nor [207] consider potential throughput benefits with network coding in muti-rate settings.

The idea of using redundancy with network coding (as in [221]) is orthogonal to the question we are asking: *"when should network coding be applied in multi-rate settings?"*

All the above papers implicitly assume that NC should be applied whenever possible. However, as we discuss in Section 2.3, this should not always be the case as it may lead to performance degradation.

NCRAWL [58] is a multi-rate network coding library. NCRAWL does not provide any insights by itself on when to (or when not to) apply network coding. We use NCRAWL as the underlying framework over which we implement PACE. In Section V, we provide more details on these implementation aspects.

2.2.4 NC on wireline networks

Finally, a large body of studies have investigated NC for wireline networks (e.g., [122, 160, 150]). However, they do not account for the inherent properties of the wireless medium.

In short, very limited steps have been taken towards assessing the applicability of practical NC in relation with the inherent characteristics of the wireless medium. As we discuss below, the application of wireless NC should be regulated.

2.3 Profiling the Applicability of NC

In this section, we discuss our analytical model for assessing whether/when the application of NC is preferable to store-and forward. Furthermore, we verify the accuracy of our analytical model via experiments on our wireless testbed.

2.3.1 Designing our analytical model

Our goal is to derive the throughput with both store-and-forward and NC in various single-hop topological scenarios. For this, we design an analytical model, which essentially computes the average number of packet transmissions in a multi-rate environment, for both the cases and from that, the throughput. We consider the general topological scenario in Fig. 2.2, which consists of N packet senders and N receivers (a total of N source destination pairs) communicating via a relay node R.

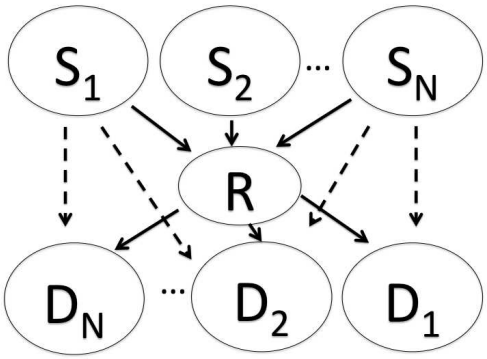

Figure 2.2: A N-node topology wherein network coding may offer performance benefits when the link qualities among the different users are conducive.

Model assumptions

Our considered setting involves links with heterogeneous qualities in terms of Packet Error Rate (PER). At a specific time instance, each participating device (node) uses a specific transmission bit rate R_{node}, while every link between a sender and a receiver has a PER equal to $p_{Sender-Receiver}$; we assume that the network is quasi-stationary, wherein PER values remain unchanged for relatively long periods. This assumption is realistic in cases with slowly varying channels. This happens when there is not much mobility and the interference patterns are quasi-static (e.g., an office space). If the fading conditions are much more dynamic, it is hard assess channel qualities. This limitation is not unique to our problem; even rate adaptation schemes may underperform [63] in such scenarios. Given that we are not designing new channel quality estimation methods, we defer solving this problem to future work. Without loss of generality, in order to make the analysis tractable, we assume that data packet lengths have a fixed size equal to L bits. Note that our analysis focuses on the generic topology of Fig. 2.2, wherein encoded packets (i.e., mixtures or two or more native packets) constructed at a router, are decoded at the next hop. We do not consider cases where encoded packets traverse more than one hop. We elaborate on this assumption later in Section 2.6. Table 5.1 summarizes the notation used in our analysis.

The "packet" transmission rate between nodes i and j is $X_{ij} = \frac{R_i}{L}$ packets/sec, while the transmission time, T_{ij}, of a packet is equal to $\frac{1}{X_{ij}}$. In what follows, we first analytically derive the average number of transmissions for each individual forwarding strategy separately. We use these derivations to estimate the average throughputs with the two cases. The analysis provides a quick and efficient way of understanding when to use NC

Table 2.1: Definitions of Notations

Symbol	**Definition**
R_i	Transmission rate for node i
L	Packet length
n	Number of flows to be coded together (equals 1 in case of store-and-forward).
$\rho^{sf}_{S_kR}$	Average number of transmissions from Sender S_k towards the Relay (case of store-and-forward).
$\rho^{sf}_{RD_k}$	Average number of transmissions from the Relay towards Destination D_k (case of store-and-forward).
$\rho^{nc}_{S_kR}$	Average number of transmissions from Sender S_k towards the Relay (case of NC).
$\rho^{nc}_{RD_k}$	Average number of transmissions from the Relay towards Destination D_k (case of NC).
ρ_R	Average number of transmissions for encoded packets from the Relay towards all intended recipients.
T_{S_kR}	Average transmission time between a sender node S_k and the relay node R.
T_{RD_k}	Average transmission time between the relay node R and a destination node D_k.
T_R	Average transmission time for the encoded packet from the relay R towards the selected MAC-level destination.
Q_{sf}	Average queuing time in the store-and-forward (sf) case.
Q_{nc}	Average queuing time in the NC (nc) case.
P_{nc}	Average processing time overhead in the NC (nc) case.
p_{AB}	Probability of error of the link between nodes A and B.
M	Maximum number of transmissions.

and when to use store-and-forward.

The case for store-and-forward

The average time taken to deliver a packet from a source to a destination in the case of store-and-forward, sf, is given by:

$$T_{avg}^{sf} = \frac{\sum_{k=1}^{N} (\rho_{S_k R}^{sf} T_{S_k R} + \rho_{RD_k}^{sf} T_{RD_k})}{N} + Q_{sf}. \tag{2.1}$$

In the above expression, the numerator represents the total time taken to transfer 'one' packet, on average, from each source to its destination. We take the average over all source-destination pairs. In addition, we include the average packet queuing time experienced by the packet, prior to its transmission attempts. The throughput of the store and forward scheme is then simply

$$\tau^{sf} = \frac{1}{T_{avg}^{sf}}. \tag{2.2}$$

The average number of transmissions for a packet from the source to its destination depends on the value of PER between the sender and the relay as well as on the PER between the relay and the destination. Thus we need to consider the links *sender-relay* and *relay-destination* in our analysis. For instance in the X-topology in Fig. 2.1, we consider individually, the direct links *Alice-Jack, Jim-Jack, Jack-Bob* and *Jack-Emma*. Taking into account the PER on each of these individual links, the average number of transmissions in the store-and-forward case on the link between the sender and the relay can be computed as (details are in an Appendix):

$$\rho_{S_k R}^{sf} = \frac{(1 - (M+1)p_{S_k R}^{M} + M p_{S_k R}^{M+1})}{1 - p_{S_k R}}, \tag{2.3}$$

while the average number of transmissions between the relay and the destination can be computed as,

$$\rho^{sf}_{RD_k} = (1 - p_{S_k R}) \frac{(1 - (M+1)p^M_{RD_k} + Mp^{M+1}_{RD_k})}{1 - p_{RD_k}} \tag{2.4}$$

where, M is the maximum number of transmission attempts (including retransmissions) on any given link (we set $M = 7$ in this work, as suggested in [33]).

The case for network coding

As with store-and-forward, the average number of transmissions with NC depends on the PER on the *sender-relay* and *relay-destination* links. However, in addition, the likelihood of correct reception also depends on the value of PER on overhearing links (e.g., Alice $\rightarrow$ Emma). Given this, the average number of transmissions in the NC case for the generic topology between the N senders and the relay node in Fig. 2.2 is (see the Appendix for the full derivation):

$$\rho^{nc}_{S_k R} = \frac{(1 - (M+1)p^M_{S_k R} + Mp^{M+1}_{S_k R})}{1 - p_{S_k R}}, \tag{2.5}$$

and the average number of transmissions between the relay node and the destinations is given by:

$$\rho^{nc}_{RD_k} = (\prod_{i=1}^{N} (1 - p_{S_i R})) \frac{(1 - (M+1)p^M_{RD_k} + Mp^{M+1}_{RD_k})}{1 - p_{RD_k}} \tag{2.6}$$

where, D_k is the chosen 802.11-level destination for the encoded packet by the relay[2]. In our work, the relay selects D_k to be the recipient with the lowest PDR value. With this,

[2]Encoded packets are unicasted to a specific node; all other recipients that successfully decode it, report to the relay the identities of the native packets that they have successfully obtained from decoding, as in [139, 58].

all intended recipients obtain the encoded packet with high probability. Hence, for ease of notation $\rho^{nc}_{RD_k}$ is simply refered to as ρ^{nc}_{R} here forth. The product $(\prod_{i=1}^{N}(1-p_{S_iR}))$ represents the success probability of packets being delivered from the transmitter(s) to the relay nodes. In the case of store and forward, we compute the success probability of each individual packet (in reaching the relay node from a sender k). In the network coding case, there are two or more incoming links to the relay; all packets need to be successfully delivered for encoding. We assume that the probabilities of success on each of the ingress links to the relay are independent (this is realistic if they experience independent interference patterns).

The average throughput in the case of NC depends not only on the average transmission times of native and encoded packets, but also on the overhearing probabilities at receivers. Based on the details in the Appendix, the average packet transmission time is given by:

$$T^{nc}_{avg} = \frac{(\sum_{k=1}^{k=N} \rho^{nc}_{S_kR} T_{S_kR}) + \rho^{nc}_{R} T_R}{N} + Q_{nc} + P_{nc} \tag{2.7}$$

where T_R is the average transmission time for the encoded packet from the relay R towards the selected MAC-level destination. Thus, T_R is the reciprocal of transmission rate used by the relay for its target destination.

and the decoding probability is given by:

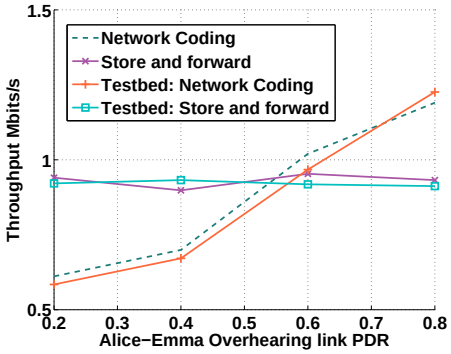

Figure 2.3: Varying the PDR for the overhearing link between Alice and Emma

$$p_i^{dec} = \prod_{k=1}^{n-1}(1 - p_{S_kR}^M)(1 - p_{S_kR})(\frac{1 - p_{S_kR}^M}{1 - p_{S_kR}} + p_{S_kD_i}\frac{1 - (p_{S_kR}.p_{S_kD_i})^M}{1 - p_{S_kR}.p_{S_kD_i}}) \tag{2.8}$$

The throughput with NC is then computed to be

$$\tau^{nc} = \frac{p_i^{dec}}{T_{avg}^{nc}}. \tag{2.9}$$

In Eq. 2.7, T_{avg}^{nc} is the sum of (i) the average transmission time, (ii) the average queuing time, and (iii) the processing time with NC. In our evaluations we use the measured values of queuing and processing times [58] in computing our analytical results since it is hard to derive exact expressions for these.

2.3.2 Experimental Validation and Inferences

Next, we validate the results from our analysis with real WiFi testbed measurements. We also draw inferences from our observations towards later formulating a set of guidelines on which PACE, our encoding logic, is based.

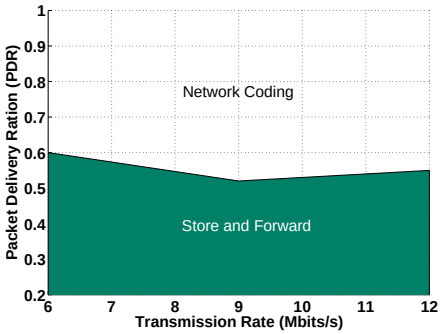

Figure 2.4: The value of the PDR on overhearing links affects the efficacy of NC (X-topology)

We defer a detailed description of our wireless testbed [28] to Section 4.5. In a nutshell, our testbed consists of Soekris net4826 boxes that run 802.11a/g. The NC functionality in our experiments is managed by the novel NCRAWL software platform [58], which has been designed specifically to accommodate multiple bit rate NC measurements. While we have cross-verified a part of our measurements with the COPE platform [139], we omit those results here in the interest of space. Each experiment lasts for approximately 5 min and is repeated 20 times, for both the NC and store-and-forward cases; for each run we log the achieved average throughput with each strategy.

We conduct an extensive set of testbed experiments across different topologies in terms of node populations and link qualities. In particular, we examine local topologies (with a single relay node) wherein we vary the Packet Delivery Ratio (PDR = 1 – PER) and the bit rate on both direct and overhearing links; we consider PDR values that range between 0.2 and 0.8, and bit rate values ranging between 6 and 12 Mbps.

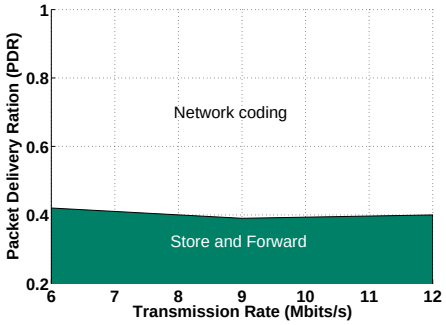

Figure 2.5: The value of the PDR on overhearing links affects the efficacy of NC (Wheel-topology)

We observe that the analytical results match the experimental results fairly well in all the scenarios considered (as seen in the corresponding figures discussed below).

Varying the PDR on a overhearing link with fixed rate

We consider various settings wherein we fix the bit rate to a specific value, while we vary the PDR on one of the overhearing links. The PDR on all other links is '1'; we adjust the transceiver positions to ensure that this is the case. Our goal here is to observe how the quality of overhearing links affects the efficacy of NC. For example, in the network setting of Fig. 2.1 we vary the PDR on the link *Alice-Emma*.

Case of two overhearing links (the X topology) Fig. 2.3 (left) depicts the average per-user throughput vs. PDR for the overhearing link between Alice and Emma, when the transmission rate (on all links) is 6 Mbits/sec. We observe that the throughput with NC is higher than that with store-and-forward when the PDR on the link *Alice-Emma*

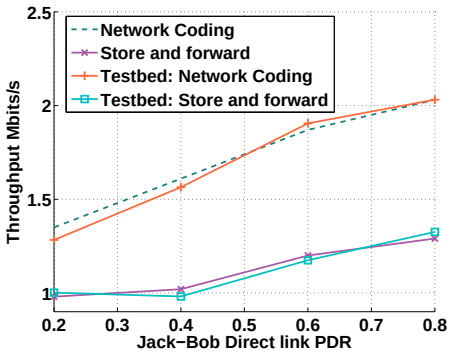

Figure 2.6: High PDR values on the relay's incoming links favor NC even if one outgoing link is poor.

is above 0.5; for values lower than 0.5 the application of NC hurts the average long-term throughput! This drop in the throughput with NC occurs due to the inability of Emma to successfully overhear Alice's transmissions. This renders the decoding of Jack's encoded packets impossible for her. We vary the PDRs on the other links, but find that if the overhearing link is poor, it makes no difference, i.e., NC always performs worse than store-and-forward (we do not present additional results here) Similar results are also seen with the other transmission bit rates. Based on these experiments, we conclude that *the decision on whether to apply NC should consider the qualities of the overhearing links; if the PDR is low on such links, NC is likely to degrade the network throughput.*

Cases with higher numbers of overhearing links (Wheel topology) Next, we compare the requirements on the overhearing link qualities with the X (2 flows) and wheel (3 flows) topologies for the range of rates considered. Fig. 2.4 and Fig. 2.5 depict results for different transmission rates vs. varying PDR values for overhearing links. In particular, we

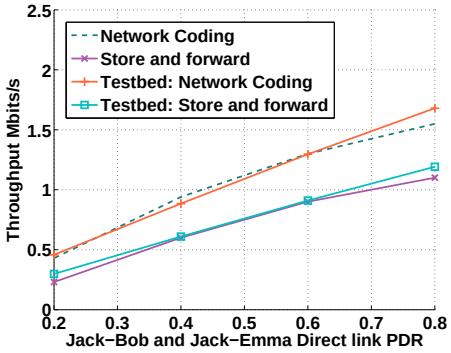

Figure 2.7: High PDR on the relay's incoming links favor NC even if both outgoing links are poor.

seek to find the sweet spot where NC outperforms store-and-forward. The x-axis depicts the transmission rate and the y-axis indicates the various PDRs achievable at those rates. The green (dark) region depicts the PDR (for each rate) where store and forward outperforms network coding. If PDRs are in the white region (light), network coding should be chosen for operations. We observe that NC gains compensate for some of the losses due the link errors. Specifically, in the X topology (Fig. 2.4) NC outperforms store-and-forward when the PDR is greater than 60% on all overhearing links. In the wheel topology (Fig. 2.5), NC outperforms store-and-forward when PDR is greater than 40%. This is because in the wheel topology case, with NC the relay typically encodes two or three packets together and thus the required number of outgoing transmissions is reduced; this compensates for the overhearing link losses to some extent. On the other hand, one would expect that the existence of more overhearing links with low PDR in wheel topologies increases the probability of erroneous overhearing and thereby decreases the achieved throughput with

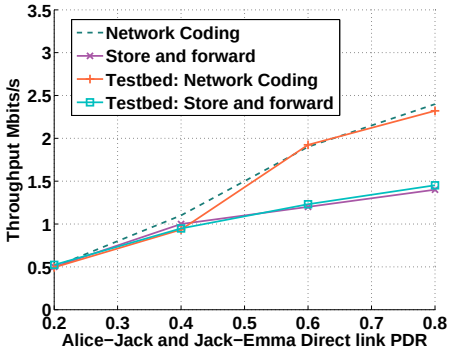

Figure 2.8: Low PDR values on the relay's incoming links affects the potential for encoding negatively.

NC. However, our experiments demonstrate that a PDR>0.4 on all overhearing links is sufficient for NC to perform better than store-and-forward; the reduced number of required transmissions compensates for the link losses on the overhearing links. Hence, we conclude that *the performance of NC when applied on wheel topologies is less sensitive to the PER (on the overhearing links) than when NC is applied on X topologies.*

Varying the PDR on the the direct (non-overhearing) links We classify the direct links as incoming (e.g., Jim → Jack) or outgoing (e.g., Jack → Emma) depending on their relative positions with respect to the relay. We perform an exhaustive set of experiments and make several observations, but do not present all the results here. We only present a key set of results instead with a transmission bit rate of 12 Mbps (similar results are seen with other rates). In all these experiments, we maintain a good quality for the overhearing links (i.e., PDR = 1).

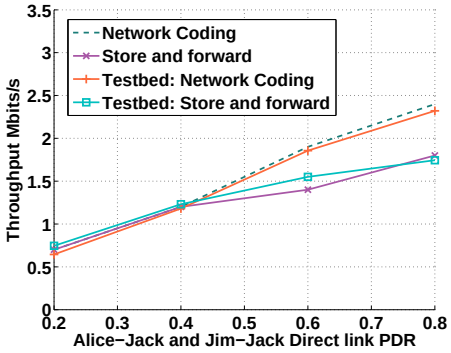

Figure 2.9: High PDR values on the relay's outgoing links do not help if the incoming links are poor.

Case with high quality incoming links Our first observation is that if all the links are of high quality, NC provides significant gains over the store-and-forward approach. We do not show this result explicitly for space constraints; however, this is captured in the next result that we show in Fig. 2.6. Here, we maintain high quality incoming links, but we vary the quality of one of the outgoing links (Jack $\rightarrow$ Bob). We see that NC outperforms store-and-forward always, and especially when the considered link is of high quality. The reason for this is the following. Due to the fair allocation nature of 802.11, store-and-forward can only provide a throughput equivalent to the poorest outgoing link from Jack (e.g. see [119]); thus, transmitting encoded packets at a rate that satisfies this poor receiver is the best one can do (it saves on the transmission over the better link). This is also reflected in Fig. 2.7, wherein we vary the PDR on both of the outgoing links while maintaining the high quality of the incoming links. *In summary, if the PDR on incoming links to the relay is high, it is always better to use NC. In other words, the quality of the outgoing links does not matter*

as long as the quality of the incoming links is good.

Case with poor quality incoming links Next, we vary the quality of one of the incoming links (Jim → Jack) and one of the outgoing links (Jack → Bob) simultaneously. We find that this causes a performance degradation with NC (see Fig. 2.8) if the pair of links have low PDR. This is because, the mismatch in the quality of the incoming links, causes a queue imbalance at the relay (Jack). Thus, the likelihood of encoding even if NC is applied by default, is very low. As a consequence, there are simply no gains to be had. The processing with NC slightly hurts performance compared to store-and-forward. As the link qualities improve and we approach a regime where all links are again good, the gains due to NC are apparent.

In the final experiment in this section, we vary the quality of both the incoming links to the router, Jack. The overhearing links and the outgoing links are all of good quality i.e., PDR ≈ 1. The throughput results with NC and the store-and-forward scheme are presented in Fig. 2.9. Again, we notice that when the PDR on the incoming links is low, there are no gains from NC relative to the store-and-forward case. The reason for this is that the input rate to Jack's queues from Alice and Jim, are low due to poor PDR. Therefore, Jack typically does not find packets from both senders and thus, is rarely able to encode packets. As the PDR increases on the incoming links, the benefits due to NC begin to increase. Again, when these links are of good quality (PDR = 0.8), NC outperforms store-and-forward by about 30% in terms of the achieved throughput (as expected, since one transmission is gained relative to the store-and-forward case).

Based on these experiments we conclude that *the decision on whether to apply NC*

should consider the qualities of the relay's incoming links; when the PDR on the incoming links is low, coding opportunities may be infrequent. Perfect overhearing does not necessarily imply that network coding should be performed if the quality of incoming links are poor.

Other conclusions: Our experiments also lead to two other conclusions (implicit in our discussions above). *(a) The outgoing links of the router are a non-factor in determining whether or not NC should be applied.* And, *(b) The dependence on the transmission bit rate is not explicit. However, the choice of the bit rate implicitly affects the quality of the overhearing and incoming links and thus, it would affect the decision on whether NC should be applied or not.*

Summary and Scope

While we have presented results with simple topologies, the results hold for more complex wheel topologies (many overhearing links) which inherently present opportunities for NC. The string topology is a special case of the X topology; it is a case where no overhearing is necessary since the native packets are already available at the end-destinations. In such cases, as long as the links are of good quality, NC helps; if these links are of poor quality, there are no gains to be had compared to store-and-forward although, there is no significant hit in performance either.

2.4 Designing PACE

To summarize our study in the previous section, we draw two main conclusions: For any given rate, NC should not be applied when: **(1)** the quality of the overhearing

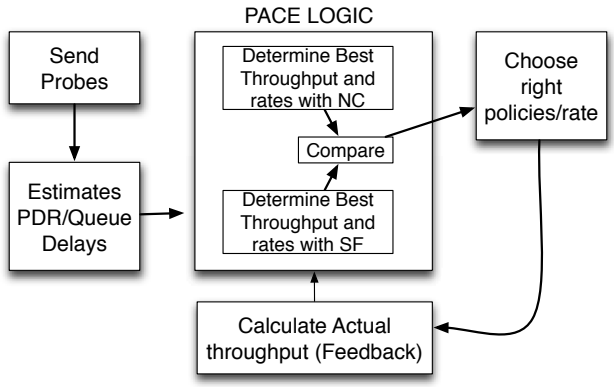

Figure 2.10: The main components of PACE.

links are poor ($PDR < 0.6$ with two overhearing links and $PDR < 0.4$ with more than two overhearing links); and **(2)** the overhearing links are of good quality, but any of the incoming links to the router are of poor quality ($PDR < 0.4$). Based on these observations, we design our decision logic engine, PACE, next.

2.4.1 Design overview

The goal of PACE is to regulate the use of NC at routers. However, it is difficult to apply the above rules directly in a multi-rate setting since the properties of the links depend on the transmission bit rate in use. Below, we describe how PACE determines whether or not NC is to be applied and the specific transmission rates to be used, at each local topology where NC can be potentially invoked.

2.4.2 Assessing the quality of links

The first step in the process is to acquire the quality of the different links (in terms of PDR) in a local topology. Here, PACE leverages the ETT probing mechanism

[80]. Specifically, each node periodically transmits probes at different rates and reports the percentage of received probes from each neighbor. With this, the relay obtains accurate information about the PDR for every rate, on each link in the neighborhood. In addition, since we use the model derived in Section 2.3, and the average queueing delays are needed here, we modify the probe formats to allow each node to report its average queuing delay over the past ten packets to the router.

2.4.3 Determining the best throughput with NC

Next, PACE seeks to determine the best throughput with NC. As the transmission rates increase, the quality of the overhearing links could potentially degrade. PACE determines the highest rate (say $\mathcal{R}_{NC}$) at which, the link qualities satisfy the requirements mandated by our guidelines above for invoking NC. It is easy to verify that this rate $\mathcal{R}_{NC}$ provides the best case for NC. Specifically, at lower rates, lower throughputs are achieved. More importantly, the NC throughput is most likely higher than the store-and-forward case at rate $\mathcal{R}_{NC}$ (since the conditions mandated by the guidelines for applying NC are satisfied at this rate). However, if the rate is further increased, the store-and-forward approach could deliver higher throughputs than NC (but this is not known at this point). The router (e.g., Jack) then applies our analytical model to compute this best case throughput (with rate $\mathcal{R}_{NC}$) for NC (say $\mathcal{T}_{NC}$).

2.4.4 Choosing the policy

Now that PACE has determined the highest throughput with NC, the question that has to be answered is: *"Is a higher throughput possible with store-and-forward with higher*

rates?" At rates higher than $\mathcal{R}_{NC}$, the store-and-forward approach may provide higher throughputs than it would at rate $\mathcal{R}_{NC}$. We seek to examine if the higher throughput with store-and-forward exceeds the throughput achieved with NC at rate $\mathcal{R}_{NC}$.

If we examine the store-and-forward throughput with different topologies, we see the following behavior. The throughput first increases upon increasing the rate. Either this behavior continues until we hit the maximum transmission rate (e.g., 54 Mbps with 802.11a) or begins to drop beyond a point. The reason for this is that as we increase rates, the PDR will drop (causing more retransmissions and delays); however, the packet transmission time decreases. Initially, the second factor dominates. At some point it is possible (depending on the topology) that the first factor begins to dominate and thus, the throughput falls. This behavior is shown for two example store-and-forward flows in a simulated X topology in ns3 (Fig. 2.11). We could not validate this experimentally since our implementation only supports rates up to 12 Mbps.

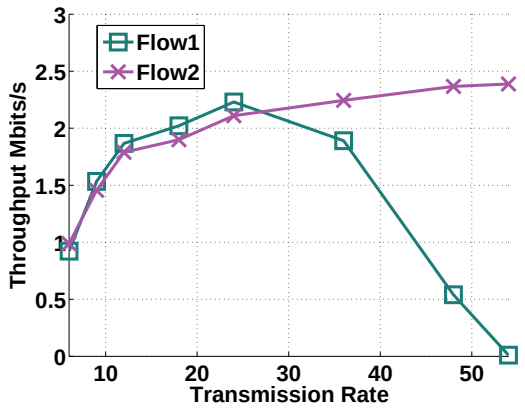

Figure 2.11: Throughput vs. transmission rates for two store-and-forward flows.

Given the behavior, we do the following. If $\mathcal{R}_{NC}$ is the highest rate possible (e.g.,

54 Mbps with 802.11a), we simply decide to use NC. If not, and there is just one higher rate (say $\mathcal{R}_{NC} = 48$ Mbps with 802.11a), we simply check the throughput of the store-and-forward case at this higher rate using our analytical model. If this computed throughput (say $\mathcal{T}_{SF}$) is higher than $\mathcal{T}_{NC}$, we choose store-and-forward; otherwise we choose NC. If there are two or more higher rates, we begin with the rate that is immediately higher than $\mathcal{R}_{NC}$, and compute the store-and-forward throughput with that rate and the next higher rate. If the throughput is increasing, we keep checking the throughputs at the higher rates, until we hit the peak throughput (as suggested by Fig. 2.11) or the maximum rate. The throughput with the store-and-forward at that rate is now the highest throughput possible with that approach ($\mathcal{T}_{SF}$). We compare the values of $\mathcal{T}_{SF}$ and $\mathcal{T}_{NC}$ as before and choose the winning policy. Alg. 4 shows the algorithmic steps followed by PACE. The counter represents the number of incoming flows to the relay. If they are less than 2, then the relay cannot encode. If they are exactly 2, then, we consider a certain threshold value. If there are more than 2 incoming flows, we consider a different value for the threshold. These values were shown to give the best performance via our measurements.

Note that the PACE logic is transparent to other NC procedures, such as cumulative packet acknowledgements [139, 58], decisions on which packets to code together, etc., as we discuss in section 2.6. Also note that the router (Jack) locally computes the processing delays with NC, and uses the computation to estimate the throughput with our analytical model as above. Moreover, PACE may employ alternative schemes for accumulating PDR information at the relay [59].

Remark: Coding a subset of flows instead of coding all-or-nothing is a possible

Algorithm 1 PACE Algorithm.

Input: ETT neighbor reports ($< \mathcal{R}_k$,p_{S_iR} for rate $\mathcal{R}_k >$, $< \mathcal{R}_k$,p_{RD_i} for rate $\mathcal{R}_k >$, Q_{sf} (at sources)) $\forall i$ and $p_{S_iD_j}$

$\forall i, j$ such that $i \neq j$ and $\mathcal{R}_k = \{6,\ 9,\ 12,\ 18,\ 24,\ 36,\ 54\}$

Output: select_NC and rate vector $\vec{r}$

Initialization: counter $\leftarrow 0$, $k \leftarrow \{1\}$,stop $\leftarrow$ false and select_NC $\leftarrow$ true

//Check if the guidelines with regards to links hold

while ! stop **do**
- $k \leftarrow$ Next k // k is an index and Next k is not always $= k + 1$
- **for** $i \leftarrow 0$ **to** n **do**
  - **if** *(1 -* p_{S_iR}*)* > 0.4 **then**
    - counter $\leftarrow$ counter $+ 1$
  - **end**
- **end**
- **if** counter $<$ *2* **then**
  - stop $\leftarrow$ true
- **end**
- **else if** counter $==$ *2* **then**
  - // Two flows case
  - **for** $c = 1$ **to** n **do**
    - **for** $j = 1$ **to** n **do**
      - // Check that the overhearing link PDR > 0.6
      - **if** $c \neq j \bigcap (1 - p_{S_cD_j}) \geq 0.6$ **then**
        - stop $\leftarrow$ true
      - **end**
    - **end**
  - **end**
- **end**
- **else**
  - // Three flows or more case
  - Repeat the Else if part but for PDR > 0.4
- **end**

end

extension for our current work. Our solution will have to be extended in the following ways. First, we need to tag outgoing transmissions or notify destinations about who are the coding participants. Second, the additional constraint of what to code in addition to when to code significantly increases the search space. We defer such possibilities for the future.

2.5 Evaluating Our Framework

In this section, we evaluate PACE via (a) ns-3 simulations on large-scale topologies, and (b) experiments on our wireless testbed over various topological settings and bit rates. While we have run a very large set of simulations and testbed measurements, we discuss only a subset here. As a high-level observation, our assessment reveals that PACE wisely activates NC on a per data flow basis, thereby offering throughput improvements of as much as 350% in some specific use-cases.

2.5.1 Evaluating PACE via ns3 simulations

We consider both grid and random topologies with 20, 50 and 100 nodes. We have set the received signal strength threshold for correct decoding to be equal to -80 dBm for all nodes, while the data packet size is 1500 bytes. We employ the ns-3 Friis propagation model, and we consider the 802.11a mode of operation. The maximum distance between any two nodes is 300 meters; we assume that there is no mobility. In addition, in our simulations we select random senders and destinations, which typically are separated by multiple hops; we apply the AODV protocol (implemented in ns-3) for routing (our approach does not depend on the routing protocol in use). PACE is applied locally at every router, and native

packets may be encoded/decoded multiple times as they are forwarded along multiple hops along their route to the destination. We vary the number of data flows between 2, 5 and 10. We first evaluate PACE over fixed-rate topologies, and subsequently we consider multi-rate possibilities with rate adaptation.

In multi-hop scenarios, we assume that a routing algorithm is in place. Hence, a route for each flow is specified. For simplicity consider an intermediate node where two flows converge. The relay indicates the transmission rate for the preceding nodes (flows) by consider the following node (on the route) as temporary destination. In other words, the local topology is considered for whether or not to apply network coding at that hop. At each hop, independent decisions are made.

PACE offers benefits over blind NC application in large-scale topologies

We first consider a grid topology with a fixed rate of 12 Mbps. As shown in Figures 2.12, 2.13 and 2.14, PACE always offers a noticeable throughput gain, due to its ability to make correct decisions on whether or not to apply NC; the gains could be as high as 350% (in the 100 node case). The benefits increase as the scale of the network increases and the routes are longer; this is because the gains on each local hop add up and the more the opportunities, the higher the gain.

From Figures 2.13 and 2.14 we observe that higher bit rates lead to less modest performance gains with PACE. This is directly attributable to the degraded quality of links at the higher rates. As seen in the figures, NC itself offers more modest benefits compared to a traditional store-and-forward setting. Here, the increased likelihood of encountering poor overhearing link qualities, as well as poor incoming link qualities, often preclude the

use of NC. But PACE still makes the right decisions at each local router on whether or not to use NC.

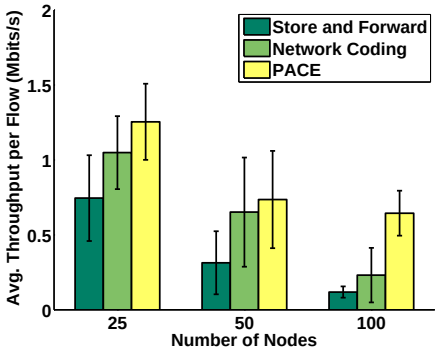

Figure 2.12: Simulation results when the transmission rate 12 Mbits/s.

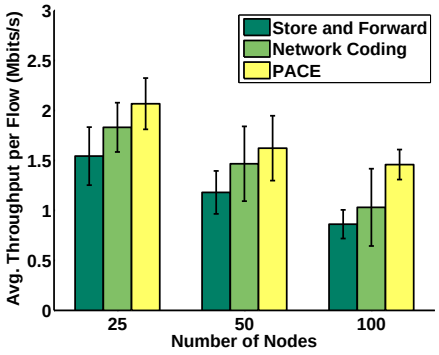

Figure 2.13: Simulation results when the transmission rate 36 Mbits/s.

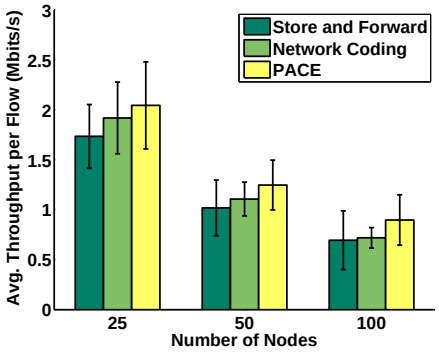

Figure 2.14: Simulation results when the transmission rate 54 Mbits/s.

The choice of parameters for PACE

Next, we show that the choices we make with PACE (PDR on overhearing links should be higher than 0.6 and the PDR on the incoming links should be greater than 0.4) are indeed the best choices in larger scale (random network) settings. Fig. 2.15 shows that if the threshold on the incoming link qualities is changed to 0.2 or 0.6, a wrong decision is made and could lead to up to a 3-fold degradation in throughput! Similarly, Fig. 2.16 shows that a wrong decision on the threshold for the overhearing link qualities could degrade the throughput by up to 3-times.

Sensitivity to the ETT probe size

As discussed, PACE relies on ETT probes, exchanged among neighbors to determine link qualities and average queueing times. In Figures 2.17, 2.18, we consider probe sizes of 256 and 512 bytes respectively. The throughput gains with the 512 bytes probe packet yields more accurate assessments of link qualities and thus, offers a higher throughput than

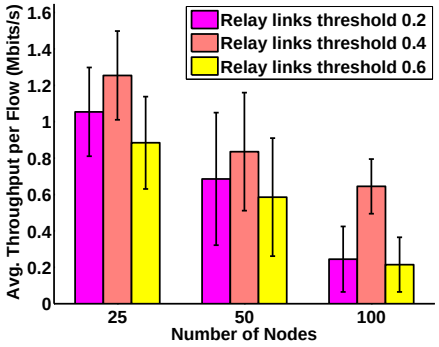

Figure 2.15: Varying the relay incoming threshold between 0.2, 0.4 and 0.6.

if a probe size of 256 bytes is used.

PACE with rate adaptation

Next, we evaluate the ability of PACE to jointly choose the policy (NC or store-and-forward) and the transmission rate to be used with the chosen policy. Since, this involves the "choice" of the right transmission rate, we compare PACE *with* NC and store-and-forward in conjunction with a popular rate adaptation algorithm, viz., the Adaptive Auto Rate Fall-back (AARF algorithm [95]). Fig. 2.19 shows that, PACE achieves a throughput gain of approximately 15% as compared to the NC case in the grid topology considered.

Evaluating PACE via testbed measurements

We have implemented PACE in Click [4] as an embedded software module in the NCRAWL platform [58]. We experiment on a wireless testbed deployed on the 3rd floor of

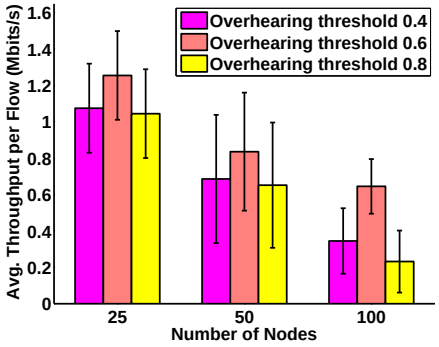

Figure 2.16: Varying the overhearing threshold between 0.4, 0.6 and 0.8

the Computer Science building at UC Riverside; the deployment is depicted in Fig. 2.20.

The nodes are based on the Soekris net4826 hardware configuration, and run a Debian Linux distribution with kernel v2.6 over NFS. Each node is equipped with a WN-CM9 wireless mini-PCI card, which carries the AR5213 Atheros chipset. Every card is connected to a 5 dBi gain external omnidirectional antenna. We experiment with the 802.11a mode of operation, in order to avoid interference from the collocated campus WLANs. All devices set their transmission powers to 16 dBm. We use fully saturated UDP traffic; the default data packet size is 1500 bytes.

Implementation Details

The proposed enforcement logic is implemented as a thin sub-layer within the MAC layer; at this layer, the PACE logic determines the best forwarding paradigm (network coding or store-and-forward), as well as the appropriate data rate to be used with the chosen paradigm. PACE is composed of three modules; (i) Link PDR measurement module, (ii)

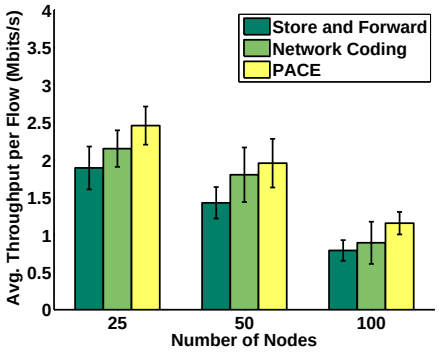

Figure 2.17: PACE vs. NC and store-and-forward for ETT message length of 256 bytes.

Policy Aware Decision Engine and (iii) Feedback module. The ***Link PDR measurement module*** is responsible for gathering information about the quality of the links incident on the node (PDR). Specifically, we modify the standard probing mechanism to incorporate ETT type probing [80]; this allows the module to estimate the PDR on each link at each possible transmission bit rate. Each node periodically reports the estimated PDR on each of its incident links to its neighbors. The relay node collects the reports and calculates the PDR for each link in its local topology (linear, x, or wheel as the case may be). The ***Policy Aware Decision Engine*** is the heart of our approach. It decides whether or not to invoke network coding. In addition, it selects the transmission bit rates that maximize the throughput given the current channel conditions (PDRs). The logic used by this module is a direct application of what was described in Section 4.4. The **Feedback module** is responsible for informing the transmitters about the chosen rates.

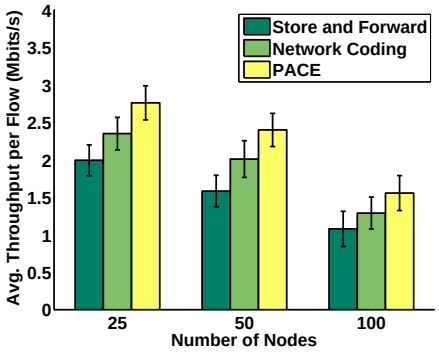

Figure 2.18: PACE vs. NC and store-and-forward for ETT message length of 512 bytes.

Experiments with fixed rates

We have performed testbed measurements on 'X'-type sub-topologies on our testbed. In these experiments, we change the qualities of the links to create 15 different sub-topologies. The testbed setup is the same as was described in Section 2.3. The results are averaged over different link PER values. Fig. 2.21 shows the average throughput with various tested transmission rates. We observe that PACE outperforms the blind NC and store-and-forward application strategies throughout. Since the experiments are over a single local hop, the gains over NC are modest ($\approx 10\%$).

Experiments with rate adaptation

Next, we consider a multi-rate case where PACE also makes decisions on the rate to use. In Fig. 2.22, we show testbed results for PACE in 5 different scenarios. Each scenario is created essentially by randomly choosing a set of links in our topology (with different link qualities). For NC, we choose the best rate (as predicted by PACE); however,

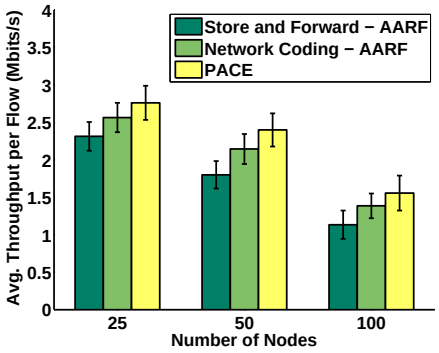

Figure 2.19: PACE performs better than the Adaptive Auto Rate Fall-back (ARF) algorithm.

no store-and-forward option is involved. PACE appropriately chooses between NC and store-and-forward, depending on the scenario. PACE achieves the same throughput as NC in the worst case scenario; this is a case where the best decision is NC almost all the time, and PACE makes that decision. However on average, it achieves 10% gain. The gain is again modest in these cases, since only a single hop is involved, and the best rate for NC is being used. However, it is important to note that in three out of the five cases, PACE offers a higher throughput. We have considered several other scenarios (with other sets of links) and we find that in about 40% of the considered cases, we observe gains with PACE (due to its properly choosing the store and forward option).

A comparison between PACE and COPE

Next, we evaluate PACE in realistic scenarios by comparing the average throughput with that of COPE or only network coding. We configured COPE to switch off network

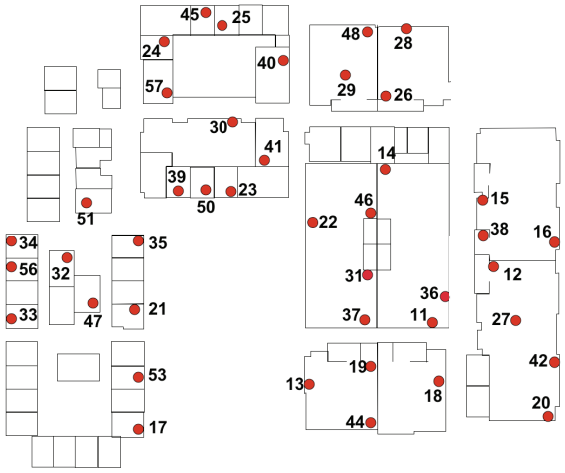

Figure 2.20: The deployment of our testbed at the University of California, Riverside. Nodes are represented by dots along with their IDs.

coding if the loss rate exceeds 20%, as in the default case [138]. The number of nodes are set to 100 and the results are averaged over 10 runs. Each run lasts for 5 minutes. One input flow consists of real video file traces obtained from [31]. Other short/long file traces were collected by downloading short and long files and capturing the traces using [30]. The file sizes are 200 KB and 50 MB respectively. These were also used as candidate input traffic traces. Fig 2.23 shows that PACE significantly performs better than COPE and NC for video and long file traces. On the other hand, for shorter files, the average throughput of PACE and COPE resemble that of using only NC. This is because with short files, the traffic loads are low/modest. Thus, the relays often do not find opportunities to encode. Thus, store-and-forward is used more often than not in all cases.

Evaluation of power consumption with PACE

PACE decreases the overall number of retransmissions compared to using either network coding or store and forward exclusively. Thus, PACE saves the consumed energy. In order to quantify the savings, we perform a set of experiments and record the total number of (re)transmission(s) in the following cases; (1) Store and Forward (2) Network Coding and (3) PACE. We compute the power consumed, on average, with our Soekris net4826 boxes with various transmission rates and payload sizes. Thereafter, we perform a direct mapping of the number of (re)transmissions to power consumption. We perform four sets of experiments during the day with the X-topology, each spanning an hour. In each experimental scenario, we reposition the nodes; we change the quality of the links in the topology at arbitrary instances in time (vary frequency) to characterize variations in conditions. The transmission bit rates are chosen as per the strategy. Fig. 2.24 shows the power consumption of PACE compared to that with store and forward and network coding. We find that in all sets of experiments, PACE outperforms both network coding and store-and-forward in terms of the power consumed. In scenario 1, the links were primarily of good quality and we did not change the quality of the links often; thus, the gains over pure network coding are modest. In scenario 2, we varied the quality of the links with the highest frequency; PACE invokes network coding or store-and-forward as appropriate and thereby, decreases the overall number of required transmissions. It therefore provides significant power savings 20% as compared to NC. The frequency with which we changed the qualities of the links in scenarios 3 and 4, were less than that in scenario 2 but more than that in scenario 1; thus, as one might expect, the gains are less than that in scenario

2, but slightly higher than that in scenario 1. Note here that, if the occurrence of poor quality links becomes prevalent, all schemes require higher numbers of retransmissions and thus, experience higher power consumptions.

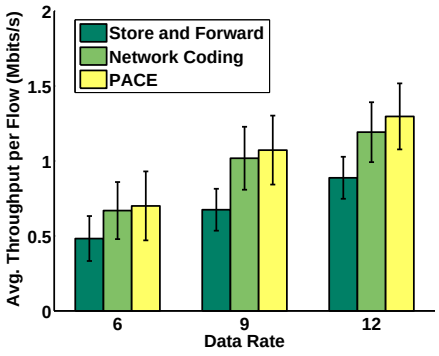

Figure 2.21: Experimental results in the X-topology for different transmission rates.

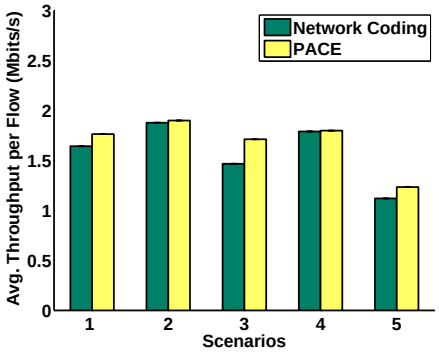

Figure 2.22: PACE offers the highest benefits at all different rates in the X-topology.

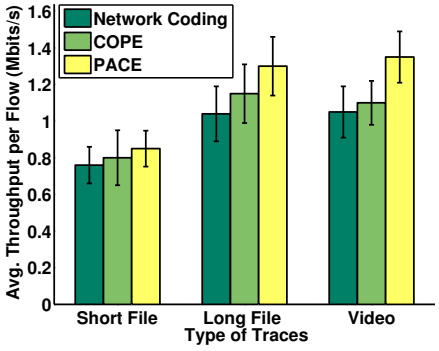

Figure 2.23: A comparison between PACE, COPE and NC for different types of traces.

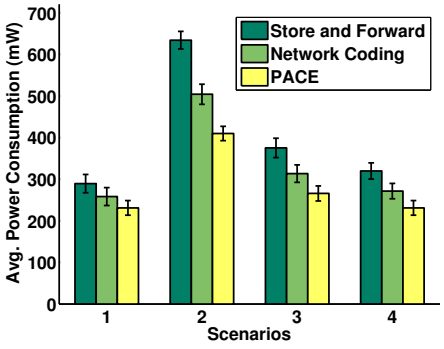

Figure 2.24: Avg. power consumption for different channel conditions scenarios.

2.6 Discussion

In this section we discuss certain design aspects of PACE.

Focus on single-router topologies: In this paper, we primarily focus on local wireless NC settings involving a single router; packets encoded by a router are decoded at the next hop. Such scenarios are prominent in WLAN deployments [89], where clients associated with the same access point (AP) exchange data; examples of application include WiFi-based home networking and applications such as online gaming and video streaming [219, 218]. In such cases, the AP essentially plays the role of the relay encoding packets exchanged among its clients. We envision that practical wireless NC will mostly be applied in such topologies, given the ease of the decision making process and the simplicity in gathering topological information at the relay site. However, we also show that this does not preclude the use of PACE in large-scale ad-hoc and mesh topologies with multi-hop routes. In particular, in multi-hop settings a packet may be encoded and decoded multiple times as it traverses multiple relays (depending on the topology) along its route to the final destination. As long as an encoded packet does not traverse more than one hop, our work is directly applicable in such settings as well: PACE will make local NC decisions at the individual routers. We plan to extend PACE for more complex topologies, such as the butterfly [139], in our future work.

The applicability of our framework with other NC architectures: We have implemented the PACE logic as part of the NCRAWL software platform [58]. We use NCRAWL given its lightweight implementation and its ability to efficiently support multi-rate experiments. Clearly, certain NC related choices of other platforms, such as COPE

[139], are compatible with our scheme. For example, the decision on which packets are to be encoded together and when, generally depends on the relay configuration; it could also depend on other factors, such as the traffic profiles and network policies. Since such decisions are orthogonal to our study and we simply adopt the same assumptions for such procedures from prior NC frameworks such as COPE; for example, we use a fixed pre-configured value for a timer upon the expiration of which, temporarily stored packets are examined, de-queued and coded together to form encoded packets at the relay. Since PACE is independent of these procedures, it is directly applicable with other NC architectures and solutions [139, 89, 206].

Multiple Relays (M): Solving the problem for M multiple relays (simultaneously setting a set of sources to a set of destinations) is challenging for the following reasons; (1) If each of the M relays transmits information without coordination, it could lead to wasteful transmissions (more like a broadcast). (2) If the source has to choose among the M relays (to choose a single relay), a new design that accounts for factors such as the load on each relay, will be necessary. (3) If instead, the relays were to coordinate of themselves, the coordination across relays to maximize coding opportunities requires communications between relays and thus, incurs overhead. The overhead versus trade-off benefits need to be studied. In all of the above cases, we believe that a different and more complicated system design is needed. Specifically, if one is not careful, there may be delays incurred in sending encoded packets, due to channel occupancy for control information exchange as mentioned above.

2.7 Conclusions

In this paper, we argue that when NC is applied in a careless manner, it may cause significant throughput degradation in multi-rate environments. In many cases, a traditional store-and-forward approach may be preferable to NC. Via extensive experiments and an analysis, we characterize the regimes where NC offers throughput benefits and those where it does not. This study allows us to formulate a set of guidelines regarding when NC should be applied. Based on these guidelines and our analytical model, we design PACE, a policy-aware coding enforcement logic, which allows a router to switch between NC and store-and-forward modes depending on the link qualities. We evaluate PACE both on a simple prototype testbed and via extensive simulations. Our evaluations show that PACE could potentially offer network-wide throughput improvements of up to 350% as compared to a fixed rate NC policy that is blindly applied.

Chapter 3

Effective Power Line Networking in Multi-flow Environments

3.1 Introduction

Power line communications are attractive for providing backhaul Internet connectivity in settings without an in-built network infrastructure, especially in third world countries. Several retail segments such as healthcare, industrial automation and warehousing, are increasingly relying on Internet connectivity [188], [181] and in many such segments, deploying an Ethernet backhaul from scratch may not be cost effective. Furthermore, in many such applications, there is a large amount of local, peer to peer data transfers that are required (e.g., health records between medical devices and IT servers *and* remote patient monitoring feeds in hospitals, video surveillance data between cameras and storage servers in retail warehouses) for which, PLC can be attractive.

PLC research is yet to mature: While PLC commercial adapters are now available for home applications [8], they primarily target low throughput unsaturated traffic flows. The viability of using PLC as an alternative for delivering Ethernet like throughputs in multi-flow settings has received little attention. Recently, there has been some work done on understanding the efficacy of the 1901 MAC (Medium Access Control) protocol, which is the basis for access control in commercial PLC adapters [250, 251, 253]. These efforts showcase the short comings of 1901 and suggest the tuning of a few protocol parameters towards improving its efficiency. However, tuning of such parameters on commercial adapters is not viable today. Moreover, 1901 has other performance issues that we showcase later. Thus, we ask the question "How can we achieve high stable throughputs with PLC in multi-flow settings in a completely standards agnostic manner?"

Challenges: There are three main challenges that we will need to overcome to answer the above question. **First,** the topology of a PLC network is often unknown since it is hidden behind walls and the connectivity is typically established without communications in mind; in fact, nodes that are geographically close are not necessarily direct neighbors. The network structure dictates which transmissions interfere with each other. While the 1901 MAC resolves this issue to some extent, it can lead to poor throughput as well as unfairness in many cases. To drastically reduce the ill effects of interference, an understanding of the network topology needs to be derived. **Second,** the quality of the PLC channel is time varying. The impedance loads on the PLC lines vary as electrical devices that are plugged in, and are turned ON or OFF. This causes the throughputs on certain links to either degrade or improve, thereby causing dynamics in the network topology. The time scales of these

dynamics will influence the solutions that one can deploy for effective management of flows. As shown later, the 1901 MAC does not account for these variations and this contributes to its inefficiency. The **third** related challenge is to resolve the first two challenges in a standards-agnostic way. This requires lightweight solutions above the MAC layer that are sufficiently adaptive so as to cope with the PLC channel dynamics.

Contributions: (1) Towards addressing the above challenges, we undertake an extensive application (flow) level measurement study. While our study leads to many interesting observations, two are especially noteworthy. First, the study shows that flows do indeed interact in a PLC network in unpredictable ways. Some flows can co-exist simultaneously, and their joint activation can yield significant throughput gains compared to when they are activated in isolation. However, in other cases, joint activation of flows can hurt the throughput compared to when they are sequentially activated. Second, the study suggests that the PLC channel is *quasi-stationary*. This is an artifact of these variations arising from electrical devices being turned on/off, which does not happen at very fine time scales and all that often.

(2) Based on the understanding gained from the above study, we design *BOLT*, a flexible framework that appropriately configures the PLC network to derive high throughputs while adhering to whatever fairness constraints are desired. Towards this, *BOLT* does the following:

- Determining flow interactions via calibration measurements can be expensive; if done naively it can result in an exponential number of measurements. *BOLT* imposes a logical structure on the PLC network using which, it is able to drastically reduce the number

of such measurements needed for flow management (now polynomial with respect to the number of nodes in the network).

- *BOLT* leverages the quasi-stationarity of the PLC channel to intelligently apply machine learning (ML) classifiers to determine flows that can be simultaneously active, and their potential throughputs. The approach (i) has high accuracy with small amounts of training data and (ii) is resistant to noise from the plugging in of electrical devices.
- Finally, based on the above estimation, *BOLT* aggressively reuses the available channel capacity by scheduling as many simultaneous flows as possible (at the granularity of time epochs) while adhering to desired fairness constraints. This scheduling is done at the application layer and the goal is to limit the level of contention that has to be handled by the 1901 MAC. This will reduce collisions due to hidden terminals or unnecessary backoffs, both of which if not dealt with, lead to loss in throughput and unfairness.

(3) We extensively evaluate *BOLT* on three different testbeds to show that it provides several fold throughput improvements (1.5x - 8.5x) over the state of the art solutions (including those in current day adapters), while remaining standards-agnostic. Further, we show that the algorithms within *BOLT* strike a good balance between performance and complexity.

3.2 Background and Related Work

In this section, we first provide relevant background on PLC. Subsequently we discuss related studies.

PLC Channel: PLC operates in the 1.6 and 86 MHz bands. Its channel [108]

is similar to the wireless channel: (a) the signal is attenuated due to cable losses and, (b) multipath propagation occurs due to cable branching and unmatched line ends. The multipath reflections depend on the electrical devices hooked on to the network and their impedance. Studies such as [276] and [151] have built channel models for indoor PLC. However, they do not measure or characterize the properties of interest such as (i) the interactions between flows in terms of simultaneous activation and (ii) the time scales of throughput variations due to channel dynamics.

PLC Adapters: Today's popular PLC adapters are Homeplug AV1, Homeplug AV2 and HomePlug GreenPHY [8]. Using 1155 OFDM subcarriers and turbo codes, AV1 operates nearly at the theoretical maximum rate (ranges from 14 to 200 Mbps). AV2 uses an additional bandwidth from 30-86 MHz and a MIMO like PHY to achieve data rates of up to 1.5 Gbps. The MAC protocols used in these adapters, are variants of 1901 which in turn is based on CSMA/CA [252]. Typically, each PLC network has a central controller that helps impose a time-slot structure to facilitate the use of 1901.

Related Work: The work in [252] models CSMA/CA for PLC. In [124], the authors perform limited experiments to gather some insights on the throughput of UDP and TCP over a PLC network. The authors in [182], try to characterize the end to end throughput over PLC. They conducted a study on PLC RTT across various traffic types. In [269] the authors use PLC to provide synchronization capabilities for wireless networks. These limited efforts however, have not explored multi-flow environments or mechanisms to improve the PLC system performance.

There is recent work on analyzing the throughput performance of the PLC MAC

protocol (based on IEEE 1901) [253, 251, 250]. The authors show that collisions and unnecessary backoffs are worse with 1901 as compared to 802.11 [250]; this is because (i) of the large slot duration and (ii) nodes increase their backoff times when they sense the channel to be busy (unlike in 802.11), with 1901. The authors argue that by tuning certain back-off parameters, the throughput of 1901 can be maximized. However, setting these parameters in commercial devices with no access to the firmware is difficult. Furthermore, the analysis does not take into account the loss in performance due to dynamics of plugged in electrical devices. Our goal is to design and develop an adaptive framework that allows flexibility in such scenarios, and reduces MAC contention, by imposing a schedule at a higher layer.

Note that while techniques from wireless (PHY, MAC) measurement studies (eg. [210]) can be borrowed and applied in the PLC context, this would (i) require support from the PLC adapters (not available today) and (ii) more importantly, result in changes to the access protocol (the 1901 standard) to address the previously mentioned issues.

3.3 Understanding PLC

To understand the factors that influence PLC throughputs, we undertake an extensive measurement study with commercial PLC adapters in a multi-flow environment. Our goal is not to characterize the PLC channel from the physical layer perspective as in prior efforts, but to understand how channel dynamics and flow interactions influence throughputs from an application layer perspective. Our measurements are at the granularity of flows; flows are from a source to a destination and could potentially encounter

multiple power outlets en route. Flow contention is handled by the MAC protocol (1901 based) in all of the experiments reported here.

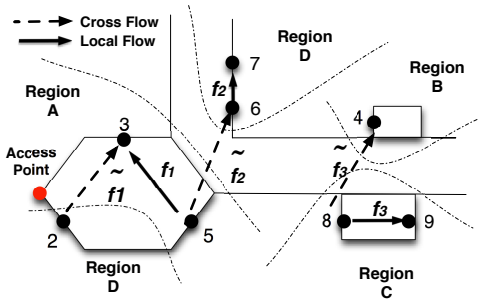

Figure 3.1: Illustration of a PLC topology.

We perform experiments on three different testbeds. The first is at an enterprise office setting (ENT), the second is at a university lab (UNI testbed), and the third is in a residence (house). Details of the testbed setups are in Section 5.7. We employ PLC adapters from multiple vendors for diversity.

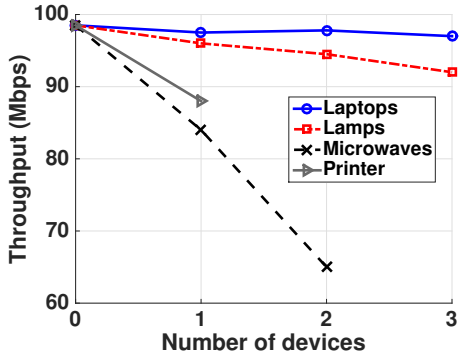

Figure 3.2: Impact of electric apparatuses.

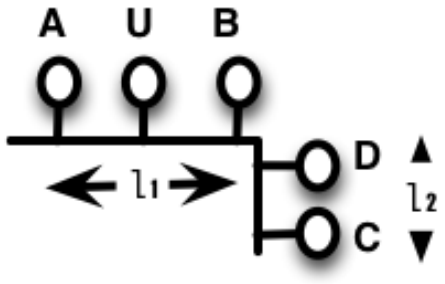

Figure 3.3: Controlled Topology

	Mircowave OFF		Mircowave ON	
	Isolation	Contention	Isolation	Contention
τ_{l1}	95	85	29	29
τ_{l2}	96	75	92	63

Table 3.1: Variation in throughput due to connecting and disconnecting devices at outlet U.

How do electrical appliances affect the throughput of a PLC network?: Switching on electric apparatus (plugged into power outlets in the PLC network) injects noise onto the channel [213], which degrades the throughput. Towards quantifying the performance degradation due to plugged in electrical devices, we first perform a controlled experiment where we vary the number of electrical devices that interfere with PLC transmissions. In Fig. 3.2, we plot the throughput of a flow in the presence (or absence) of different apparatuses. The distance between the source and destination was $\approx$ 2 m. The electric apparatus (fluorescent lamps, Dell laptops, small microwave ovens, printer) are connected to a power outlet that is 80 cm from the destination power outlet. Using *iperf* [22], a UDP flow was generated between the source and the destination. It is hard to a priori know the capacity of links in a PLC network; wire material and age will dictate the

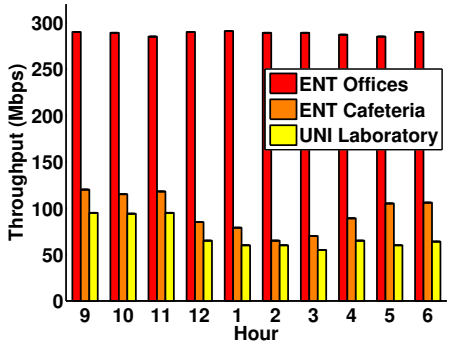

Figure 3.4: Dynamics over hours.

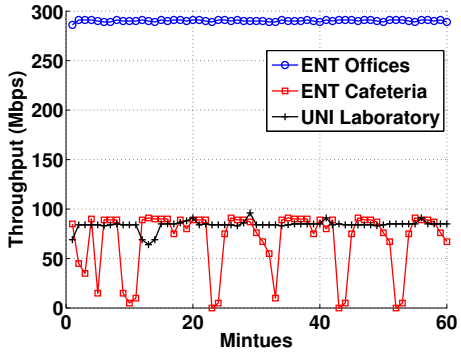

Figure 3.5: Dynamics over minutes.

maximum throughput of a line. Thus, we start with a relatively high UDP flow rate (500 Mbps) and gradually reduce the rate until we observe no losses (this is the estimated line capacity). The results are averaged over 20 runs, each lasting 60 seconds. As seen from the figure, lightweight devices such as fluorescent lamps or laptops do not inject much noise and thus, barely hurt the throughput. However, microwave ovens or printers (HP LaserJet 4200 in our study) are heavyweight in terms of the noise they inject and significantly hurt the throughput. It is interesting that as the number of devices of a certain type increase,

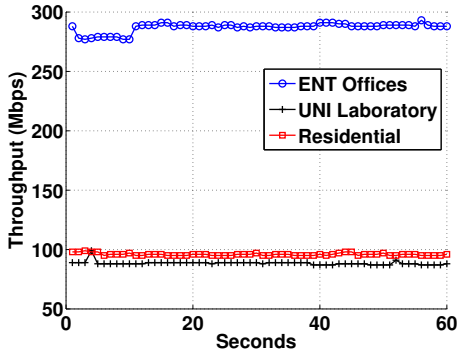

Figure 3.6: Dynamics over seconds.

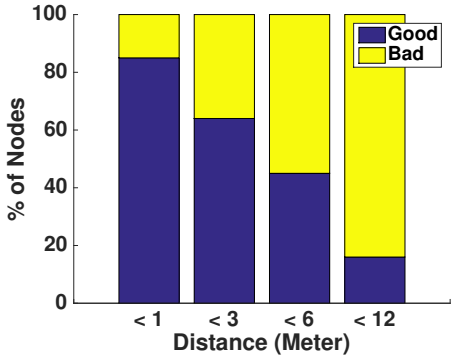

Figure 3.7: Proximity versus throughput

the throughput degradation appears to be linear; thus, after an initial calibration phase, the impact of additional devices (of the same type) can be empirically estimated (although we do not use this property explicitly in *BOLT*). The key takeaway here is, the fact that different brands/types of electric apparatuses project different levels of noise (based on their electric load), makes it challenging to predict the throughput of a flow.

Microscopic study: Towards understanding the impact of electrical appliances on 1901, we construct a controlled topology with five power outlets as shown in Fig. 3.3 in a

residency; sources A and C, and destinations B and D, are connected via AV2 adapters to the outlets as shown. Power outlet U is unused in one set of experiments (setup 1); in a second set of experiments (setup 2), an active microwave oven (power 650 watts) is plugged into U (similar effects were seen with our HP LaserJet 4200 printer but the results are not shown because of space constraints). The results from the experiments are tabulated in Table 3.1. The average throughputs of the flows from A to B and from C to D in *isolation* are 95 Mbps and 96 Mbps, respectively in setup 1. In setup 2, they are 30 Mbps and 92 Mbps respectively. It is evident that the flow from A to B is significantly affected by the microwave oven, but not the one from C to D. When the two flows are activated *simultaneously* in setup 1, the throughputs achieved by the flow from A to B and that from C to D, are 85 Mbps and 78 Mbps, respectively; notice that the 1901 MAC decreases the throughputs of the individual flows, but provides some semblance of fairness. The overall throughput is higher and thus, this may be desirable. However, with setup 2, while the throughput of the flow from A to B remains almost unchanged compared to isolated operations (29 Mbps), the flow from C to D takes a significant hit (throughput drops to 63 Mbps) during concurrent operation. Thus, even though the microwave by itself does not influence the latter flow, the interactions with the flow that is affected degrades its performance when 1901 is used (the two flows share the capacity and the flow affected by noise eats into the capacity of the flow that is relatively unaffected). Specifically, if the poor quality flow accesses the channel, it can cause the good quality flow to repeatedly back off. The overall throughput is still higher than if the flows were sequentially activated but this may not always be the case (as seen in later experiments in uncontrolled settings). Thus, there is a need to account for these

interactions, and depending on whether the throughput increases or decreases, switch from concurrent activity to sequential activity or vice versa when electrical devices are turned ON/OFF.

How can one expect the throughput on a PLC channel to change with time?: Figs. 3.4-3.6 presents the dynamics in the throughput measured over different time scales in various settings. The data is from 150 to 250 arbitrarily chosen flows in three different environments (other flows exhibited similar behaviors): ENT offices, ENT cafeteria and a UNI laboratory. In Fig 3.4, we show the average throughput of the considered flows over hours. The noise due to devices in ENT offices is extremely low compared to that in the other cases; this is because the electrical equipment here mostly consists of lamps and laptops. In the cafeteria, the throughput is generally lower due to more heavyweight appliances (e.g., refrigerator) but is generally steady. However, a throughput degradation is noticed (Fig. 3.4) due to the usage of devices such as the microwave oven, coffee maker, etc. during the lunch period. The throughput in the UNI lab is again lower compared to the ENT offices due to the presence of devices such as printers and heavy duty servers which are typically on. After about 11:00 am, when students start using their research equipment (e.g., switch on their desktop machines or other servers) a throughput degradation is seen.

A minute-by-minute depiction of channel fluctuations during the lunch period at the ENT cafeteria (12:00 - 1:00 pm) is shown in Fig. 3.5. We see that while the average throughput is similar to that in a relatively static setting (UNI lab), operation of electric apparatus can cause significant variations in noise and therefore the achieved throughput,

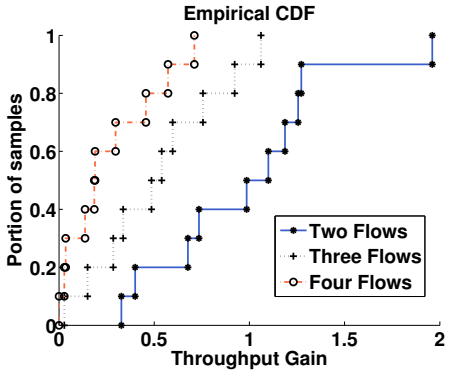

Figure 3.8: Concurrent invocation of flows.

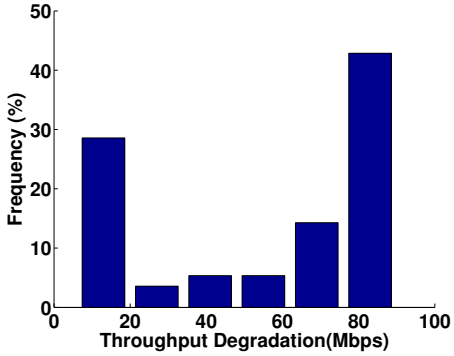

Figure 3.9: Contention hurts throughput.

at these short time scales (order of minutes). Our experiments did not reveal any variations at even finer time scales (seconds or milliseconds). We conclude that in general, the PLC channel experiences only slow fading and is *quasi-stationary* in nature unlike wireless (with fast fading); however, it can exhibit large variations in short time scales (order of minutes) in dynamic environments during certain times of the day.

Does proximity between the transceiver pair imply high throughput?: Next, we measure the throughputs between all possible pairs from our 16 node UNI testbed.

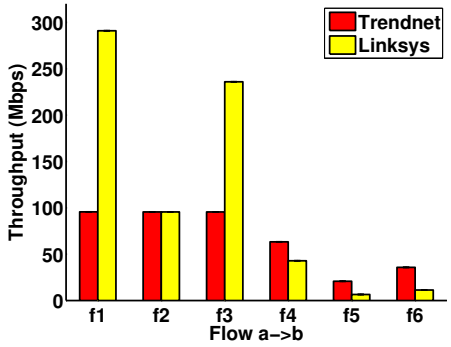

Figure 3.10: The effect of different adapters brand on the average achievable throughput.

We measure the geographical distance between a pair of outlets, and the throughput between the outlets in isolation. We classify the throughputs into two categories: HIGH if it is $> 80Mbps$, and LOW otherwise (the line capacity was $\approx$ 130 Mbps). From Fig. 3.7 we see that proximity does not always translate to a high throughput. More importantly even distant nodes could enjoy high throughputs (e.g., when they are more than 6 or even 12 meters apart). This demonstrates that one cannot determine which transceiver pairs are likely to be adjacent in the network topology just based on geographical proximity (e.g., as will be the case in wireless). Note that wiring diagrams of buildings do not reflect the actual electric connectivity because of: (i) connecting different apparatuses changes the actual electric load (ii) repairs and updates of the electrical wiring constantly change the wiring diagram.

Do flow interactions increase or decrease throughputs with 1901 ? We consider our ENT setup with interacting regions, shown in Fig. 3.1. It is obvious that flows that are logically separated in the PLC network (don't interact) can be simultaneously

Individual Throughput		
f_1	f_2	f_3
291	95.6	236

$\tilde{f}_1$	$\tilde{f}_2$	$\tilde{f}_3$
42.9	11.8	11.3

LZF (pairwise)			
	f_1	f_2	f_3
f_1	NA	379	331
f_2	379	NA	221
f_3	331	221	NA

CZF (pairwise)			
	$\tilde{f}_1$	$\tilde{f}_2$	$\tilde{f}_3$
$\tilde{f}_1$	NA	30.9	18.39
$\tilde{f}_2$	30.9	NA	11.6
$\tilde{f}_3$	18.39	11.6	NA

Table 3.2: Throughputs (Mbps) for Fig. 3.1.

active (in fact, 1901 does take care of this). The scenarios of interest to us are smaller constrained areas, where flows interact at the MAC layer. We examine if 1901 effectively handles contention in such cases.

We concurrently establish (i) two (ii) three and (iii) four randomly chosen sets (S) of flows (200 sets in each case) and examine the aggregate throughputs that are achieved. For the concurrently chosen flows, the transmitters and the receivers are distinct. We only establish flows between transceiver pairs that can directly receive signals from each other. The aggregate throughput of each *set* of flows is averaged over 10 runs, each lasting for 30 seconds. All packet sizes are equal to 1480 bytes. Fig. 3.8 shows the CDF of the aggregate throughput gain of operating the set S of flows concurrently relative to the sum of their individual average throughputs i.e., $\frac{\tau_S}{(\frac{\sum_{i \in S} \tau_i}{|S|})}$, where τ_i is the throughput of flow i when operated in isolation and τ_S is the aggregate throughput when the flows in set S operate concurrently. We make many interesting observations: (i) Operating three flows provides a marginal throughput gain of about 8% in only 10% of the scenarios. More importantly, invoking three or more flows concurrently almost always leads to a loss in throughput (gain

< 1) compared to invoking the flows in isolation. Note that this will be the case with current PLC solutions, where there is no flow regulation at the application/network layer and 1901 arbitrates channel access. (ii) Operating a pair of flows leads to a throughput gain in 60% of the cases, with 40% of the cases yielding over 25% gain and 10% of the cases even yielding over 50% gain. (iii) At the same time, even with pairs of flows, there are several instances (40% of the cases), where operating the flows in isolation is better. These trends are consistent across different PLC networks that we consider (UNI and ENT) and also across different adapters (results omitted due to space constraints).

Achieving spatial reuse while avoiding pitfalls of contention: The higher aggregate throughput from operating *pairs* of flows concurrently indicates that there is room for spatial reuse in PLC contention domains (similar to wireless). However, with an increased number of flows (≥ 3), this opportunity disappears. Note that the 1901 MAC uses a 2-level backoff scheme and larger slot times, and these contribute to inefficiency in its contention process; the impact is exacerbated with an increasing number of flows. This was analyzed in depth by [250]. Another potential cause for the throughput degradation with increasing number of simultaneous flows is hidden terminals; here, two transmitters who cannot hear each other, transmit simultaneously to cause collisions at a receiver. Unlike in a wireless setting (e.g., WLAN) where hidden terminals are often in the proximity of each other, they could be at arbitrary unknown locations in the PLC topology. To illustrate the ill effects of hidden terminals, we form a line topology of four nodes. We choose a fixed flow (from node 1 to 2), and initiate a second flow (from node 3 to 4) concurrently; the second flow may contend with the first flow, and depending on the relative positions of the transceivers may project

a hidden terminal. From Fig. 3.9, we see that in $\approx$ 42 % of the cases, the throughput degrades significantly (by 80 Mbps) and largely results from hidden terminals (owing to the topology). In summary, opportunities exist for spatial reuse but concurrent flows must be carefully orchestrated to avoid throughput degradations.

Impact of Adapter Diversity: To ensure the generality of our observations, we experiment with commercial PLC adapters two different brands: TRENDnet (TPL-401E2K) and Linksys (PLEK500). Both adapters are IEEE 1901 compliant. In addition, we also investigated the impact of compatibility issues between adapters of the two brands on PLC performance. TRENDnet is based on Homeplug AV1 specification, while Linksys is based on Homeplug AV2 that offers higher peak data rates. We found both these adapters to perform consistently with the previously discussed results. However, the individual and aggregate throughput values vary across brands. There is no single adapter that performs the best in all scenarios. We found Linksys to achieve higher throughput in most scenarios. However, when the packet error rate was high, TRENDnet's throughput was higher. Fig. 3.10 shows that Linksys does better or equal to TRENDnet for high-quality links ($>$ 50 Mbps) while TRENDnet's performance is better in low-quality links. When both TRENDnet and Linksys adapters are jointly used for a flow, we found no compatibility issues (as dictated by their specifications). However, their achievable throughput is governed by the minimum of the achievable throughput of both adapters (governed by their specifications). For instance, for high-quality links, the achievable throughput is limited to that provided by TRENDnet. Since it is hard to determine vendor-specific information with regards to adapters, these studies suggest that a measurement based learning approach is probably

best when trying to manage flows within a PLC network.

3.4 Zoning the PLC network

For ease of management, we subdivide the PLC network into what we call zones. A zone is similar to a contention domain in a wireless network, but as viewed at a higher layer; transceiver pairs with common nodes, that can sustain high throughputs belong to the same zone. If flows in two zones do not interfere with each other, then the zones can simultaneously carry flows with 1901. However, if they interfere with each other, then the arbitration of channel access across the two zones is currently handled by 1901. In the following we consider such *coupled zones* and show that spatial reuse could be possible in such zones as well.

Zones: We define a flow, f_{ij} as a stream of application packets that originates from a sender, i, towards a destination, j. A zone, $\mathcal{Z}$, is defined as a subset of nodes, $n \subseteq N$, such that the average throughput, τ, between its members is $\geq \alpha$. Therefore, a node, $i \in \mathcal{Z}$ iff $\tau_{ij} \geq \alpha$, $\forall j \in \mathcal{Z}$ $(j \neq i)$. We say that two zones $(\mathcal{Z}_1, \mathcal{Z}_2)$ are *loosely coupled (LC)* if there are no common nodes between them and the throughput between any node in the first zone and any node in the second zone is less than a threshold, η, i.e. $\tau_{ij} \leq \eta$, $\forall i \in \mathcal{Z}_1, j \in \mathcal{Z}_2$ $(0 < \eta << \alpha)$. Loose coupling implies that flows that are fully contained within the two LC zones, do not strongly influence each other and can probably be active simultaneously to provide spatial reuse. In contrast, if $\tau_{ij} > \eta$, we say that zones i and j are strongly coupled (SC) or just coupled.

Zones can capture the throughput influence flows have on each other based on their

mutual contention/ interaction on the physical medium. While we discuss how to create and maintain zones (Section 3.5) and choose α (Section 5.7) later, we reiterate here that a simple clustering of nodes based on proximity to form zones, is inappropriate as shown earlier.

Classifying flows: Given the zones, the PLC flows can be classified into two categories: local zone (LZF) and cross zone flows (CZF). A flow is local if both the source and sink nodes belong to the same zone, while a flow is considered a CZF, if the source and sink nodes belong to different zones. In Fig. 3.1 (our ENT testbed) each region (see dashed lines) corresponds to a zone. The throughputs obtained by various flows within and across zones is presented in Table 3.2. we see that LZFs tend to exhibit higher throughputs than CZFs. When two flows minimally impact each other, we call them *disjoint flows*. Disjoint flows can be invoked concurrently (reuse) to yield higher aggregate throughput than the average of the individual flows (as seen in Table 3.2). For example, if f_1 and f_3 are concurrently active they yield an average throughput of 331 Mbps as opposed to 263.5 Mbps when they were sequentially invoked. However, this is not the case for non-disjoint (coupled) flows; for example, concurrent invocation of $\tilde{f}_1$ and $\tilde{f}_3$ (Table 3.2) yields an average throughput of 18.39 Mbps which is lower than the 27.1 Mbps that is achieved if they are activated sequentially. We point out here that the automated formation of zones and identifying disjoint flows is non-trivial (we do this with *BOLT* next).

3.5 *BOLT*: System Design

In this section, we describe *BOLT* and its component functions. *BOLT* orchestrates an appropriately chosen set of concurrent flows within a single PLC network (at the granularity of time epochs) to enable high throughput. It overcomes the negative effects that arise with 1901 based MAC protocols during periods of high contention (as discussed earlier), by controlling the flows that are simultaneously active. In brief, *BOLT* leverages the concept of zones, and tries to aggressively activate disjoint flows together, while separating strongly coupled flows in time. It is flexible and can incorporate any desired fairness requirement. It is also adaptive to the restructuring of the PLC topology due to electrical devices being turned ON/OFF. It sits at the application layer and is standards agnostic i.e., works with any off-the-shelf PLC adapters; thus, it is readily deployable today.

Challenges: Building *BOLT* in practice has a number of associated challenges. Specifically: **(i)** How do we form zones? Ideally, the zones should include all the nodes that satisfy the requirements defined earlier (throughput between each pair within a zone should be $\geq \alpha$). However, in practice, forming ideal zones may be hard and the overhead for doing so may be prohibitive. **(ii)** How do we determine which flows can be activated simultaneously, and which ones should be isolated from each other ? We need to do this with a small set of training measurements. In addition, the measurements could be noisy

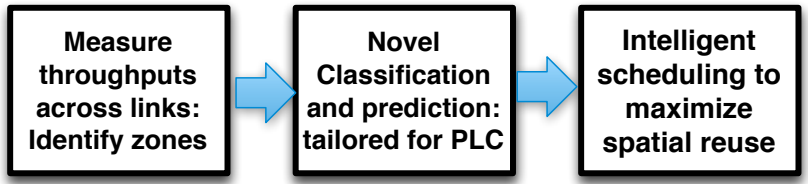

Figure 3.11: High Level Operational View of *BOLT*

due to switching on/off of electrical apparatuses interim. **(iii)** Given the flows that can concurrently be activated, how do we optimally schedule them to maximize spatial reuse ? **(iv)** How do we handle the dynamics of the PLC channel?

3.5.1 Overview

BOLT is built with the above challenges in mind. The quasi-stationary nature of the PLC channel allows *BOLT* to adopt a learning approach to accurately predict the throughput due to interactions between pairs[1] of contending flows with a small set of training measurements. It then efficiently schedules multiple application layer flows in real-time towards achieving very high spatial reuse. *BOLT* is primarily implemented at a central coordinator (CC), which is essentially one of the PLC nodes. *BOLT*'s operations consist of the following (continuously executing) phases (see Fig.3.11):

(1) Training phase: The CC collects *individual* flow measurements and uses them to construct zones efficiently. This allows the CC to later determine a set of disjoint flows that can be active simultaneously across loosely coupled zones. Subsequently, an additional small set of real-time throughput measurements are performed on *pairs of flows* between nodes chosen from pairs of strongly coupled zones. These measurements are then input to the next phase to determine which flows in SC zones can also be concurrently activated.

(2) Contention inference phase: Using the measurements from the training phase, features that capture flow interactions are intelligently identified. Subsequently an

[1]Based on our measurement study, we do not consider combinations beyond flow pairs in SC zones (as they degrade performance).

appropriate set of machine learning classifiers are used to accurately (i) classify flow pairs into those that yield a higher aggregate throughput during concurrent operation (termed *reuse-friendly*) and those that do not, and (ii) predict the resulting aggregate throughput for flow pairs that are reuse-friendly.

(3) Scheduling phase: Next, *BOLT* invokes an efficient, yet flexible, scheduling algorithm at the CC to determine the set of application flows that should operate concurrently at the granularity of time epochs. The goal of the scheduler is to maximize reuse in the PLC network, while at the same time, ensuring fairness between flows (across epochs). The schedule is disseminated to the relevant PLC nodes using the same PLC channel or a side-channel.

Handling Dynamics: The addition or removal of new users or electrical appliances or changes in the flow throughput, impact zone construction. This information is fed back to the CC in *BOLT* every epoch (T_e); in response, CC implements scheduling changes. However, the flow configurations (classification and prediction) themselves are updated at coarse time scales of tens of seconds or minutes (T_c); this is sufficient to handle the dynamics in the PLC channel.

3.5.2 Measurements to train *BOLT*

The first phase of *BOLT* involves performing a small set of measurements. The measurements serve two purposes: (a) identifying zones and (b) capturing flow interactions.

Identifying Zones in the PLC Network: Using the notion of zones reduces overhead by restricting flow interaction measurements (needed for training). It also helps readily identify opportunities for reuse (disjoint flows). Recall that zones consist of a group

of nodes such that the throughputs achieved between any pair within the group is higher than a threshold. This threshold, α, implicitly dictates how large the zones will be. Ideally, given a value of α, the zones should correspond to the largest cliques such that the throughput between the members of each clique is $> \alpha$. This reduces the number of zones formed and the consequent flow interaction measurements. Towards determining the zones, based on the measurements, the central controller forms a graph $G = (V, E)$ with all the nodes in the vertex set V; an edge exists between two nodes if the flow between them yields a throughput $\geq \alpha$. Now, the goal is to find the minimum set of cliques that will cover all nodes; this will ensure that the largest cliques are classified as zones. Unfortunately, this maps to the NP-hard, *vertex clique cover* problem, whose goal is to find the minimum set of cliques that cover all the vertices in a graph. Thus, we leverage a well-known lemma [258] to design a simple, greedy algorithm for zone construction that yields good performance in practice.

Lemma 1 *The chromatic number of a graph is equal to the minimum number of co-cliques (cliques in the complement graph) needed to cover the vertices of the graph.*

The chromatic number refers to the minimum number of colors needed to color a graph. Using the above lemma, we first construct the complement of graph G, namely G'. Then we color the vertices of G'. Now, vertices (nodes) belonging to the same color in G' form a single zone (clique) in the original graph G. Thus, we will have as many zones in G as the number of colors needed to color G'. However, graph coloring is itself a hard problem. Hence, we employ the popular greedy algorithm (Welsh-Powell algorithm) for coloring, which at each iteration, picks the vertex (among un-colored vertices) with the highest degree and assigns

it the smallest color (number) that is not used by any of its neighbors. This algorithm is known to use at most $\max_i\{\min\{\delta_i + 1, i\}\}$ colors, where δ_i is the degree of node i in the graph (nodes are ordered based on their degree) [257]. Thus, the algorithm uses at most $\Delta + 1$ colors (Δ being the maximum degree in G') and hence constructs at most $\Delta + 1$ zones. Unless the concerned graph exhibits certain structure (e.g., chordal graphs), it is hard to provide good worst case guarantees for coloring and hence clique cover problems. However, we show the effectiveness of our algorithm in practice, in Section 5.7.

All the nodes measure the link throughputs (every T_C seconds unless significant flow througphut changes are perceived) to their neighbors (the nodes with which they can communicate) and send this information to the CC. Given this set of individual flow measurements, the CC simply executes Algorithm 2 for determining the zones.

Algorithm 2 Zone Construction

Input: $G = (V, N)$

Output: $\vec{\mathcal{Z}}$ set of zones **Initialization:** $G' = Complement(G)$ $\mathcal{U} = V'$ uncolored vertices. $\mathcal{C} = \phi$ Colored Vertices $C = \phi$ set of colors

while $\mathcal{U} \neq \phi$ **do**
| $v = HighestValence(\mathcal{U})$ Color(v) $C \leftarrow$ UpdateColorSet(C,v) $\mathcal{U} \leftarrow \mathcal{U} - v$ $\mathcal{C} \leftarrow \mathcal{C} \cup v$

end

for $\forall c \in C$ **do**

$\mathcal{Z}_c = \phi$ **for** $\forall v \in \mathcal{C}$ $c = Color(v)$ **do**
| $\mathcal{Z}_c \leftarrow \mathcal{Z}_c \cup v$

end

end

Measurement of Flow Interactions: In a very naive case, if one were to con-

sider all combinations of flows to determine whether or not they yield a throughput gain when jointly activated, one would end up with an exponential number of possibilities. By only considering pairs of flows (as guided by our measurement study) we drastically reduce this requirement. In this case, if there are N nodes, $\binom{N}{4} \cdot 3 = O(N^4)$ measurements are needed with a brute force approach; this is because one can select four nodes to establish a pair of flows (3 distinct flow pairs for every 4 nodes). The approach would still incur significant overhead as these measurements might need to be performed every T_c, and N can be large.

In *BOLT*, interactions between flows are measured only across *pairs* of proactively formed zones. This results in a total of $\binom{Z}{2} \cdot \binom{\frac{2N}{Z}}{4} \cdot 3$, which in turn translates to $O(\frac{N^4}{Z^2})$ measurements; here, Z is the number of zones and nodes are assumed to be uniformly distributed across the zones for simplicity, i.e., $\frac{N}{Z}$ per zone). The reduction comes from restricting flow pair measurements to only *pairs* of zones (flow pairs, whose four end points are such that they belong to more than two, i.e., 3 or 4 zones, are not measured). Further, even for each pair of zones, not all flow pairs (involving both local and cross flows) in the two zones are measured. Only a fraction x of the net flow pairs, i.e. $\binom{\frac{2N}{Z}}{4} \cdot 3x$ in the two zones are picked randomly and measured; this is then used to construct the flow interaction models (explained in Section 3.5.3). The latter is in turn used to predict interactions between other flow pairs (spanning more than 2 zones) that were not measured. Our experiments reveal that the appropriate construction of zones (with larger size), allows x to be as low as 0.1 (10%), while still yielding good prediction accuracy.

3.5.3 Flow Contention Inference

At the end of the measurement phase, the CC can determine the LC and SC zones. Conceivably, links in the LC zones can be concurrently activated to increase spatial reuse. The challenge is to determine which links can be concurrently activated within and across SC zones to further increase spatial reuse. We discuss our approach for addressing this challenge in this section (our approach identifies spatial reuse opportunities across both LC and SC zones).

The quasi-stationary property of the PLC channel motivates the use of machine learning (ML) to learn and predict the impact of flow interactions. Our approach consists of a classification and, a prediction stage. The classification stage determines whether the aggregate throughput from concurrently running a pair of flows exceeds the average throughput over running the flows in isolation. The prediction stage estimates the aggregate throughput of a pair of flows activated concurrently, using training data. The two stages are based on classifier and prediction models used in ML; *BOLT* includes a feedback mechanism that recalibrates these models every T_c to adapt to channel variations.

Classification: Towards identifying flows that can be concurrently activated, we first identify a set of features that capture interactions between the transceiver pairs of the flows. Then, we intelligently use a set of classification models that take these features as inputs and determine if the flow pairs are amenable to concurrent activation.

What features to use?: Whether or not two transceiver pairs (flows) can simultaneously communicate depends on the interactions between them (carrier sensing, interference, noise etc. affect these interactions). We consider interactions (i.e., throughput) between all

possible combinations (i.e., every pair from the four nodes making up the two flows) as the basis feature set. Next, we reduce this basis feature set towards making the process faster and more accurate. This reduction is based on well-known ML techniques (dimensionality reduction). We consistently see that two of these features get eliminated in our experiments. Specifically, this reduction is a direct artifact of zoning; since the flow pairs we consider are between pairs of zones, the interactions between a transmitter and an unintended receiver can be derived based on the interactions between the two transmitters, the two receivers, and between the legitimate transmitter-receiver pairs. We do not elaborate on this here due to space constraints, but one can easily construct toy examples to see why this is the case.

The classification algorithm takes the reduced set of feature values and maps them onto two clusters (one for concurrent, one for isolated). Different classification algorithms have their pros and cons (based on the ways in which the clusters are formed). Although the PLC environment is relatively stable, there could be noise in the training set from electrical devices being turned ON/OFF. Thus, the classifier needs to be noise resistant. As a design goal, we seek to make sure that the training data for classification is not large (and yet achieve good accuracy). Finally, we want the approaches to be simple and efficient (fast).

Given the above objectives, we intelligently combine three classifiers to achieve very high accuracy. Our approach, shown in Fig. 3.12, first combines the outputs of two well known, simple classification techniques viz., Classification Trees (CT) and TreeBagger (TB).If the outputs of CT and TB differ (i.e., one suggests concurrent operations while the other contradicts), we choose a third approach namely, the nearest neighbor classifier (NN) to resolve the conflict. A description of these classification techniques can be found in [259].

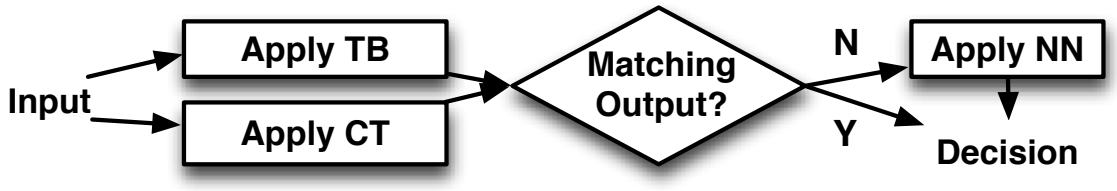

Figure 3.12: Classification Process Overview

The rationale for our approach is as follows. CT does not require a large training set but is sensitive to noise. The quasi-stationarity ensures low noise most of the time; however, plugging in devices causes noise which may cause CT to underperform. Thus, while CT is generally usable, it needs to be supplemented to deal with the noisy cases. TB uses innovative ordering to reduce impact of noise; however, the ordering inherently introduces a sampling bias (meaning some clusters may have a better chance of being picked). To reduce bias a larger training set may be needed. Thus, while it eliminates the issue with CT, it brings up a new problem. Thus we combine CT and TB and if they yield the same output decision, we can be relatively confident that the decision is the right one. If however, their outputs contradict, we go with a third classier for resolution. Here we choose a very popular generic classifier, viz., NN. NN offers lower accuracy than CT and TB (hence it was not chosen in the beginning) with small amounts of training data, but is not sensitive to noise and does not introduce sampling biases. We could have used multiple classifiers and used a majority vote at this stage, but prefer the simpler approach of just using NN since it offers high enough accuracy with low complexity.

Prediction of aggregate throughput : Finally, *BOLT* seeks to estimate the potential aggregate throughput for a given pair of flows that are activated concurrently (along with the individual contribution of the flows in the pair); this prediction is later

used when scheduling flows. We use boosting tree (BT) [117] as a regression approach for prediction. Our choice is motivated by the fact that BT is an ensemble method that fits complex non-linear variables (i.e., features) to predict outcomes. In addition to being fast, BT is resistant to missing data (low amount of training data) and eliminates outliers (due to noise). BT achieves a high prediction accuracy by adaptively combining multiple binary trees [117].

3.5.4 Flow Scheduling

At this stage, the CC knows which flows are reuse-friendly and the throughput gains from concurrent invocation of such flows. The scheduler then chooses the active flows at each epoch to maximize the aggregate throughput while meeting desired fairness constraints. Such scheduling problems are typically cast as utility maximization problems, as in

$$\text{Maximize} \sum_{f \in \mathcal{F}} U(\bar{\tau}_f) \tag{3.1}$$

where, $\bar{\tau}_f$ is the average throughput received by flow f in T_c, and $\mathcal{F}$ represents the set of flows. The choice of the utility function, $U()$, determines the fairness policy. We impose proportional fairness ($U(\bar{\tau}_f) = \beta_f \log(\bar{\tau}_f)$) [142], but *BOLT* can support other fairness policies governed by concave utility functions (e.g., max-min, min potential delay); β_f is the priority weight for a flow (e.g., latency-sensitive flows could have higher priority). The approach accounts for flow diversity and allocates resources to provide average throughputs ($\bar{\tau}_f$ over long-term, T_c) proportional to the flows' priorities and transmission rates. Solving the above problem ensures that the aggregate throughput of flows is maximized in a

proportionally fair manner over time scales of T_c.

Per-epoch Scheduling: It has been shown that finding the optimum solution to the above problem is equivalent to solving the following per-epoch (T_e) scheduling problem viz., maximizing the aggregate *marginal* utility in *every epoch*, i.e., Maximize $\sum_{f \in \mathcal{F}} \Delta U(\bar{\tau}_f)$ [142, 214]. ΔU_f is the marginal utility received by flow f in the epoch and is given by $\Delta U_f = \frac{dU_f}{dT_e} = \frac{dU_f}{d\bar{\tau}_f} \cdot \frac{d\bar{\tau}_f}{dT_e} = \frac{\beta_f \tau_f}{\bar{\tau}_f}$ for proportional fairness [214], where τ_f is the predicted throughput (transmission rate) for flow f in the current epoch and $\bar{\tau}_f$ is the *long term* average throughput received by the flow f so far. At the end of every epoch t, the average throughput received by a flow f is updated as,

$$\bar{\tau}_f(t) \leftarrow (1 - \frac{1}{T_e})\bar{\tau}_f(t-1) + (\frac{1}{T_e})\tau_f(t) \tag{3.2}$$

where $\tau_f(t) = 0$ when flow f is not scheduled. Updating the average throughput through an exponentially weighted moving average allows for fast adaptation to network dynamics. By tracking the throughput received by a flow thus far with $\bar{\tau}_f$, the scheduler weights the flow (as $\frac{1}{\bar{\tau}_f}$) accordingly in the next epoch to ensure fairness at time scales of T_c.

We now focus on the per-epoch scheduling problem, i.e.

$$\text{Maximize}_S \sum_{f \in S} \frac{\beta_f \tau_{f,S}}{\bar{\tau}_f} \tag{3.3}$$

Here, β_f and $\bar{\tau}_f$ essentially serve as constant weights for the flow in the current epoch. Hence, the optimization is w.r.t. to the flows chosen for schedule (S) in the current epoch; here, the throughput of a flow ($\tau_{f,S}$) chosen in the schedule depends on other flows in the set S, scheduled concurrently.

Algorithm: We want to select flows and/or flow pairs for each epoch; the selection

must ensure that the chosen flows do not negatively impact each other. Our scheduling problem can be cast as a maximum weight independent set (MWIS) problem as follows. We represent each individual flow (f), as well as each of the reuse-friendly flow pairs (f, g) as a separate vertex on a graph G, with the weight (w_i) of the vertex (i) being the weighted throughput achieved by the corresponding flow ($w_i = \frac{\beta_f \tau_f}{\bar{\tau}_f}$) or flow pair ($w_i = \frac{\beta_f \tau_{f,fg}}{\bar{\tau}_f} + \frac{\beta_g \tau_{g,fg}}{\bar{\tau}_g}$). Here, $\tau_{f,fg}$ ($\tau_{g,fg}$) is the throughput of flow f (g) when flows f and g operate concurrently. Any two vertices whose mutual flows are not disjoint have an edge between them in G. If two vertices represent a flow pair each, then each of the flows in one pair must be disjoint w.r.t. each of the flows in the other pair to avoid an edge in G. Now, the scheduler seeks to find the subset of flows and flow pairs that are disjoint, for which the aggregate weighted throughput is the maximum. This essentially reduces to finding an independent set of vertices (disjoint flows/flow-pairs) on G with maximum weight and automatically yields the optimum solution to our per-epoch scheduling problem.

However, finding even a maximum independent set or MIS (without weights or equivalently unit weights) on general graphs is NP-hard. Hence, we use a greedy algorithm (inspired by those for finding the MIS [258]) that yields efficient performance in practice. At each iteration, the vertex with the smallest degree is chosen and added to the independent set and its edges and neighboring vertices are removed. With vertices carrying weights in our case, we suitably adapt the algorithm to pick the vertex (i^*) that yields the smallest loss in weighted throughput in each iteration,

$$i^* = \arg\min_i \gamma_i, \text{ where } \gamma_i = \sum_{j \in N(i)} w_j - w_i \tag{3.4}$$

Note that, when we pick a vertex (flow or flow pair), we eliminate its neighbors from the

schedule and hence their weight contributions. Thus, at each iteration, the algorithm strives to pick the vertex that not only contributes maximum weight by its addition to the schedule but also minimum loss due to removal of its neighbors. In the end, we are left with a set of flows and flow pairs that are mutually disjoint and can be scheduled concurrently in the current epoch to maximize the aggregate weighted throughput. At the end of the epoch, the average throughputs ($\bar{\tau}_f$) of all the flows are updated based on Equation 3.2, which in turn affects their corresponding weights $\frac{\beta_f}{\bar{\tau}_f}$ and hence controls the relative flow priorities (track fairness) for scheduling in the next epoch.

Remarks: When the flow weights are the same, the problem and our algorithm reduce to that of a MIS and yield a worst case approximation guarantee of $\frac{\Delta+2}{3}$ [111].

Algorithm 3 Scheduling Algorithm (*Schedule* ($\vec{f}$))

Input: $\vec{f}$ flows to be scheduled and their weights

Output: $\mathscr{S}$ the schedule with the maximum weighted throughput

Initialization: Compute loss γ_i for each vertex i (flow or flow pair) based on its and neighbors' weights: $\mathscr{A}$.

while $\vec{f} \neq \phi$ **do**
  $\vec{f_c} \leftarrow$ PickSmallestLoss($\vec{f}$,$\mathscr{A}$) $\vec{f} \leftarrow \vec{f} - \vec{f_c}$ $\mathscr{S} \leftarrow \mathscr{S} \cup \vec{f_c}$

end

3.5.5 Handling Multi-hop Flows

So far, we discussed single-hop flows, whose end points can directly connect to each other. It is possible that the end-points of a desired flow may not be directly reachable from each other; thus, multiple hops are needed to establish connectivity. In other cases,

a multi-hop path may yield a substantially higher throughput than the single hop path. *BOLT* can accommodate such multi-hop scenarios easily with the following extensions.

Identifying multi-hop flows: When a single hop path is not possible for a flow, the multi-hop path that yields the highest effective throughput for the flow is chosen. It is easy to see that the effective throughput of a multi-hop flow f with H_f hops is $\tau_f = \frac{1}{\sum_{h=1}^{H_f} \frac{1}{\tau_{f,h}}}$, where $\tau_{f,h}$ is the flow throughput on hop h alone. On the other hand, when multi and single hop paths are possible for a flow, the multi-hop path yielding the best effective throughput is first selected and compared against the single hop throughput. Only if the selected multi-hop throughput exceeds the single hop throughput by a substantial amount (say 20-30 Mbps), the former is chosen. Otherwise, the single hop path is favored to avoid increasing contention in the PLC channel (more hop-flows for a single flow) for a marginal increase in throughput.

Scheduling multi-hop flows: Once the path for the flows are determined, the optimal approach is to address the end-end flow scheduling problem jointly over multiple hops as in wireless multi-hop networks. However, given the complexity of the associated solutions, we focus on a light-weight solution that leverages our single-hop scheduling solution from earlier and extends it to incorporate multi-hop flows in an efficient manner. We do this by decomposing a multi-hop flow with H_f hops into constituent H_f single hop flows. However, we weight each of the constituent single-hop flows' priorities as $\frac{\beta_f}{H_f}$ to spread the flow's utility across its constituent hops as well as to ensure that multi-hop flows are not un-duely favored over single-hop flows during contention. An obvious pitfall of this approximation is that since the various hops of a multi-hop flow are individually scheduled and optimized,

we might schedule more data on the upstream hops of a flow that do not eventually make it to the destination and hence throughput/utility of the multi-hop flow. To address this issue, we incorporate the notion of *data buffers* during scheduling through two considerations: (i) the transmitter of a hop (h) must not schedule more data than that available in its data buffer (i.e. received from its immediate upstream hop transmitter and yet to be scheduled, $B_{f,h}$), and (ii) the transmitter on a hop must not schedule more data than what its receiver can accomodate in its buffer meant for that flow ($B_{th} - B_{f,h+1}$). While the first consideration accounts for flow conservation, the second consideration employs a maximum buffer threshold (B_{th}) at each hop of a multi-hop flow to ensure that a hop does not push too much data without its immediate downstream hop ($h+1$) forwarding that data towards the destination (next hop).

Now, accounting for multi-hop flows, our per-epoch scheduling problem can be modified as,

$$\text{Maximize}_S \sum_{f \in S} \sum_{h=1}^{H_f} \frac{\beta_f \min\{\tau_{f,h}, B_{f,h}, B_{th} - B_{f,h+1}\}}{H_f \bar{\tau}_f} \tag{3.5}$$

$B_{f,h}$ is the buffered data at the transmitter on hop h for flow f that is available for schedule, while $B_{th} - B_{f,h+1}$ indicates the amount of data that can buffered at the receiver for this flow. Both these factors limit the amount of throughput that can be scheduled on that hop in the current epoch. Note that $B_{f,1} = \infty$ and $B_{f,H_f+1} = 0$ for a multi-hop flow and reduces to the single-hop flow scheduling problem when $H_f = 1$ for all flows. The scheduling algorithm itself changes only slightly. In addition to the priority weight updates, whenever the marginal utility of a flow on a hop is computed, its throughput is limited by the data buffer available at both the transmitter and receiver on that hop. The rest of the algorithm

remains the same. At the end of each epoch, the throughput of a multi-hop flow is measured to be that delivered on the last hop of that flow (τ_{f,H_f}), which is then used to update the average throughput of that flow ($\bar{\tau}_f$). The data buffers at the transmitter and receiver of every hop that got scheduled in the epoch are updated for thier respective multi-hop flows.

3.5.6 Discussion:

Our focus in this work is on link-level single-hop flows. The end-points of a desired flow may not be directly reachable from each other and multiple hops may be needed to establish connectivity. With *BOLT*, we maximize the link throughput considering each hop independently. Maximization of multi-hop path level throughput is approximated by multi link level single-hop optimization.

If all nodes are unable to reach a single PLC coordinator, the network may be sub-divided into multiple PLC sub-networks and a coordinator chosen for each. Then, *BOLT* will operate (at the coordinator) within each smaller PLC sub-network independently to allow for scalability. We believe that in large PLC networks, with such a sub-division, a large fraction of nodes in one PLC sub-network possibly can be made disjoint from those in the other sub-networks (different parts of an enterprise building). We expect the contention between nodes at the edge of these sub-networks to be small and handled by the 1901 MAC. A careful assessment of how to form such sub-networks is left to the future.

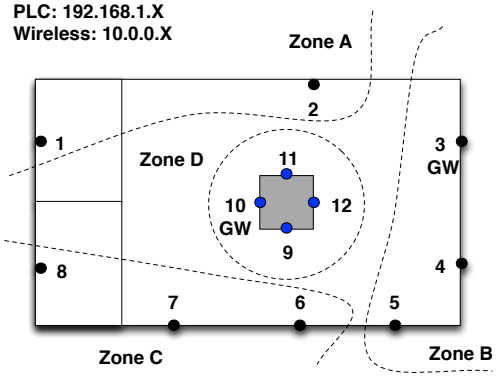

Figure 3.13: UNI Testbed.

Training Size	10%	20%	30%	40%	50%
BC	87(3)	89(2)	88(4)	93(3)	95(2)
TB	86(5)	90(2)	91(3)	94(3)	94(1)
NV	76(14)	74(11)	82(4)	79(7)	80(5)
NN	84(4)	87(2)	89(2)	93(3)	95(2)
SV	83(4)	85(2)	86(3)	87(3)	92(3)
CT	85(5)	87(4)	86(4)	93(3)	94(3)

Table 3.3: Classification accuracy (%): mean (std dev); ENT dataset

Training Size	10%	20%	30%	40%	50%
BC	82(2)	84(2)	85(3)	91(4)	93(1)
TB	80(4)	81(3)	85(4)	90(3)	92(1)
NV	71(14)	69(11)	79(4)	81(4)	79(3)
NN	80(3)	82(1)	83(2)	88(4)	92(3)
SV	79(4)	80(2)	81(3)	82(3)	89(3)
CT	81(3)	83(2)	86(4)	91(4)	92(3)

Table 3.4: Classification accuracy (%): mean (std devn); UNI dataset

Technique	UNI	Time	ENT	Time
LL	13	7	12	7
DT	14	6	11	6
BG	16	6	14	6
BT	11	5	10	4
NE	13	20	12	19
LS	17	6	15	6
RV	15	240	11	238
GR	15	10	12	10
VH	14	15	12	14

Table 3.5: Prediction error (%) and training time (msec)

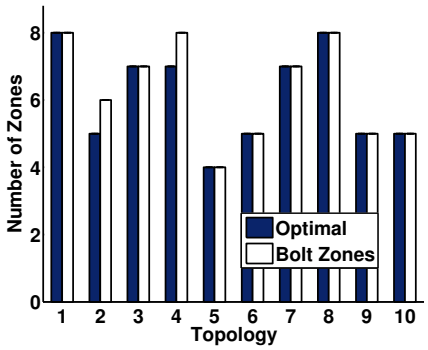

Figure 3.14: Sub-optimality of *BOLT*'s zone construction.

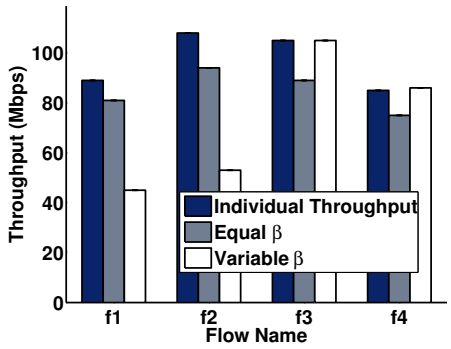

Figure 3.15: Evaluating fairness with *BOLT*.

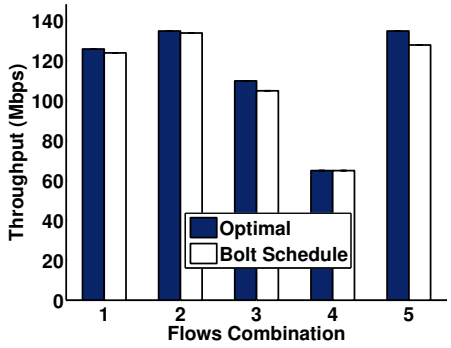

Figure 3.16: Sub-optimality of *BOLT*'s scheduler

3.6 Experimental Evaluations

We implement *BOLT* and deploy it on three testbeds (an enterprise, a university and a residence). We evaluate each of *BOLT*'s components as well as its holistic performance.

3.6.1 Implementation and setup

Testbeds: Our first testbed is in an enterprise and has eight machines (nodes) running Ubuntu 9.4 with two interfaces (a WiFi interface and a PLC interface) and spanning offices, labs, cafeterias and conference rooms. We experimented with different topologies but the one shown in Fig. 3.1 was the default topology. UDP flows with payload sizes of 1480 bytes, were established between node pairs for 30 seconds by default. The second testbed (shown in Fig. 3.13) is in a large lab setting at a university and consists of 16 machines running Ubuntu 14.4. The third testbed consists of 6 nodes, that are deployed at a residence; two of these run OS X and four run Windows 8. The data collection and scheduling phases of *BOLT* are implemented in C++. The classification and prediction models are implemented with Matlab. We implement the approach described in [166] to estimate the average flow throughput (in isolation) via a probing phase. Note here that we operate the experimental networks at relatively high loads since these are the regimes where the management of multiple flows becomes important.

Experimental Setup: Each network has an assigned central controller (CC) that performs network coordination and scheduling. The PLC channel itself is used as a control channel to collect measurements and disseminate schedules. The overhead for control

traffic is extremely small and does not interfere with ongoing communications; these packets essentially only contain the identifiers of the transceivers, file sizes in the case of requests, and time epochs assigned in the case of schedules and are 12 bytes long.

We consider one-hop PLC flows generated using Pareto and Uniform random variables; they are set-up between two randomly picked nodes in the test-bed. Later we consider a gateway model, where all PLC flows converge towards a common gateway(s) to access the Internet. *BOLT* schedules flows at the granularity of epochs of 100 ms. Unless otherwise specified, each experiment lasts for 5-10 minutes and results are averaged over twenty runs. We consider three levels of traffic load viz., high, medium and low that correspond to averages of (3,10, >10) flows per time epoch (100 msec). The average throughput per flow per epoch is our main metric of evaluation. To save space, we mainly present results from the UNI testbed (unless explicitly mentioned); experiments in other settings yielded similar behavioral results.

3.6.2 Benchmarking *BOLT*'s Components

Zone Construction: In this experiment, we compare the number of zones formed by *BOLT*'s zone construction algorithm with the *optimal* number of zones in the PLC topology. Ideally, node pairs between which the link quality is good must belong to the same zone. Determining a threshold for classifying a link as a "good quality" link is hard. We find via several experiments that if a link is able to achieve 75% of the maximum (isolated) link throughputs possible, it can be classified as a good quality link. Thus, in our UNI network, we assume a throughput threshold of $\alpha = 80$ Mbps for determining the zones (maximum link throughputs are ≈ 100 Mbps). Using a similar approach we find that nodes

in different loosely coupled (LC) zones can establish flows that can only sustain $< 1\%$ of the maximum throughput possible; correspondingly we set η to be 1 Mbps on the UNI network. The optimal number of zones is found via an exhaustive search across combinations of nodes (possible due to the moderate-scale topology) and picking the combination with the least number of zones. We consider ten different PLC topologies. As seen in Fig. 3.14, in 8 out of 10 of the topologies, *BOLT* constructs the optimal number of zones. Only in two of the topologies, *BOLT* employs one additional zone. Although our current testbed is relatively small, recall that zones are similar to contention domains in wireless; thus, small regions of interacting zones are of interest. Due to the inherently constrained nature of such interactions, we believe the efficiency of our zone construction will also scale to larger topologies.

Picking α: While a smaller value of α reduces the number of zones (more nodes per zone) and thus decreases the training measurements needed, it negatively affects the accuracy of predicting flow interactions. Our choice of α strikes a balance between accuracy and reducing training measurements.

Classification and Prediction: Since the accuracy of the classification and prediction models directly impact *BOLT*'s performance, we now compare our classifier/prediction approach against state-of-the-art techniques. First, we compare *BOLT*'s classifier (*BC*) against five popular classifiers (described in [259]) viz., TreeBagger (TB) , Naive Bayes Classifier (NV) , Nearest Neighbors (NN) , Support Vector Machines (SV)and Classification trees (CT). Table 3.3 depicts this comparison w.r.t different training set sizes for the ENT dataset (results from the UNI dataset were similar and are thus omitted due to space

constraints). By intelligently combining results of CT, TB and, NN in case of ties, *BC* achieves higher average accuracy than any of the five individual classifiers in more than 90% of the training scenarios (CDF over training scenarios not presented here). Further, with just 10-20% of the training data, *BC* is able to provide about 90% accuracy, which is highly promising. The reasons for BC's superior performance are as follows. First, by combining the results of TB, CT and NN, *BC* outperforms each of these individual classifiers. For SV to be efficient, its parameters need to be calibrated for each specific topology; using SV without such customized calibration can cause high false positives and negatives. In NV, the posterior probability estimate is negatively affected if there is only a small occurrence of a certain class (e.g., an isolated node in a zone).

To evaluate the accuracy of our prediction model using Boosting trees (BT), we compare it with eight other different prediction models (described in [117]): Least squares Linear regression (LL) , Decision trees (DT) , Bagging trees (BG) , Neural networks (NE) , Least Squares Support Vector Machines (LS), Relevance Vector machine (RV) Gaussian Process Regression (GR) and, Variational Heteroscedastic Gaussian Process Regression (VH). In Table 3.5, we show the root mean square error (RMSE) rate (%) and training time (ms) for each technique for the two datasets. The RMSE captures the accuracy of prediction, while the training time captures the computational intensity (i.e., the time required to construct the prediction model). We see that in addition to its high accuracy, BT's running time is the fastest.

Scheduling: To evaluate the efficiency of our scheduler, we create two sets of six flows each. *First*, we set the priority coefficient of both flow sets to one ($\beta_f = 1$, *Scenario*

1). *Second*, we increase the coefficient for the latter set to two, while keeping that of the first set to one (*Scenario 2*). We plot the throughput of two of the flows from each set in Fig. 3.15 for the two scenarios. In *Scenario 1*, *BOLT*'s scheduler loses about 5-10 Mbps in throughput for the individual flows compared to their throughputs in isolation (total network throughput is higher). This shows that the throughput loss due to sharing of the PLC medium is minimal, indicating that the scheduler picks the right set of flows to operate concurrently (so as to not degrade their individual throughputs appreciably). Further, the flows receive their average throughput proportional to their individual epoch throughputs (proportional fairness is ensured). In *Scenario 2*, the higher priority flows have a two-fold increase in their relative throughputs, at the expense of the flows in the first set. This means that in a given duration, the first two flows are able to transfer much more data than the second two flows (which have lower priority and thus get scheduled less often). This indicates *BOLT*'s ability to differentiate between flows of different types, thereby allowing it to prioritize and provide lower latency for real-time flows.

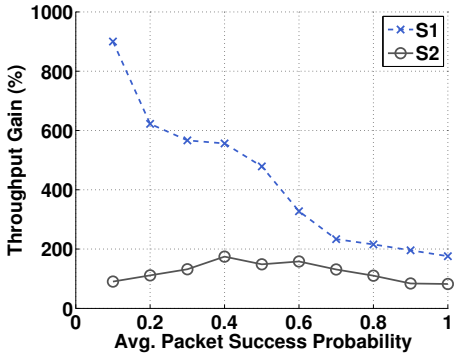

Figure 3.17: Throughput gain over baselines.

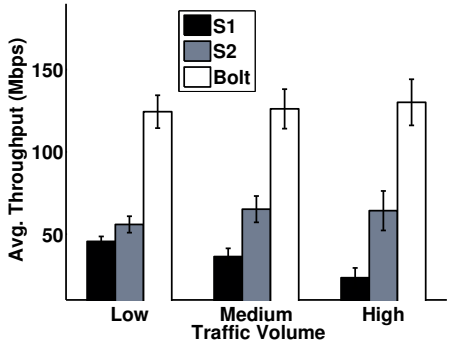

Figure 3.18: Impact of traffic load.

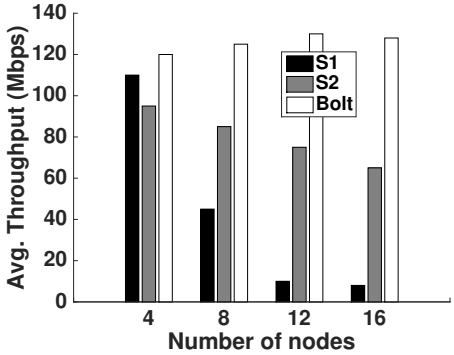

Figure 3.19: Impact of topology size.

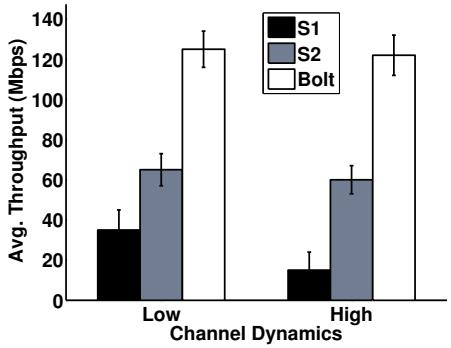

Figure 3.20: Effect of channel dynamics.

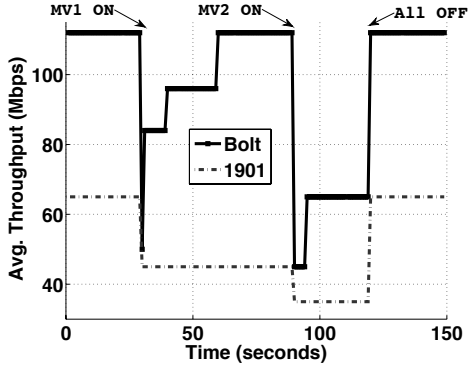

Figure 3.21: resiliency to channel dynamics.

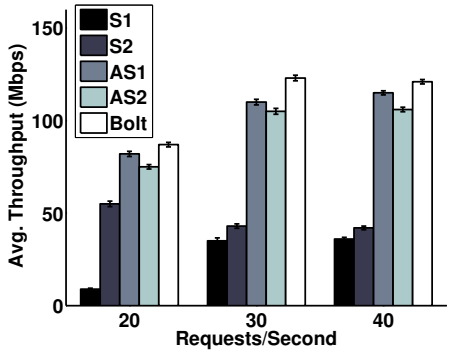

Figure 3.22: Holistic evaluation

Scheduler sub-optimality: We now quantify the sub-optimality of *BOLT*'s per-epoch schedule computation (recall that finding the optimal schedule corresponds to solving the NP-hard MWIS problem). We consider a topology with a subset of six nodes and establish four flows (arbitrarily chosen) between these nodes. We choose five such combinations, C_k, $k \in \{1, 5\}$. We also exhaustively generate all possible schedules for each combination, and pick the optimum i.e., the one that yields the highest utility (weighted throughput). We compare this utility against that achieved with *BOLT*'s schedule (priority weights are

set to 1). Fig. 3.16 compares the utilities for the five combinations in the two cases. We observe that *BOLT*'s (much simpler) scheduler yields utilities that are at most 5% lower than the optimal.

3.6.3 Holistic evaluation of *BOLT*

Impact of Scheduling: We consider two baselines: one where all given application layer flows are concurrent (state-of-the-art PLC operation, **S1**), and another, where such flows are queued and only one flow is active in any given epoch (**S2**). Since no classification/prediction models are required for either of these baselines, they can be considered as alternate potential systems for PLC. In both cases, the MAC protocol based on 1901 in the adapters arbitrates the channel. In the first experiment, we vary the average packet delivery ratio (PDR) and measure the throughput gain of *BOLT* over the two baselines. PDR is a function of the channel dynamics (eg. electrical apparatuses plugged on/off) that needs to be controlled to allow for comparison; hence, we control PDR by manually dropping packets with a certain probability at the receiver. We keep the average traffic load to be three flows per epoch. As shown in Fig 3.17, we observe that the throughput gains with *BOLT* as compared to S1, can be as high as 8.5x. The gain is higher in bad channel conditions, where running all incoming flows together (in S1) leads to aggravated collisions/back-offs. By scheduling the right sets of flows together, *BOLT* alleviates this effect and achieves much higher throughputs. With S2, running individual flows sequentially avoids the collision/back-off penalty; however, it also misses out on reuse opportunities that lead to higher total throughput. The throughput gains with *BOLT* over S2 are also significant (100-200%) and remain almost constant across different channel conditions. Here,

BOLT's gains are mainly from its ability to leverage spatial reuse.

Impact of Traffic Load: Next, we study the impact of traffic load in Fig. 3.18. The average throughput of S1 decreases as traffic load increases; more flows increase contention (backoffs) and collisions from hidden terminals. The average throughput of S2 remains almost constant over different traffic loads since it schedules a single flow in every epoch and hence does not depend on the number of flows (but only their individual throughputs). Notably, *BOLT*'s throughput is at least 2.5x that of S1 and 1.5x that of S2.

Impact of variations in the topology size: Next, we examine how the performance of *BOLT* changes as the topology size is varied; the topologies we consider contain SC zones. For each topology (of a given size), we divide the nodes into two equal size sets; the first are the sources and the second set, the destinations. Thus, the load increases linearly with the topology size. For each topological size, we choose 20 randomly chosen configurations (as described above), and run experiments for 30 seconds each. From Fig. 3.19, we see that *BOLT* maintains a steady throughput as the topology scales. There is in fact a slight improvement in throughput due to better exploitation of spatial reuse from the increased number of flows. S1 suffers from collisions/backoffs as the topology (and thus, the corresponding load) is scaled. The throughputs with S2 remain fairly stable; however, because of a slightly higher number of poorer quality links, the throughput takes a slight dip as the topology size (and thus, the number of flows) is increased.

Impact of Channel Dynamics: Our next experiments (Fig. 3.20) are done during different times of day and capture the impact of channel dynamics on *BOLT*'s performance. The rate of switching (on/off) of electric apparatuses (which induces the

dynamics) during peak times (arrival of people to work, lunch time, etc.) is high and at other times (early morning, at night, etc.) is low. The average throughput of *BOLT* does not vary much in either case; it schedules different subsets of flows across different epochs and thus copes with the dynamics. Similarly, S2 has a relatively steady performance as it picks different flows across epochs.

To further illustrate *BOLT*'s ability to cope with dynamics, we conduct a controlled experiment on our residential testbed. Here, we turn on two microwave ovens in an operational PLC network at different times. At each instance, as shown in Fig. 3.21, the throughput takes a hit. However, *BOLT* quickly isolates the effect of the poor quality links and the network is reconfigured to restore the throughput to almost the same levels prior to the oven(s) being turned on. On the other hand, without *BOLT*, the poor quality links further degrade the throughputs that were achieved with just 1901 (which were lower than with *BOLT* to begin with); the more the noise the bigger the hit with 1901. This experiment exemplifies *BOLT*'s ability to quickly cope with channel dynamics to restore throughput. Fairness is compromised a little in the short term (results not shown due to space constraints), but is restored long term as the microwaves go off.

Impact of Classification/Prediction Models: Finally, to study the impact of the models' choices on *BOLT*'s performance, we consider two alternate systems, where we retain the scheduler in *BOLT*, but change its classification/prediction models: *(a) Alternative System 1 (AS1):* The classification model is Nearest Neighbor (NN) and the prediction model is Gaussian Process Regression (GR). *(b) Alternative System 2 (AS2):* The classification model is Naive based (NV) and the prediction model is Neural networks (NE). We

also consider a different scenario where all nodes communicate with two common gateway nodes connected to the Internet. We stream multiple HTTP pages (CNN, Facebook, Twitter, Google, Yahoo) through these two gateway nodes. The average size of the HTTP web pages are 4.4, 1.3, 2.7, 5.4 and 2.4 MB, respectively for these sites. We generate requests for web pages according a Pareto random variable, and assign the requests to each node randomly. The results are averaged over 30 runs, each lasting for 1 minute. In Fig. 3.22, we compare the average throughput of *BOLT* with that of AS1 and AS2. We see that the bulk of *BOLT*'s gain comes from its scheduling component. However, its prediction/classification models still offer gains over AS1 and AS2 (15-30%); this shows that the selection of the right classification/learning models is useful. Specifically, the higher accuracy from these models contribute to better scheduling decisions and hence, system throughput. S1 and S2 underperform *BOLT* as well as AS1 and AS2 because of their sub-optimal scheduling decisions (as observed in the prior experiments).

3.7 Conclusions

In this paper, we design and implement a standards agnostic framework *BOLT*, to realize the throughput potential of PLC, for it to serve as a viable backhaul for local network connectivity. *BOLT* aggressively reuses the spectrum while avoiding collisions and backoffs to drive the network throughput to near-optimal levels, while enforcing desired fairness requirements. It is lightweight and uses only a small set of online training measurements. Real-world experiments showcase *BOLT*'s ability to improve system performance significantly over state-of-the-art PLC solutions.

Chapter 4

Exploiting Subcarrier Agility to Alleviate Active Jamming Attacks in Wireless Networks

4.1 Introduction

Wireless communications can be easily disrupted by malicious injection of interference, aka jamming. Given the commercial availability of jamming devices today [15, 12, 23], mounting Denial-of-Service (DoS) attacks using jamming is an easy task.

How easy is it to combat jamming? Previous efforts have tried to mitigate jamming by tuning several physical layer knobs. Examples include adaptive power and rate control, or the use of lower modulation rates in order to reduce the packet error rates (PER) [187, 194, 189] in the presence of jamming interference. Frequency hopping has also

been considered in cases where there is significant additional available bandwidth for use [183, 197]. All of these prior studies conclude that in general, it is very difficult to overcome the impact of active jamming, especially when jammers account for the inherent properties of MAC layer protocols [48]. Our extensive testbed measurements using legacy WiFi devices as well as programmable wireless boards [17] support such an argument.

Our measurements however, also reveal a new, promising dimension for malicious interference avoidance in OFDM (Orthogonal Frequency Division Multiplexing) settings [215]. Specifically, we identify a feature that can be exploited with OFDM to mitigate jamming; more importantly, this can be applied in conjunction with most previously proposed anti-jamming schemes.

Exploiting an intrinsic aspect of OFDM signal propagation: OFDM is currently a widely adopted transmission scheme in many different wireless network technologies (e.g., LTE [223], WiMAX [212] and 802.11 [9]). In traditional OFDM implementations, the transmission power is uniformly distributed across a predefined set of frequency subcarriers; the number and width of these subcarriers dictates the available channel bandwidth [215]. Due to physical obstructions and interference, signal power (even that of a jammer) undergoes different levels of fading across the different subcarriers. As a result, on some of the subcarriers the received jamming signal strength can be high, while on other subcarriers it is likely to be low[1] [272].

Employing subcarrier-level radio agility: Our testbed measurements also indicate that jamming signals are likely to experience varying levels of fading on different

[1]As discussed in section 4.3, we focus on the case of OFDM jammers due to the difficulty in detecting their presence [194].

OFDM subcarriers. As a result, some subcarriers may not be "significantly affected" by the malicious power emission; such "cleaner" portions of the available spectrum could be temporarily used for legitimate packet transmissions, as long as a transceiver pair is made aware of which those subcarriers are.

Thus, we design and implement a framework that allows a transceiver pair to exchange information that reveals the "clean" subcarriers in the available spectrum, where the jamming signal experiences significant fading. Once such sub-carriers are identified, we pool power onto them (to the extent allowed), and utilize them for packet transmissions to increase the probability of successful packet delivery and thereby the long-term throughput (while being actively jammed). Note here that by active jamming we refer to cases where there is prevalent jamming interference when communications are ongoing; this includes both constant and reactive jamming. More specifically, our contributions in this paper are the following:

1) Experimental characterization of jamming interference: We perform a large set of testbed measurements using WARP reference boards in order to observe the impact of fading on jamming transmissions, across different OFDM subcarriers in a spectral band. We experiment on different network topologies and with various jamming patterns. We validate our hypothesis that there may be portions of the spectrum where the jamming signal experiences deep fading.

2) Design of our subcarrier-level radio-agile anti-jamming framework: We design a framework that enables a pair of legitimate transceiver pair, say Alice and Bob, to exchange information regarding which subcarriers should be used for packet transmissions

in each link direction (note that the fading patterns for the jamming signal will differ at Alice and at Bob). For this, we leverage *raptor* codes [227] to securely and efficiently exchange information on the "clean" sub-carriers. Subsequently, we design an algorithm (executed at each transceiver) that considers the subcarrier-specific SINR and the expected number of packet retransmissions, in order to make subcarrier-level transmission decisions, such that the long-term user throughput is maximized. These components constitute our jamming interference mitigation scheme, JIMS.

3) Implementation and evaluation of our framework: We implement JIMS on the WARP platform [17] and evaluate its efficiency via extensive experiments in the 2.4 GHz ISM band. We involve legacy WiFI nodes in many of our experiments. We also implement our framework on the NS3 simulator in order to observe its efficacy in large-scale wireless network settings. Both our experiments and simulations incorporate diverse jamming patterns that reflect a large number of settings. We observe that the application of JIMS can achieve a user throughput restoration of up to 75% and network-wide throughput improvements that range between 30% and 75%.

The rest of the paper is structured as follows. In section 4.2, we discuss relevant background and previous work. In section 4.3, we discuss our WARP measurement based observations on the effects of fading on jamming signals, and subsequently we derive a set of conclusions that drive the design of our framework. In section 4.4, we present the details of our proposed design. In section 4.5 we evaluate our framework via extensive testbed measurements and simulations. Finally, our conclusions form Section 5.9.

4.2 Background and Related Work

In this section, we first provide the relevant background on jamming attacks. Subsequently we discuss previous related studies on anti-jamming and differentiate our work.

Malicious interference injection: As discussed in section 4.1, various types of jamming devices are readily available in the market today [15, 12, 23]. Although initial models were very simple in their operation (i.e., they were simply emitting energy all the time), newer devices have incorporated intelligent power emission patterns, in order to conserve battery power and avoid detection. More specifically, jammers can emit power continuously or intermittently. Intermittent jammers are further categorized based on the duration of the active and inactive time intervals; for example, periodic jammers use fixed durations for these intervals. Moreover, *reactive* jammers act more intelligently, by emitting power only if they overhear traffic; this makes them more energy-efficient and more difficult to detect [67].

Previous related studies: Most of the previous efforts on alleviating or avoiding malicious interference employ frequency hopping, or power and/or bit rate adaptation techniques. With frequency hopping, legitimate users decide on a hopping pattern across the set of available channels in an effort to avoid the jammer [183, 126, 197]. However, frequency hopping techniques cannot avoid jammers that can distribute their power across multiple bands simultaneously [234]. In [194] the authors use power control and bit rate adaptation towards mitigating jamming interference; however, the proposed approaches can exacerbate interference due to increased power levels or inappropriately set carrier-

sense thresholds. The proposed bit rate adaptation is only useful if the jammer is silent intermittently. Similar techniques are proposed in [187] and [189]. A survey on jamming attacks and mitigation solutions can be found in [196]. An overview of the analysis of robustness and weakness of OFDM-based systems is presented in [225]. The authors examine jamming attacks on OFDM in addition to proposing possible counter measures. Our approach, unlike the prior approaches leverages OFDM diversity; most importantly, *it can be used in conjunction with many of such techniques.*

Unintentional interference was modeled in [193] while an anti-jamming technique for tactical communications was studied in [271]. In [135], the authors proposed a generic anti-jamming system for MIMO based OFDM systems. Different jamming techniques were investigated theoretically through analysis in [169] as well as in [69].

Yao et al. in [165] propose a DSSS (Direct Sequence Spread Spectrum) anti-jamming method for broadcast transmissions. The method relies on spreading codes to encode a bit stream of data. However, due to its nature, this work is not applicable in wireless communication systems that are based on OFDM.

Note here that this work is based on our preliminary work in[45].

4.3 Sub-carrier Radio Agility aids Anti-jamming

In this section, we describe our testbed experiments on assessing the behavior of malicious interference from the perspective of OFDM sub-carrier level propagation. Our measurements offer insights on how the jamming power is distributed across the subcarriers of the available spectrum. These insights motivate and form the foundation of our radio-

agile anti-jamming framework design, which we discuss in section 4.4.

In a nutshell, our measurement-based, key findings are the following:

- The Received Jamming Signal to Noise Ratio (or **RJSNR**) experienced by legitimate users (transceivers), can often be quite low on some OFDM subcarriers.
- Due to the asymmetry in the perceived RJSNR per subcarrier, a transceiver pair needs to exchange information regarding the subcarriers with respect to which the RJSNR is low, at each end (of the link).
- Due to variations in RJSNR over time, nodes need to periodically send updated channel feedback. A low-overhead feedback frequency of the order of once every 1000 msec suffices in relatively static settings.

In what follows, we describe our threat model and experimental configuration; subsequently we present our observations.

The threat model: We consider a jammer (Eve) that transmits OFDM signals with the same transmission power budget as legitimate users, thereby imitating a typical legitimate device to avoid detection. Other than this, we do not require any other constraint on the jammer.

The power budget of jammer is the same as the to power budget of the transmitter. The jamming power is distributed on all the subcarriers uniformly; we also examine cases where the jammer can distribute this power arbitrarily i.e, in any fashion it seeks.

We also assume that a pair of legitimate transceivers use a shared symmetric key to encrypt the channel state information that they need to exchange. The key may be either preloaded, or derived via an authentication and key agreement protocol (e.g. [61, 62]).

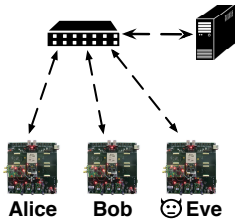

Figure 4.1: Alice, Bob and Eve are all placed on a straight line.

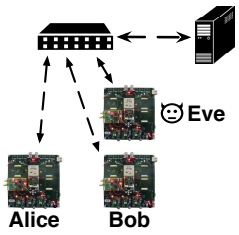

Figure 4.2: Eve is placed at 90 degrees to Alice and Bob.

We also do not address the problem of jammer detection; we restrict ourselves to the mitigation of active jamming. We assume that schemes such as those proposed in [264] can be used to distinguish between benign and malicious interference.

Experimental setup: Although our study is generally applicable with any wireless OFDM system, throughout the rest of the paper we particularly focus on measurements in the ISM 2.4 GHz band. We consider a 20 MHz channel in the ISM band (channel 6, centered at 2.437 GHz); the channel consists of 64 subcarriers, 48 of which are used for data transmissions. We perform our experiments late at night in a campus building, and we verify that this channel is not used by any collocated WLAN networks. Our experimental assessment on the effects of fading on jamming signals involves a pair of legitimate devices (Alice and Bob), and a custom-made jammer (Eve). Alice, Bob and Eve are all stationary

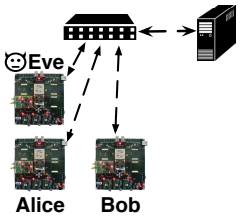

Figure 4.3: Eve is placed at 90 degrees to Alice and Bob.

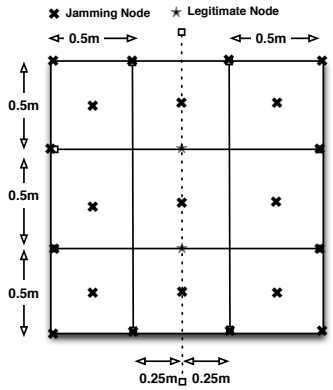

Figure 4.4: Alice, Bob and Eve Position Grid.

nodes that use fixed power budgets. Note that although all nodes operate in the ISM band, for this set of experiments they do not follow the IEEE 802.11 CSMA-CA MAC protocol; instead, Alice transmits packets to Bob as soon as they arrive at her output queue. All three devices are based on WARP programmable boards [17], which are connected to a management server (Fig. 4.1-4.3).

Each reference board is equipped with a Xilinx Virtex-II Pro FPGA and 4 daughter-boards operating in the ISM band. The OFDM implementation that we use (WARPLab v6) supports BPSK, QPSK and 16 QAM modulation rates, and a 40 MHz sampling rate. Legitimate packets carrying CSI information have a length of 240 bytes, while data packets have a length of 1500 bytes; each experiment lasts for 5 minutes and is repeated 20 times.

Nodes are placed in a two dimensional grid of dimension 1.5m X 1.5m. The

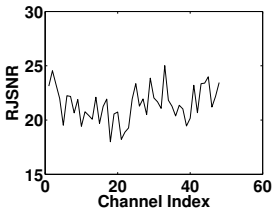

Figure 4.5: Alice's perceived RJSNR.

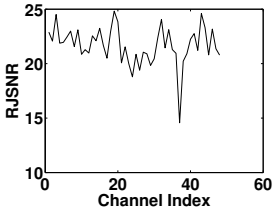

Figure 4.6: Bob's perceived RJSNR.

transmitter and receiver positions are relatively fixed (50 cm apart) as shown in Fig. 4.4.The jammer is placed at 12 different locations on the border of the grid and at 9 positions inside the grid, also as shown in Fig. 4.4.

Experimental insights: Next, we elaborate on specific network configurations and discuss our observations.

i. Jamming signals often experience deep fading on some OFDM subcarriers: We configure the jammer to constantly emit electromagnetic energy on channel 6 (2.437 GHz), and we capture the observed RJSNR at legitimate nodes[2]. A sample of our measurements is depicted in Fig. 4.5 and Fig. 4.6, which shows the RJSNR as perceived by Alice and Bob on each of the 48 data subcarriers. We observe that on quite a few of the subcarriers, the RJSNR can be quite low.

This promising observation serves as the main motivation in designing JIMS: *If legitimate transmitters could somehow estimate the subcarrier-level RJSNR values at the*

[2]As we discussed earlier, channel 6 was not used by other wireless networks in the neighborhood.

receivers, they could simply use only those subcarriers where the RJSNR is low, for packet transmissions.

ii. The jammer's fading profile differs at each receiver: This is evident from our RJSNR measurements depicted in Fig. 4.5 and Fig. 4.6: the perceived RJSNR per subcarrier differs at each legitimate node. This observation is in line with previous studies which have also observed that different receivers observe different signal qualities from the same transmitter, due to differences in noise and fading (which in turn is because of differences in their positions relative to the reference transmitter) [223]. Our measurements suggest that *in order for Alice to utilize low-RJSNR subcarriers when transmitting packets to Bob in the presence of Eve, Bob should send reliable channel state information (CSI) feedback to Alice to indicate the subcarriers relatively unaffected by Eve.*

iii. The RJSNR value changes over time on each subcarrier: We perform experiments to observe the variation of RJSNR over time. Similarly as above, in this set of experiments Eve continuously emits electromagnetic energy, while Alice and Bob measure the corresponding RJSNR for her signal. As intuitively expected, due to fading, scattering and power decay, Alice and Bob observe different RJSNR values over time on each subcarrier of the available spectrum. While we have performed measurements with approximately 120 different intervals for sending CSI feedback, in Figures 4.7 and 4.8 we plot how the RJSNR values for three specific subcarriers, for two different feedback intervals, i.e., for 2300-msec and 1000-msec, respectively.

We observe that if the RJSNR information is fed back once per 1000 msec, the

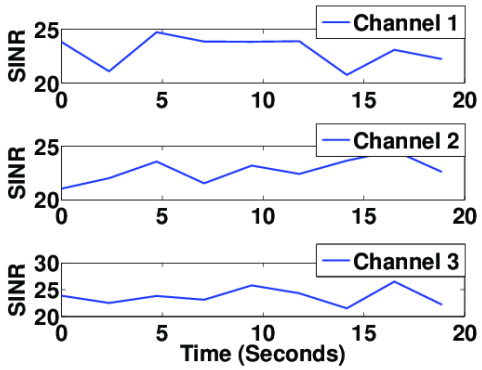

Figure 4.7: Per subcarrier RJSNR with feedback every 2.3 sec.

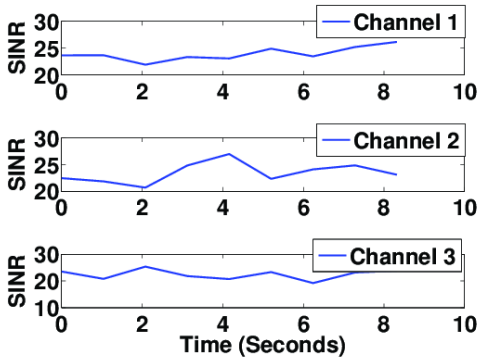

Figure 4.8: Per subcarrier RJSNR with feedback every 1 sec.

intermediate RJSNR variations are captured much more accurately than with the 2300-sec interval. Clearly, the smaller the feedback interval, the higher the probability that a significant variation in RJSNR is captured. In other words, more frequent CSI feedback increases the accuracy in Alice's determination of the subcarriers where the jammer's signal strength is low. On the other hand, as the frequency of feedback messages increases, so does the network overhead. Note that in the presence of malicious interference, it is very important to utilize the scarce available bandwidth even more wisely, compared to benign conditions. We carefully examine our measurements with different feedback intervals (where

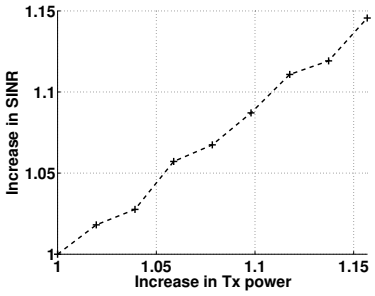

Figure 4.9: The effect of increase in transmission power on the SINR in the presence of a jammer.

each feedback message contains a vector of pointers to subcarriers that should be used by the transmitter, as we discuss in the following section). We conclude that *while the jammer is active, a Channel State Information (CSI) feedback periodicity that limits the network overhead to acceptable extents, while providing an accurate view of the channel is on the order of once every 1000 msec.* We use the value of 1000 msec in our evaluations. In section 4.5 we provide more details on the assessment of network overhead using more frequent feedback intervals.

iv. Effect of power allocation on the SINR value: In an interference dominated setting (as in the presence of a jammer), the SINR which is $\frac{P_R}{N+I}$ can be approximated to be $\frac{P_R}{I}$; in these expressions, P_R is the received power, I the interference power and N the ambient noise power. P_R is proportional to the transmit power P_T and the channel attenuation, say ζ. In static settings (slow fading), ζ can be expected to be fairly stable over time. If the interference from the jamming signal I is also stable over time (jammer does not move much[3], which is the scenario we consider here), one can expect the SINR to change in proportion to P_T. In other words, if we change P_T to, say, αP_T, the SINR

[3]Typically jammers are strategically placed in areas where they can disrupt ongoing communications for a prolonged period of time; this suggests that they do not move frequently.

can also be expected to scale by the factor α. We conducted testbed measurements in order to validate this hypothesis. Specifically, Fig. 4.9 depicts the increase in the SINR, relative to an increase in P_T, in the presence of the jammer. The results demonstrate that our hypothesis holds. We use this observation to determine the gains in SINR with power reallocation with JIMS.

4.4 Our Subcarrier-Level Radio-Agile Design

In this section, we describe the design of our jamming interference mitigation scheme (JIMS), which is based on the key observations made in Section 4.3. The scheme consists of three major steps. **First**, the legitimate pair of transceivers independently determine the OFDM subcarriers that are relatively unaffected by the jamming signal. **Second,** by means of using Raptor codes [227], they exchange the information they have determined (CSI) in the first step. **Third**, each transceiver uses this information, to transmit symbols on only an appropriately chosen set of subcarriers (that are relatively unaffected at the receiver).

To maximize the likelihood of correct reception, and facilitate higher transmission rates on the relatively unaffected subcarriers, we further consider an extended version of JIMS, which involves pooling power from the subcarriers that remain unused (to the extent allowed by regulations) to those subcarriers on which, symbols are actively transmitted. We call this extended version of JIMS as JIMS-PA (for Power Allocation).

4.4.1 Determining the subcarriers affected by the jamming signal

We consider two ways for detecting the subcarriers that are affected by the jammer. For ease of discussion, let us assume that Alice is executing this step. She simply measures the signal from the jammer when there are no other transmissions in the vicinity. Towards this, we first assume that somehow Alice knows that a jammer is in operation using one of the techniques proposed in [264]. Next, we assume that Alice can simply listen and detect the jamming signal. If the jammer emits energy continuously or without regards to whether or not Alice and Bob are transmitting, this can be done easily. If the jammer is reactive i.e., only transmits upon sensing a transmission from Alice, Alice can send a short pilot to trigger the jammer and subsequently go silent (assuming half-duplex mode of operations as is common with legacy systems); the jamming signal that spills beyond Alice's prompt can then be captured to determine the jammer's profile. The signal can be then decomposed to determine the SNR on each of the subcarriers in the operational band. In other words, the subcarrier level RJSNR can be determined. We call this approach the *explicit approach* of determining the affected subcarriers.

Second, let us assume again that using an appropriate technique from those reported in [264], the presence of the jammer is detected. Bob then sends a pilot signal to Alice. Alice then determines the SINR on each of the subcarriers in that pilot signal. In a nutshell, if either the signal quality is low and/or the jamming signal is high, on a specific subcarrier, that subcarrier is deemed unfit for communication. Other subcarriers where neither of the above scenarios hold true, are appropriate for transmission. We call this approach the *implicit approach* of determining the affected subcarriers.

4.4.2 Subcarrier selection

Using either the explicit or implicit approach, Alice is able to determine the quality of communications on each of her subcarriers. Now, she has to determine the appropriate set of subcarriers for use by Bob, for him to communicate with her. The process of selecting this set is different with the explicit and implicit approaches described above.

With the explicit approach, the good subcarriers (to be used for communication by Bob) are chosen based on simple RJSNR threshold. Specifically, if the RJSNR is lower than a certain threshold on a subcarrier it is deemed a *good* subcarrier.

A simple way to choose the RJSNR threshold is to determine average RJSNR from that observed on all subcarriers, and use those that have RJSNRs lower than the average. Specifically, the threshold η is computed to be:

$$\eta = \frac{1}{n} \sum_{i=1}^{n} RJSNR_{c_i} \tag{4.1}$$

where, $RJSNR_{c_i}$ is the measured RJSNR on subcarrier, i and n is the total number of data subcarriers. Upon computing η, Alice classifies those subcarriers, c_i with $RJSNR_{c_i}$ greater than η to be unsuitable for reception. Later, in Section 4.5, we examine the performance of JIMS with other possible thresholds as well.

With our implicit approach, Alice measures the SINR on each of the subcarriers on a pilot transmitted by Bob. Thus, with this approach, she first computes the average SINR, ξ, by considering all the subcarriers as follows.

$$\xi = \frac{1}{n} \sum_{i=1}^{n} SINR_{c_i} \tag{4.2}$$

where, $SINR_{c_i}$ is the SINR with respect to subcarrier, i and n is the total number of data

subcarriers. Alice then only chooses those subcarriers, c_i with $RINR_{c_i}$ higher than ξ as the good subcarriers (for Bob to communicate with her).

Choice of the right threshold: One of the challenges that arises with both the explicit and the implicit schemes is "How do we choose the right threshold (be it RJSNR or SINR depending on whether the explicit or implicit approach is used)?" For simplicity, let us just consider the implicit approach; instead of choosing ξ as above, let us assume that we choose a different static threshold ξ'. If we are liberal, and chooose ξ' to be low, we include a large set of subcarriers; however, the SINRs on some of these subcarriers will be unacceptably low. If instead, we are conservative and choose a high value for ξ', we may end up excluding a large number of subcarriers (on a few of which, communications may in fact be possible), and thus, end up achieving a lower throughput than what is possible. We find via experiments that choosing the average value (as discussed above) to be the threshold, provides a good compromise between the two extreme cases, in most scenarios. We evaluate this choice, by comparing the performance with other cases where a static threshold is chosen, in Section 4.5.

4.4.3 Exchanging CSI

At this point, both Alice and Bob have determined the set of subcarriers on which, they expect to be able to receive symbols from each other, in the presence of the active jammer. Unfortunately, the subcarriers on which Alice can receive information (known only to Alice at this stage) may be different from those on which Bob can receive information (known only to Bob at this stage). Thus, we need a way for Alice to let Bob know "which subcarriers to use" for communicating with her (Bob needs to do likewise).

A low throughput channel using Raptor codes to exchange CSI: Towards, this we leverage Raptor codes to communicate this information (which as previously mentioned, is called the CSI). Raptor codes belong to the class of fountain codes with linear encoding and decoding times. Fountain codes are rateless fault-tolerant codes that can enable reliable communications on erasure channels; examples of fountain codes include Raptor codes [227] and LT-codes [167]. Encoded symbols are generated by the encoder on-the-fly. The decoder recovers the source block by collecting a sufficiently large set of encoding symbols. Hence, Raptor codes facilitate communications in the presence of the jammer (jammed symbols could be considered to be erasures), by utilizing a very low throughput channel (as shown by our experiments later in this paper). Thus, in JIMS we only utilize these for the exchange of CSI information, and later simply utilize the relatively unaffected carriers without applying Raptor codes.

Specifically, Alice uses a bit vector to indicate the subcarriers to be used by Bob, and encodes this using Raptor codes. She transmits the encoded bit vector repeatedly (each time, the vector is encoded differently), until Bob is able to retrieve the source block (the bit vector). Upon this, Bob knows the set of subcarriers to use for correct reception at Alice. He uses only those sub-carriers to send legitimate symbols (from now on), and acknowledges the receipt of the bit vector. He also indicates (again with a bit vector), the subcarriers that are suitable for him, for reception in the presence of the jammer. At this point, both Alice and Bob are aware of the relatively unaffected subcarriers at each other's end.

Encrypting CSI: It is possible for the jammer to sniff the information encoded in the above message exchange. If it is able to retrieve the information with regards to the good

subcarriers at Alice or Bob, it can (a) skew its power allocation (when transmitting) on the subcarriers to increase the interference on these specific subcarriers and/or (b) construct and send fake CSI information to Alice and Bob that aids the jammer's goal. To prevent the jammer from gleaning the CSI information, we encrypt the bit vector using a symmetric key that it either pre-provisioned, or securely established via an authenticated key exchange protocol, as discussed earlier.

Summary: In summary, JIMS consists of the three steps described in each of the previous subsections. An algorithmic representation of JIMS is provided in Algorithm 4.

Algorithm 4 JIMS Channel Measurement Procedure

Input: *received_signal* is the physical signal received from the antenna.

Output: *channels_vector* is the selected channel vector.

Initialization: *channels_vector* $\leftarrow 0$

$RJSNR\_vector \leftarrow$ process(*received_signal*)

$\eta \leftarrow$ calculate_selection_threshold(SINR_vector)

for $i \leftarrow 0$ **to** 48 **do**

    channels_vector (i) = decide(η, $SINR\_vector(i)$) **if** *channels_vector (i) == '1'* **then**
        counter $\leftarrow$ *counter* + 1

    end

end

return *channels_vector*

4.4.4 JIMS with Power Allocation (JIMS-PA)

Thus far, JIMS simply identified those subcarriers that were relatively unaffected by the jamming signal from Eve, and used those subcarriers for the exchange of information between Alice and Bob (in Eve's presence). Since, the information on the other subcarriers, i.e., those that are heavily affected by Eve are relatively unusable, we ask the question, "Can we reallocate some of the power from such subcarriers, to the subcarriers that are being used in order to enhance the throughput?" The answer to this question is that, such a reallocation is possible to some extent. However, one cannot simply reallocate all the power onto the "good" subcarriers for two reasons. First, because of the spectral flatness regulations specified in the 802.11 standard (specifically 802.11n) [9], the difference in the powers allocated to two subcarriers cannot exceed 2 dB. Second, if we blindly assign high powers to the good subcarriers, Eve will notice the anomaly, and can target those subcarriers. Thus, we can only reallocate powers to some extent, and we seek to do so here while adhering to the first constraint.

Specifically, let us assume that Alice has learnt of the SINRs experienced by Bob on each of the composite subcarriers. Based on the received SINR information, Alice now seeks to maximize the throughput, $\tau_a(N_d)$, by finding (a) the appropriate number of "good" subcarriers, N_d, (b) the best bit-rate or modulation for use (M_c), and (c) the optimal power reallocation strategy as discussed above.

Power units: Before formally defining the problem, we define what we call "power units". As mentioned earlier, the spectral flatness constraint requires that we limit the difference in transmission powers between any two subcarriers to 2 dB. This in turn implies

that we can only remove at most 1 dB, or add at most 1 dB to a subcarrier. Conservatively, we limit the power removed/reallocated, from/to any subcarrier, to 0.75 dB. It is hard to consider all possible power allocations, by considering the transferred power quantums to be real valued. Therefore, we reduce the search space by quantizing the 0.75 dB budget into discrete power units. We set a power unit to be equivalent to 0.08333 dB (other settings are possible). Power reallocations across subcarriers is always in terms of a "number" of power units (at most 9 units can be transferred with our setting).

Our objective: Now, we formally define the problem to be a throughput maximization problem as follows:

$$\begin{aligned} \underset{N_d, M_c, \mathcal{X}_\Sigma}{\text{maximize}} \quad & \tau_a(N_d) \qquad (4.3) \\ \text{subject to} \quad & N_d \subseteq N \qquad \text{(I)} \\ & M_c \in \{2,\ 4,\ 16,\ 64\} \qquad \text{(II)} \\ & \mathcal{X}_i \leq \mathcal{Y} \quad \forall i \qquad \text{(III)} \\ & \mathcal{X}_\Sigma = \sum_{i=1}^{N_d} \mathcal{X}_i \leq \hat{\mathcal{X}}(N_d) \quad \text{(IV)} \end{aligned}$$

where, N_d is the subset of subcarriers selected for communication and N is the set of all subcarriers. $\mathcal{X}_i$ is the number of power units that allocated to a subcarrier $i \in N_d$, $\mathcal{Y}$ is the maximum number of power units that can be removed or added to a subcarrier. $\hat{\mathcal{X}}(N_d)$ is the maximum total power that can be reallocated (discussed later).

In the above formulation, Alice seeks to maximize the throughput by appropriately selecting a set of subcarriers, N_d, the appropriate modulation M_c on these subcarriers, and the best power allocation strategy. The last two constraints limit the maximum power

transferrable to a subcarrier, and the total power that can be transferred.

The maximum power available for reallocation (referred to as the total power budget) depends on the excess power that can be removed from *usable* subcarriers (this is just a reallocation of power from among the good subcarriers), $\mathcal{X}_j^{excess}$ and the amount of total power units that can be taken from the unused subcarriers. A used subcarrier has an excess power if its SINR value exceeds the minimum required SINR threshold for the current modulation. This threshold is different for different modulation schemes. Similar thresholds has been used in SNR based rate adaptation schemes [149]. We use the mapping provided in section 4.3 (specifically Fig. 4.9) between subcarrier power and SINR in order to calculate the excess or requirement of the power on an subcarrier.

The available power budget for a subset of N_d subcarriers can be expressed as:

$$\hat{\mathcal{X}}(N_d) = (|N| - |N_d|)\mathcal{Y} + \sum_{j=1}^{|N_d|} \mathcal{X}_j^{excess} \tag{4.4}$$

We reallocate power units to usable subcarriers to ensure that the sender is able to transmit at a higher rate than before and/or be able to convert "bad" subcarriers to "good" subcarriers. Thus, this process increases the overall capacity in the presence of the active jammer.

Considering all possible power reallocations towards finding the maximum possible throughput, results in an exponential number of possibilities. Specifically, there are $O(\mathcal{Y}^{|N_d|})$ combinations as per which, power can be assigned to the $|N_d|$ subcarriers. The number of ways by which power can be *removed* from unused subcarriers is $O(\mathcal{Y}^{|N|-|N_d|})$. There are M_c modulation types. Hence, the number of combinations for the power reallocation to be considered is $O(M_c\mathcal{Y}^{|N_d|}\mathcal{Y}^{|N|-|N_d|})$ or simply $O(\mathcal{Y}^{|N|}))$. To reduce computational

complexity, we propose a heuristic that runs in polynomial time and is independent of $\mathcal{Y}$.

Algorithm 5 JIMS-PA Algorithm

Input: $\vec{S}$ received SINR vector

Output: N_d, M_c, $\vec{\mathcal{X}}$ the selected power strategy.

Initialization: $\vec{S} \leftarrow \text{sort\_dec}(\vec{S})$ $\tau_{avg}^{BPSK}(N_d - \{i\}) \leftarrow 0$ $N_d \leftarrow \phi$;

for $i \in N$ **do**
  $N_d \leftarrow i$ $\tau_{avg}^{BPSK}(N_d) \leftarrow$ calculate_throughput()
  if $\tau_{avg}^{BPSK}(N_d) > \tau_{avg}^{BPSK}(N_d - \{i\})$ **then**
    $\vec{\mathcal{X}} \leftarrow$ perform_power_reallocation() $M_c \leftarrow$ calculate_throughput_and_MCs()
  end
  else
    is_success $\leftarrow$ call Algorithm 3 **if** *is_success* == *true* **then**
      $\vec{\mathcal{X}} \leftarrow$ perform_power_reallocation() $M_c \leftarrow$ calculate_throughput_and_MCs()
    end
    else
      break
    end
  end
end

Details of JIMS-PA: Let us assume, for simplicity that Alice is communicating with Bob in the presence of a jammer. Alice transmits a known pilot using all the subcarriers. Upon receiving the pilot, Bob calculates the per subcarrier SINR and sends the computed (SINR) values in an ACK/NACK packet to Alice (Note that here the raw SINR values are sent as opposed to simply a bit vector that indicates the good subcarriers). Then both

Alice and Bob apply JIMS-PA as in Algorithm (5).

Initially, JIMS-PA sorts the subcarrier SINR values in descending order. At each step a single subcarrier, i, from the sorted subcarrier list, N, is considered ($i \in N$) as discussed below. JIMS-PA initializes two subcarrier sets; usable (N_d) and unusable ($N - N_d$). In the beginning, the usable set is empty. In each step, a new subcarrier (specifically the subcarrier that supports the highest SINR) from the unusable set is considered for addition to the usable set. With this new subcarrier, say i, let us assume that the cardinality of the usable set is N_d. JIMS-PA then calculates $\tau_{avg}^{BPSK}(N_d)$, the throughput using BPSK modulation, considering the subcarriers in the usable set (throughput calculation discussed later). It compares this throughput with that achieved without i, i.e., $\tau_{avg}^{BPSK}(N_d - \{i\})$ (this is the throughput with the usable subcarrier set from the previous step). If the throughput degrades by including the new subcarrier for BPSK modulation (less vulnerable to errors) then it will degrade with higher modulations (the packet error probability on this subcarrier would be worse for higher modulations such as QPSK, QAM16, etc.). Thus, at this point there are two possible cases; (1) the value of $\tau_{avg}^{BPSK}(N_d)$ is greater than $\tau_{avg}^{BPSK}(N_d - \{i\})$ or (2) the value of $\tau_{avg}^{BPSK}(N_d)$ is less than $\tau_{avg}^{BPSK}(N_d - \{i\})$. We call these Case 1 and Case 2, and elaborate on them below.

Case 1: In this case, JIMS-PA adds subcarrier i to the usable set. It then considers the reallocation of $\mathcal{Y}(|N| - |N_d|)$ power units from the subcarriers in the unusable set, to those in the usable set. Beginning with the poorest subcarrier (lowest SINR) in the usable set, it incrementally assigns power, one unit at a time. It ensures that it does not violate constraints (III) and (IV) in the maximization formulation 4.3, when performing

the power reallocation. It then does an internal reassignment of powers from among the subcarriers in the usable set, and determines that best applicable modulation scheme (in terms of the achievable throughput) with the resulting SINR values (recall the discussion of how the increase in power can be mapped onto increases in SINR values from Section 4.3). It also computes the maximum achievable throughput with the current set of subcarriers in the usable set.

Algorithm 6 Subcarrier Revival Algorithm

Input: $\tau_{avg}^{BPSK}(N_d - \{i\})$, Subcarrier i

Output: *is_success* subcarrier can be revived or not

Initialization: $is\_success \leftarrow$ false

for $j \leftarrow 0$ **to** $\mathcal{Y}$ **do**

add_power_units_to_i(j) $\tau_{avg}^{BPSK}(N_d) \leftarrow$ calculate_throughput() **if** $\tau_{avg}^{BPSK}(N_d) > \tau_{avg}^{BPSK}(N_d - \{i\})$ **then**

$is\_success \leftarrow$ true

end

end

Case 2: In this case, it is clear that simply adding subcarrier i to the usable set will be in fact detrimental to the throughput. However, it may be possible to *revive* or make subcarrier i usable via power reallocation. Thus, JIMS-PA adds the maximum possible power (transferred from subcarriers in the current unusable set) to subcarrier i. If at this point, the throughput with the added subcarrier exceeds that with BPSK computed in the previous step, JIMS-PA proceeds as with Case 1. If not, the process stops. The subcarriers (excluding i) in the usable set are the subcarriers chosen for use. Power reallocation is

then applied formally to this set, and communications now take place using this set of subcarriers.

There are three stopping conditions for JIMS-PA; *(i)* there is no power budget left for reallocation, *(ii)* a subcarrier cannot be revived, and *(iii)* all the subcarriers are added ($N_d = N$). JIMS-PA declares a solution when one of these three conditions is satisfied. Upon reaching a solution, JIMS-PA returns the set of subcarriers(N_d), the modulation (M_c) to use and the per subcarrier power allocation ($\vec{\mathcal{X}}$).

Computational complexity: It is easy to verify that the run time for JIMS-PA is $O(|N| \times M_c \times \mathcal{Y})$, where $|N|$ is the total number of sub-carriers, M_c is the number of available modulations and $\mathcal{Y}$ are the available power units to be assigned. In brief, JIMS-PA iterates over $|N_d|$ subcarriers and for each sub-carrier, i, it iterates over the available modulation schemes to select the best modulation with power redistribution. Since the subcarriers are a priori sorted, power redistribution only takes $O(N)$ time.

Computing the throughput with a given set of subcarriers: Next, we present the calculation of the throughput based on per subcarrier SINRs. In order to make the analysis generally applicable with different transmission technologies and MAC layer protocols (WiFi, LTE, WiMAX, etc.), we do not consider MAC layer-specific packet retransmissions in our computation and experiments.

The throughput τ_a depends on three factors (1) the maximum number of re-transmissions, (2) the duration of the OFDM symbol, (T_s), and (3) the packet error probability (PER), p_e. The PER for a set of subcarriers (assuming that the bit errors occur independently in a uniform manner throughout the packet) is:

$$p_e = 1 - \prod_{i=1}^{|N_d|} (1 - p_b(i))^{L/|N_d|} \tag{4.5}$$

where p_e is a function of the bit error probabilities (denoted by $p_b(i)$) on each of the subcarriers that are used and L is the packet $p_b(i)$ depend on the modulation in use and the SINR value. We simply use the *erfc* function [42] to calculate $p_b(i)$, given M_c and the SINR.

The average transmission time, T_{avg}, of a packet is a function of the expected number of transmissions, $E(R)$ and the packet transmission time (given the set of subcarriers), $\rho(N_d)$. Specifically,

$$T_{avg}(N_d) = \rho(N_d) \times E(R) \tag{4.6}$$

If the maximum number of retransmissions possible is R, the expected number of transmission attempts is given by,

$$E(R) = \sum_{r=1}^{R} r p_e^{r-1}(1 - p_e) = \frac{1 - (R+1)p_e^R + Rp_e^{R+1}}{(1 - p_e)} \tag{4.7}$$

The packet transmission time for a given number of subcarriers, $|N_d|$, can be calculated as follows

$$\rho(N_d) = \frac{L}{|N_d| M_I} \times T_s \tag{4.8}$$

where T_s is the OFDM symbol duration, L is the packet length and M_I is the modulation index ($M_I = \log_2 M_c$). In the above, the packet transmission duration is simply computed to be the product of the number of simultaneously transmitted symbols in a packet and the symbol duration. From Equations 4.6, 4.7 and 4.8, the average transmission time $T_{avg}(N_d)$ is given by:

$$T_{avg}(N_d) = \frac{L}{|N_d| M_I} \times T_s \times \frac{1 - (R+1)p_e^R + Rp_e^{R+1}}{(1 - p_e)} \tag{4.9}$$

The average throughput, $\tau_a(N_d)$, is given by

$$\tau_a(N_d) = \frac{1}{T_{avg}(N_d)} \tag{4.10}$$

4.5 Implementation and Performance Evaluation

In this section, we evaluate our proposed framework via extensive real testbed as well as simulation experiments.

4.5.1 Testbed Implementation

We implement our proposed schemes on top of the WARPLab framework [29], which runs on WARP boards in real time. The proposed framework is implemented as a thin layer between PHY and MAC layers to determine the most appropriate subcarriers. The receiver communicates the obtained information through short messages with the transmitter (as described in Section 4.4). We modify the transmitter to perform power allocation to the selected subcarriers (described with JIMS-PA) upon receiving CSI feedback, included in ACK/NACK packets from the receiver. The ACK/NACK packets are encoded using Raptor codes. Each ACK/NACK packet contains a subcarrier vector of 48 bits, which correspond to the 48 OFDM subcarriers. A bit with value one means that the corresponding subcarrier is selected. In all our experiments the CSI feedback rate is set to 1 sec. We follow the 802.11n guidelines and set the spectral flatness limit to +/- 2 dB. The maximum number of power units is set to 9. We implement three jamming models; periodic jamming, random jamming and continuous jamming. While JIMS primarily targets active jamming, it should return to default operations when the jammer is off (meaning, all subcarriers should be used with default powers); we examine if this holds with periodic and random jamming. The jammer generates packets with a uniform power distribution across all sub-carriers.

For the periodic jamming, the adversary transmits the jamming signal periodically, thereby effecting the legitimate communication for the period it is ON. The jamming pulse lasts for one second. For the random jamming, the adversary transmits the jamming signal at random time intervals for a random time span between 0.5 & 2 seconds. We use the same experimental setup and parameters described in section 4.3.

Raptor codes: To encode ACK/NACK packets, we implement Raptor based Forward Error Correction (FEC) as specified in the standard [16]. We first divide the CSI packet into a number of source blocks, Z. Each source block consists of k OFDM data symbols. Raptor encoding is applied independently on each symbol. The encoder generates l encoded symbols for the k data symbols in a block that are uniquely defined by a set of constraints [16]. For our experiments we set $l = 2k$; k varies between 10 and 50.

The original data symbols can be recovered from any subset of the encoded symbols of size equal to or slightly larger than the number (k) of original symbols. In addition, the coding rate varies according to the observed average $RJSNR$. We assume that the receiver knows the symbol size (bits that make up the symbol) and the number of symbols, k, in the source block. Upon receiving the encoded symbols, the receiver passes the received symbols to the decoder. reception of k or more encoded symbols, allows the block to be recovered with some non-zero probability. By forming a matrix and performing Gaussian elimination on k (or more) encoded symbols, the decoder retrieves the source symbols. If the decoding fails, the decoder waits for more symbols and tries again.

The Advanced Encryption Standard (AES): To prevent an adversary from sniffing the CSI packet, legitimate transceivers employ the AES based encryption for ACK/NACK

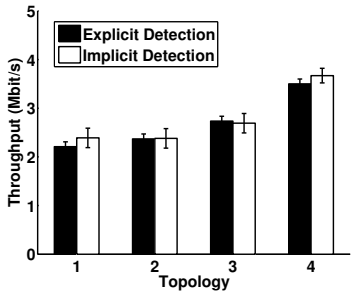

Figure 4.10: Throughput: Explicit vs. Implicit detection.

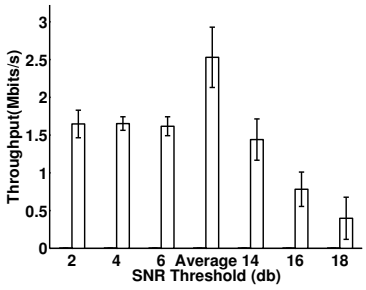

Figure 4.11: SINR threshold selection.

packets, using a shared 128-bit key. We use the publicly available AES implementation in [3] and integrate it with our framework in WARPLab.

4.5.2 Experimental Results

In this section, we evaluate the proposed scheme with/without power adaptation and demonstrate its practical applicability. The testbed setup is identical to what was described in Section 4.3. By default, the jammer continuously emits malicious interference.

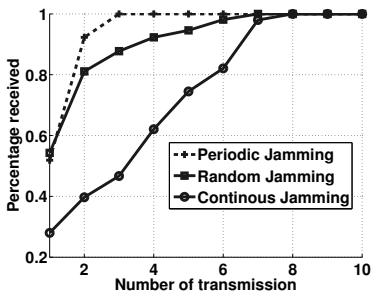

Figure 4.12: Number of transmissions with JIMS for encoded CSI packets.

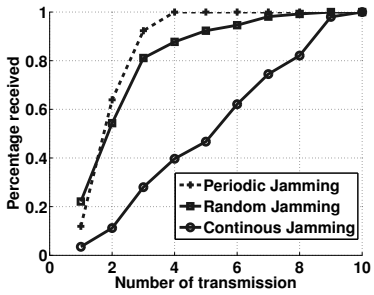

Figure 4.13: Number of transmissions with JIMS-PA for encoded CSI packets.

Explicit vs. Implicit jamming approaches for determining the good subcarriers: We first compare the performance of explicit and implicit approaches for determining the good subcarriers. Fig. 4.10 shows the throughput achieved with the explicit and implicit approaches with different network settings. We create four different network topologies as shown in Fig. 4.1-4.3 with a legitimate sender-receiver pair and a continuous jammer as an attacker. We see that both schemes perform equally well. The comparable performance is expected since jamming is the dominant effect on the throughput (with both schemes); the impact of the wireless channel (fading) is less pronounced in comparison with that of jamming. Both approaches (albeit in different ways) use the common strategy of identifying those legitimate subcarriers that are affected by the adversary. In one case, the receiver tries to explicitly recognize the impact of jamming while in the other case, the

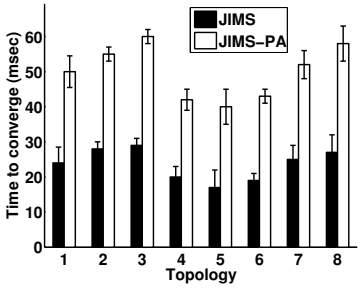

Figure 4.14: Convergence Time with JIMS and JIMS-PA.

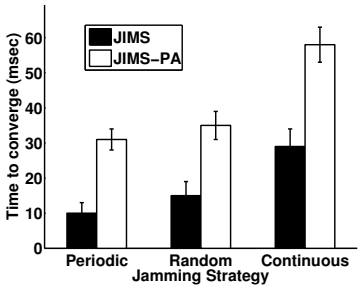

Figure 4.15: Convergence times vs different attack strategies.

jamming signal is treated as interference and the impact is implicitly captured. In both cases, JIMS leverages the least affected subcarriers for communication.

Choice of SINR threshold: As we discussed earlier, an inherent challenge in subcarrier selection is choosing the appropriate SINR thresholds. Since both the explicit and implicit approaches have similar performance, we only consider the SINR to demonstrate the effectiveness of our approach of using the average (recall Section 4.4.2) as the threshold for subcarrier selection. For this experiment, we use a legitimate sender-receiver pair, which is communicating in presence of a continuous jammer. We try 6 different static SINR thresholds (from low to high values) and compare the performance with their use (in JIMS) against the average SINR threshold that we advocate. Fig. 4.11 shows the throughput achieved by the legitimate sender-receiver pair with the different SINR thresholds. We

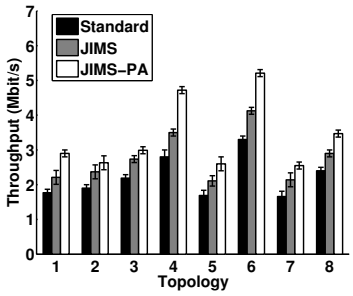

Figure 4.16: Throughputs with Standard, JIMS and JIMS-PA.

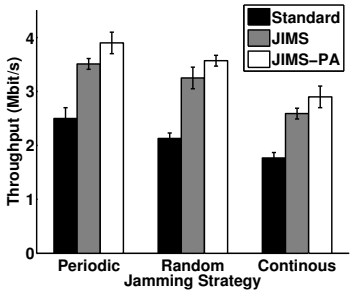

Figure 4.17: Throughputs with different attack strategies.

observe that average SINR threshold achieves a higher throughput than any other SINR thresholds; specifically it can achieve up to 5 times more throughput than the poorest static threshold. We have validated that this average threshold is also robust to changes in network topology and environment in contrast to static SINR thresholds but do not show the results due to space constraints.

CSI exchange: Next we evaluate the effectiveness of the CSI exchange under different jamming attacks. The metric of interest is the number of encoded (re-)transmissions required to deliver CSI packets. Figs. 4.12 and 4.13 show the percentage of successfully received packets with respect to the number of transmissions for JIMS and JIMS-PA. The results average is computed over 20 runs for a given topology. In particular, we fix the position of the transmitter and the receiver. Thereafter, we position the jammer in specific

position inside the grid to create a topology. With a periodic jamming attack, 80% of CSI packets are delivered with three encoded transmissions, while with the random jammer, four transmissions are required to deliver the same number of CSI packets (the skewed periods cause this effect since sometimes the jamming periods are longer) with both schemes. As one might expect, the continuous jamming attack is the hardest to cope with; we need eight transmissions to deliver 80% of the CSI packets. We observe that in the worst case, the delivery of a CSI packet requires ten encoded transmissions.

Convergence times of the CSI exchange: JIMS's performance is critically dependent on the exchange of CSI packets. However, as mentioned earlier, these packets are sent using all the subcarriers with Raptor encoding and suffer from interference from the jammer. We examine if this first step in JIMS's design can become a communication bottleneck by measuring the time it takes to successfully exchange CSI packets and begin the usage of the good sub-carriers; we refer to this as the convergence time. We measure the convergence time under 3 attack scenarios and with various network topologies.

Fig. 4.14 depicts how the convergence time varies for 8 different network topologies. For each topology the results are averaged over 20 trials. We see that the observed maximum convergence time is less than 35 ms. This demonstrates that JIMS converges fairly quickly. Fig 4.14 shows that JIMS-PA requires about 60 msec to converge in the worst case. This is because JIMS-PA has to send more data (SINR information) in the ACK/NACK packets as discussed earlier; this larger data transfer is especially difficult due to the jamming signal. Thus, JIMS-PA takes a longer time to converge. If the nodes under consideration (both the legitimate transceivers and the jammer) are relatively static (small amounts of motion),

these convergence times are sufficiently small in terms of overhead and can allow sustained use of the "good" subcarriers for relatively, much longer periods.

Fig. 4.15 depicts the impact of the attack model on the convergence times with JIMS and JIMS-PA. It is immediate that continuous jamming is the most hurtful while the other two attack models have similar (lesser) impact. Even with continuous jamming however, in the topologies considered, the maximum convergence time is 25 ms. Fig. 4.15 shows that JIMS-PA needs more time to converge compared to JIMS in all cases as expected. With continuous jamming, JIMS-PA has a convergence time of approximately 60 msec. In the best case scenario, it requires about 32 msec to converge.

Throughput performance: Next, we compare the performance of JIMS and JIMS-PA against a standard system which utilizes all the subcarriers for communication. Fig. 4.16 demonstrates the performance of three schemes in terms of throughput under different network topologies in the presence of a continuous jammer. Results are shown for 8 network topologies, as described in section 4.3. We see that JIMS outperforms standard 802.11 system by up to 65%, while JIMS-PA does so by up to 75%. JIMS-PA provides an additional gain over JIMS (about 10%) in spite of the increased overehad during the CSI exchange process.

In Fig. 4.17, we depict the performance of the schemes in the presence of three attack strategies viz., periodic, random and continuous jamming. The results with the continuous jamming are as before. With random and periodic jamming, all the three schemes perform better. The JIMS schemes outperform the standard scheme to a slightly lesser degree. For example, JIMS-PA outperforms the standard system by up to 56%.

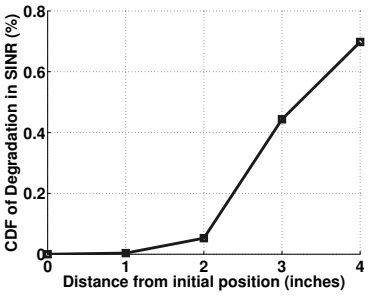

Figure 4.18: Degradation in SINR with mobility.

Impact of mobility: To examine the effect of user mobility on the performance of JIMS, we move the receiver away or towards the transmitter at a constant speed from its original position and measure the SINR at fixed intervals. Fig 4.18 shows the CDF of the degradation in the average SINR for a mobile receiver as a function of the distance that the receiver has moved from its initial position. The result suggests that the degradation in the average SINR with small extents of mobility (moving a smartphone or walking) is minimal.

We find that when the mobile receiver moves 1 or 2 inches away from its initial position, the throughput gains remain intact. However, if the receiver moves over a distance like 3 inches, the average SINR degrades more than 40%. Thus, the average throughput also decreases. At this point, the CSI values need to be exchanged again and the SINRs recomputed.

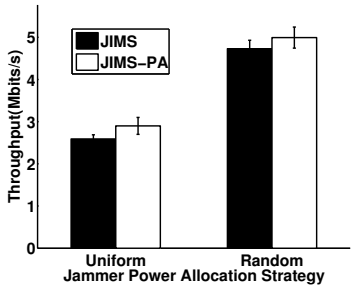

Figure 4.19: Throughput comparison for various jamming power allocation strategies.

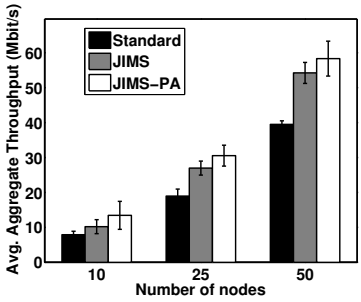

Figure 4.20: Throughput with increased node density.

4.5.3 Simulations

Simulation setup: In order to examine the performance of JIMS and JIMS-PA in larger scale settings, we have implemented both schemes on NS3 version 13, using a detailed PHY layer model called PhySimWifi [14]. We use an on off client server model to generate application level traffic. For the MAC and network layers, we use 802.11a with Friis propagation loss model. The packet size is fixed to 1500 bytes and the transmission power is set to 20 dBm with the background noise varying between -90 and -99 dBm. We have averaged the results over 25 runs where each run lasts for 100 seconds. We consider random and grid topologies, and place the jammer in arbitrarily chosen positions. To ensure transmissions even when sensing the jamming signal, we disable carrier sensing. Again, continuous emissions by the jammer occur in the default setting.

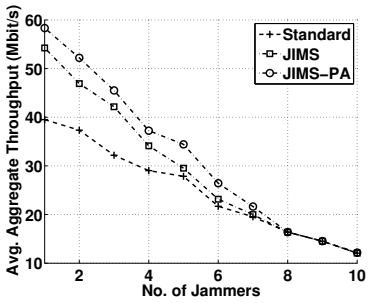

Figure 4.21: Effect of multiple jammers on throughput.

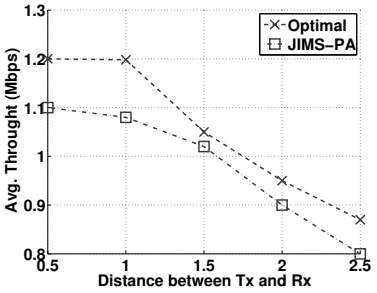

Figure 4.22: A comparison between JIMS-PA and the optimal stratgy

Scalability: We plot the average aggregate throughput with JIMS and JIMS-PA with different numbers of nodes in the presence of a jamming attack. Fig. 4.20 shows the aggregate throughput with JIMS, JIMS-PA and the standard system with increasing node density. In this scenario, we use a single continuous jammer which is trying to disrupt the communication in its neighborhood. With increasing node density, the benign interference levels also go up in addition to the interference from the jammer. As we can see from Fig. 4.20 that JIMS-PA outperforms the standard system by upto 70%. In addition, JIMS-PA performs better than JIMS by 30% in the best case due to intelligent power reallocation.

Multiple jammers: To show the resiliency of JIMS against multiple jammers, we varied the number of jammers in the network (benign interference still exists) and calculated the aggregate throughput for both JIMS-PA and JIMS. Fig. 4.21 shows the

average aggregate throughput with an increasing number of jammers, in comparison to a case with a single jammer with a 50 node random network topology. We see that JIMS-PA and JIMS both suffer as we increase the number of jammers. However, in a two Jammer case, the throughput loss is about 15%; this still is better than the standard case with a single jammer. If there are more than 7 jammers, JIMS-PA and JIMS cannot restore any throughput since it is likely that all subcarriers are now affected by jamming.

Optimal Strategy: We compare the average throughput of JIMS-PA with an optimal solution (continuous power allocation and the optimal selection of the (best) subcarriers via exhaustive search) for a fixed number of subcarriers and modulation type. Specifically, to keep the computational complexity low,we fixed the number of subcarriers to 4 and employ only two modulation schemes, namely, BPSK and QPSK. We varied the distance between the transmitter and the receiver and randomly position the jammer. The results are averaged over 10 runs. Each run lasts for 90 seconds. As shown in Fig. 4.22, the throughput of JIMS-PA is only slightly worse ($\leq 15\%$). Note that given the small number of subcarriers available for selection, this worst case scenario is pretty good. If there is a much larger number (as is the case in practice) one can expect the performance if JIMS-PA to be even closer to the optimal allocation. This is because with a larger number of subcarriers there is a greater flexibility for power re-distribution.

4.6 Conclusions

In this paper, we seek to mitigate the impact of an active jammer (e.g., a reactive or wideband jammer). To do so, we exploit the inherent features of OFDM. Specifically, we

perform experiments that show that the jamming signal has different fading levels with respect to different OFDM subcarriers. We propose JIMS, a jamming interference mitigation scheme, using which, transceivers can identify subcarriers that are relatively unaffected by jamming and utilize them for communications. We show that JIMS restores throughput up to 75%, in the presence of an active jammer via experiments on our WARP testbed. At this time, we rely on prior schemes to detect the jammer, and utilize JIMS only when a jammer is detected. Integrating JIMS with such detection schemes effectively will be considered in future work.

Chapter 5

Catch Me if You Can: VM Migration for Thwarting Malicious Co-Residency on the Cloud

5.1 Introduction

Infrastructure-as-a-Service (IaaS) providers allow VMs that belong to different users, to share the same physical infrastructure. Thus, the risk of sharing a physical machine with a potential malicious VM is very real [262, 211, 265]. Once an attack VM is able to co-reside with a victim VM on the same physical machine, it can launch arbitrary attacks (e.g., using side channels to achieve information leakage) to compromise the security of the victim VM. Although providers continuously make improvements to better isolate resources across VMs, new vulnerabilities are expected to emerge as hardware architectures and hypervisor

technologies evolve [161].

Prior to launching such attacks however, the attacker typically needs to place a malicious VM on the physical machine that houses the victim VM (co-reside with the victim). The process of performing co-residency today requires the attacker to launch its VMs, use side-channels to ascertain co-residency, and upon failures terminate and repeat the process. Once co-residency is achieved, the longer a victim VM resides on the same physical machine occupied by the attacker VM, the higher is its risk of being compromised.

In this paper, we have two main objectives. First, we seek to get an in depth understanding of the ways and the effectiveness with which an attacker can achieve co-residency with a victim VM in practice. Towards this, we undertake an extensive experimental effort on Amazon's EC2 cloud infrastructure, to understand the side channels that an attacker can use to ascertain co-residency with a victim VM. En route, we discover a new set of very stealthy and highly effective timing based side channels that can be used today to ascertain co-residency.

Migrating a VM is a way of mitigating long periods of co-residency with an attacker VM [179]. As our second objective, we seek to to determine under what conditions a victim VM should be migrated to minimize its co-residency time with an attacker, given a bandwidth/downtime cost the user of the VM is willing to bear. Towards this (based on the above experimental studies) we formulate a set of guidelines, which are based on the time that a victim VM has resided on a host machine and the very side channel that the attacker could have used to ascertain co-residency. We perform extensive experiments on our in house cloud (built using CloudStack [71]) to demonstrate that our guidelines can

drastically reduce the times for which a victim VM co-resides with an attack VM with low costs in terms of downtimes and bandwidth.

To summarize, our contributions are as follows:

- We carry out extensive experiments on Amazon EC2, arguably the most popular cloud provider, to develop a comprehensive understanding of the efficacy of an adversary in successfully co-residing its VM with a victim's VM.
- We build a simple model that can provide us with rough estimates of how long it takes for an attacker with varying capabilities, to successfully co-reside with a victim. The model requires very few measurements and can provide guidelines on how often VMs should be migrated in different scenarios.
- We discover a set of new highly effective timing based side channels that can be used by an attacker to determine if any of its VMs co-resides with a targeted victim VM. Our side channel provide the highest accuracy in ascertaining co-residency as compared to other previously proposed side channel tests ($\approx$ 86 %), but with lower false positive rates. In addition, we believe that they are much harder to detect than the latter since they do not create explicit congestion on a shared resource.
- We consider VM migration as a countermeasure to thwart malicious co-residency and develop a set of guidelines on when to invoke victim VM migration given a cost budget in terms of the bandwidth expenses and downtimes that the user is willing to tolerate. We perform extensive experiments on an in house KVM-based private cloud (users cannot invoke live migrations on commercial clouds today) to evaluate our guidelines. The results show they can drastically reduce the times for which a victim VM co-resides with

an attack VM. Specifically, with very reasonable performance costs (of the order of MB of bandwidth and seconds of downtime per day, per VM migrated), the fraction of time that the victim VM co-resides with an attack VM can be limited to about 1 %.

Roadmap: In §5.2 and in §5.3 we present related work and our threat model, respectively. We present an experimental study on co-residency on Amazon EC2 and showcase our new attacks in §5.4. We present a model to estimate the time taken to achieve co-residency in §5.5. Our mitigation guidelines are derived in §5.6. We present our evaluations in §5.7. A discussion on how the migration guidelines influence costs while mitigating side channels targeting information leakage is provided in §5.8. We conclude in §5.9.

5.2 Related work

Side channel attacks targeting information leakage: Side channel attacks exploit physical information leakage such as timing information, cache hits/misses, power consumption etc. This information is typically obtained based on the usage of shared resources (e.g., cache). There are several side channel attacks on cloud tenants that have been previously studied [262, 211, 46, 128, 130, 274, 245]. Side channel attacks can target different shared resources; examples include the cache, shared storage etc. Based on the attack, the time taken to successfully extract information ranges from the order of minutes to hours.

Co-residency with a victim process: For almost all side-channel attacks reported, an attack process (VM) will need to co-reside with the victim process (VM) on the same physical machine. The attacker will need to launch its VM and use some kind of side

channel to ascertain if has co-resided with a victim. Side channels have also been proposed for enabling co-residency checks (e.g.,[247]). However, these prior efforts do not provide a comprehensive understanding of how effective these are in terms of their accuracy and the time it takes for an attack VM to successfully co-reside with a victim VM.

Reverse engineering the algorithm for determining the placement of VMs empirically, as in [247] and [266], although hard, might be useful in the short term. However, placement algorithms are likely to dynamically change over time [266]. Because of this, one can largely consider the placement algorithms to be opaque (and possibly customized to users); instead of trying to reverse engineer the process, we develop measures to determine co-residency, and construct a model which provides rough estimates of the average time taken for acheiving co-residency (regardless of the nuances of placement). Further, cloud providers have ensured that many co-residency checks proposed much earlier (e.g., [211, 273]) are no longer feasible[1]. To the best of our knowledge, we are the first to propose network timing based side channels for ascertaining co-residency; note that there are other network timing based attacks previously studied but they are quite different (e.g., [231]).

Defending side-channel attacks: Cloud providers as well as the research community is continuously looking for ways to improve resource isolation which can help defend against side channel attacks. Efforts such as [202] and [158] introduce random delays while accessing a resource to thwart timing based side channel attacks. [204] and [73] employ software level defense mechanisms as countermeasures against cache based side channel attacks; for example, the idea in [204] is to obfuscate the program at the source code level to provide the illusion that many extraneous program paths are executed. If an attacker

[1]We have also experimentally verified that this is the case.

conducts a prime and probe attack (where he primes the cache and probes for determining changes to cache sets) his observations will be skewed. Although the current methods can defend against known side channel attacks, it is unclear if there exist other type of attacks that are unknown to the research community. Other vulnerabilities could appear in the future due to advancements in computer architecture and hypervisor technologies. Mitigating malicious co-residency can significantly alleviate the attacker's ability to launch such attacks.

VM migration to mitigate side-channel attacks: VM migration has recently been considered to counter cloud-based side-channel attacks targeting information leakage in [179]. In brief, the authors of Nomad [179], model information leakage from side channel attacks over time and determine how often migration is needed. However, they assume that the attacker has successfully co-resided with the victim. Nomad also assumes that such decisions on migration to cope with side channels, are made by the provider and not the users of the VMs. The users may not be willing to accept the performance penalties (downtimes) that inevitably occur when VMs are migrated for such purposes. We make no assumptions on what the provider will do in terms of placement of VMs.

Unlike in Nomad, we try to minimize the occurrence and periods of successful malicious co-residency. We account for the time that an attacker takes in order to successfully co-reside with a victim; this can influence the costs associated with migrations (reduce the frequency). The users can choose when to migrate their VMs based on their risk averseness and the costs they are willing to bear.

In a realistic scenario, where users are likely to configure their own VM migration

policies (e.g., enable, disable, choose periodicity etc.), not accounting for the fact that the attacker takes time to first co-reside with the victim (as with Nomad) in addition to the time taken for an information leakage attack, for driving migrations will increase migration frequencies and thus bandwidth/downtime costs (discussed later). We later show that very frequent migrations could also adversely affect security when the attacker simply stays put on a physical machine (since the victim VM can potentially return to the very same machine).

5.3 Threat model

We assume that an attacker seeks to co-reside its VM on the same physical machine as a certain targeted victim VM and co-reside for as long as possible. First we consider a case where the attacker launches a set of VMs, repeatedly if needed, and attempts to have one of these attack VMs co-reside with a victim's VM. We assume that it has no knowledge or control over the policies followed by the cloud provider for VM placement (as with Amazon's EC2). We call such an attacker an "reactive attacker"; this scenario is reflective of what the attacker can do on today's commercial clouds.

Next, we consider the cases where (like any user) an attacker can choose to migrate its VM, if user driven migrations are allowed. We assume that the provider does not unilaterally perform migrations (without user requests) like in [179], since this may cause downtimes without user consent (which some users may not want to experience). An attacker VM could simply choose to stay put on its initial physical machine assuming that the target VM will (due to migration) will be placed on the same (physical) machine eventually.

We call such an attacker a "static attacker". Finally, we also consider a possibility that an attacker may choose to migrate periodically itself. We call such an attacker, a "periodic" attacker. In both of the above cases, we assume that an attacker continuously checks for co-residency (using one of the approaches discussed in Section 5.4), since the victim could now at any point, migrate to the machine on which its VM resides. Note that these attack strategies cannot be implemented and tested today on Amazon's EC2 (migrations are not viable as of today); we test them on our in house cloud in Section 5.7.

Once the attacker is able to verify with high accuracy that one of his attack VMs has successfully co-resided with the victim's VM on the same physical machine, an attempt is made to launch a previously proposed side-channel attack (described in Section 5.8) to successfully create a leakage of information from the victim. However, we do not explicitly focus on such side-channel attacks themselves in this work; we provide a discussion on the impact of our work on such attacks in Section 5.8. For simpilicity, we assume that the number of virtual machines owned by the victim remains unchanged, i.e. the number of virtual machines does not vary over small time scales (hours or days). We also assume that after a migration occurs, the attacker does not know "where the victim process has been migrated." We assume that it is not interested in triggering other attacks (e.g., causing a DoS attack by inducing repeated migrations).

5.4 Characterizing co-residency via experiments

We perform extensive experiments on Amazon's EC2 over a period of 5 months, to obtain an understanding of (a) the accuracy and (b) the time taken by an attacker to

Model	Virtual CPU	CPU Credits / hour	Mem (GiB)	Storage
t2.micro	1	6	1	EBS
t2.small	1	12	2	EBS
t2.medium	2	24	4	EBS
t2.large	2	36	8	EBS

Table 5.1: Instance Type Comparison

successfully (we define what mean by success below) co-reside its VM with a targeted victim VM (T_{CR}), while using different types of side-channels to verify co-residency. Specifically, we implement and test, multiple previously proposed ways of verifying co-residency (co-residency tests). In addition, we design new timing based co-residency tests that are stealthy and yet, very effective. We reiterate here that some of the previously proposed side-channel based tests to verify co-residency (e.g., [211, 273]) do not work anymore (as verified by our experiments) since cloud providers have taken steps to prevent them [274].

Categories of co-residency tests: We divide co-residency tests into two categories; controlled/internal and external. In internal co-residency tests (ICT), we control both victim and attacker VMs. These tests primarily demonstrate co-residency with high fidelity (and can serve as ground truth) but cannot necessarily be used by an external adversary. Specifically, we create contention on a shared resource using the attacker VM and measure the changes in access times to that resource experienced by the victim VM (as compared to the access times experienced with no contention). We assume that a co-residency test is successful if an associated ICT test classifies the test as being successful.

In the external co-residency tests (ECT), we only control the attacker VM. We exploit a service running on the victim VM to try to create contention on a (possibly) shared resource. The attack VM then compares the response times to the service with and without contention. Here, we also propose the use of new timing tests to decide whether

two machines co-reside or not. The accuracy of the ECTs (which represent the mode of operation of an adversary in a real setting) is assessed by comparing the result with that of a ICT. The time taken for successful external co-residency tests is a critical metric that we are interested in. In all of these experiments we assume an active attacker as discussed in Section 5.3.

General setup: We initiate 20 micro or 20 small instances each, for a victim and an attacker account (for ground truth validation). All the instances run Ubuntu 14.04 LTS [239]. We conducted our experiments on three different datacenters (us-west-2a, us-west-2b and us-west-2c). Four different side channels that are exploitable to verify co-residency, reported in the past three years, were implemented [247, 265, 273] and tested. A key contribution we make is the design of new, very effective, timing-based co-residency tests.

The victim VM hosts 3 services, each very different in nature; thus, the co-residency tests in the three cases, for identical set ups, take different times. The services are Taiga, ownCloud, and MediaServer. Taiga is an open-source project manager software that involves a mix of CPU, disk (frequent), and memory workloads. ownCloud is an open-source file hosting service (resembles Dropbox) that involves memory and disk intensive workloads. MediaServer is an open-source wiki-page server that involves a mix of CPU, disk (rare), and memory workloads. We use different type of VMs; their specifications are summarized in Table 5.1 (we use the Amazon EC2 jargon [83]). Later in the section, we provide further details on each specific experiment.

ICT experiments: In each experiment, as mentioned, the attacker and the victim have 20 VMs each. We end up with 400 possible combinations of attack and victim VMs

(20×20); thus, 400 co-residency tests will need to be performed for each considered shared resource. With four shared resources this translates to 1600 tests[2]. A comprehensive study using this approach would take a prohibitive amount of time, considering that the failure of co-residency would incur termination and relaunching of attack VMs. To speed up the process, we create a pre-configured VM image that can be readily instantiated. Second, we recognize that the failure of co-residency tests using certain shared resources, can with high probability suggest that other tests (on other shared resources) will also fail. Therefore, by ordering the tests, we drastically reduce the number of tests (by eliminating tests that are highly likely to fail). For example, the failure of the co-residency test that considers the memory bus as the shared resource (bus contention test) described in [247] (details later) will indicate that the attack and the victim process do not share the same CPU. Finally, for each attack VM, we perform a co-residency test with regards to a shared resource, with all the victim VMs in one shot. By using these reductions, we were able to reduce the time taken per run to 20 minutes, on average. Thus, by running experiments for 15 hours per day, we were able to test more than 50,000 pairs of attack and victim VMs.

ECT experiments: We go through a similar process (with all the optimizations outlined above) except that we can only induce workloads on victim VMs by sending external requests (e.g., to upload a new file onto the ownCloud server). Further, we will allow all 20 attack VMs to perform the tests simultaneously. If any of them detects the co-residency successfully, we further cross-validate the result with a follow-up ICT experiment. If the validation holds true, the process is deemed as success and stopped.

[2]We wish to point out here that these tests are primarily used as benchmarks and cannot be actually used by an external attacker.

Launching and termination: For every failed attempt at co-residency (the co-residency test fails with respect to a considered shared resource), the attacker must terminate his VMs and then re-launch them in an attempt to again successfully co-reside with his target victim. The time taken to launch and terminate these processes are denoted by t_l and t_d, respectively. These times will contribute to the overall time that an attacker will have expend, in order to successfully co-reside with the victim VM. Using the process described above, we experimentally quantify these times.

5.4.1 Implementation of prior co-residency tests

We implement and test previously proposed co-residency tests on EC2. In these tests, an attacker creates contention on various shared resources (e.g., cache, bus) and uses a side channel to determine if his process co-resides with a victim process. The time taken to perform a co-residency test is denoted by t_c; this time primarily depends on type of shared resource used to determine co-residency, as well as the type of service running on the victim VM. A successful co-residency test indicates with high probability that the attack and the victim VM share the same physical resources.

Bus contention based ICT and ECT: Bus contention tests were designed and evaluated in [262] and [247].

ICT: In the ICT, the attacker VM allocates a chunk of memory that is larger than the size of the last level cache (64 MB in our experiments). Is then misaligns the memory access pointer by adding an offset to it (two bytes). The issuing of an unaligned, atomic access operation (such as a read, or the XADD operations for x86 processors) [247] causes the locking of the memory bus. The causes the victim VM to see a significant increase

(around 3X) in the memory access time if both the VMs share the same physical machine.

ECT: In the ECT, the attacker engineers a set of external requests (such as HTTP or FTP) which cause the victim VM to access the memory bus; the request sizes will have to be larger than the last level cache (LLC) size [247]. Internally, the attack VM locks the memory bus (similar to what is done in the ICT). By comparing the response times in this case, with the response times without the locking of the memory bus, the attacker is able to determine if his VM co-resides with the victim VM (a significant increase is seen if the VMs co-reside).

LLC based ICT: Next, we test if two VMs share the same CPU by creating contention on the LLC [164]. If this causes an increase in the LLC access time for the victim VM, we conclude that both VMs share the same CPU. The LLC in our experiments is the L3 cache; its size is 25.6 MB with a cache line size of 64 bytes and an associativity of 20. The page size is 4096 bytes.

The attacker VM allocates 1 or more GB of memory and regularly reads and writes in multiple page size increments (to ensure a page miss for the victim VM). The victim VM allocates an L3 cache size (of 25.6 MB) and iteratively reads pages in order. If the two VMs share the L3 cache, a significant increase ($\approx$ 1.8 X) in the access time is observed by the victim VM.

ICT with L1 cache: Two VMs that share the same CPU do not necessarily share the same core. We test if two VMs share the came core via a contention based test on the L1 cache. The L1 cache size is 32 KB, its associativity = 8, and the page size is 4096 bytes. As in the previous test, the attack VM repeatedly evicts the L1 cache by requesting

data that is not in the cache. The victim process continuously tries to access the same data (e.g., a certain data structure) repeatedly, and measures the access times. With contention a 1.5-3X increase in the access time is experienced. Note here that even if the two VMs share the same CPU (the LLC test could yield a success), this test will fail if they do not share the same core. If two VMs share the same core, an active side channel attack can probe the contents of the L1 cache during execution [198].

5.4.2 A new ICT

Storage based ICT: The test seeks to determine if two VMs share the same disk. Although the idea of checking if a disk is shared is not new [266, 247], to the best of our knowledge, we are the first to design and implement an approach to determine co-residency based on disk storage access in a cloud environment. We find that this test does not inherently provide any advantage over the previously proposed ICTs discussed above (as shown later), but we present it nevertheless, for completeness. To create contention for disk access, the attack VM accesses a relatively large file ($>$ 6 GB). It repeatedly does this access while varying the block size from 512 Bytes to 8 MB. The victim tries to perform a storage operation; in our experiments, it copies a similar large file from storage and we measure the average time taken. For each block size used, we measure the average time taken by the victim for copying the file from storage. The attacker's goal is to cause an increase in the average seek time (compared to when it does not access the large file). For the Amazon EBS (Elastic Block Store) storage, a minimum increase of 33% in the total transfer time is observed when there is disk contention.

5.4.3 New timing based ECTs

Next, we propose a set of new, stealthy yet very effective timing-based ECTs.

ECT based on RTT timing behaviors: Today, cloud providers employ load balancers and multi-path routing to be able to dynamically handle high traffic loads and denial of service attacks [136]. However, it is reasonable to assume that packets which are destined to VMs that reside on the same physical machine are exposed to similar effects at a given time (same paths), and experience similar delays along the route. Thus, one might expect that while the behaviors change dynamically, the delays experienced by packets that are destined to VMs on the same physical machine exhibit consistent (similar) temporal variations.

In order to ensure that the results are not biased by hypervisor scheduling delays (typically the maximum time slice that a VM can obtain before being switched out of context $\approx$ 10 msec) we probe at coarse time granularities (i.e., the probing period is set to $\geq$ 100 msec). With this set up, we receive one response from each of the two VMs under consideration (the attack and the victim VM) per probe. If the two VMs are on the same physical machine a delay observed between the responses from the VMs is much less than the probing period; unless (in rare cases) where the physical machine load rapidly changes, these delays are of the order of OS scheduling delays. Otherwise, much higher variations are observed in the delays (because of different routes and traffic on those routes). In order to perform the comparisons, we normalize the observed values with respect to the maximum observed response times (to eliminate temporal variations in load across the paths taken by the probes).

We measure the round trip times (RTTs) continuously (by instantiating connections separated by randomly chosen periods) from an outside observer (attacker) to the two VMs in question. We then perform a time series analysis to determine whether the timing profiles observed with respect to the two VMs are very similar. We call this test the "behavioral timing test".

To ensure that we can accurately compare the time profiles, we instantiate the connections to the two VMs (almost) simultaneously. To measure the RTT, we primarily rely on the TCP handshake. We also use ICMP messages when applicable. By collecting a long enough RTT trace, we can accurately determine if the two VMs are on the same physical machine or not. Further, we can deduce if the VMs are connected to the same TOR (top of the rack) switch. The intuition is that if the two VMs are co-located on the same physical machine, they experience similar processing delays (which depends on the workload of the machine). Indeed, as shown later (Table 5.3), the results of this test are fairly accurate as validated by comparison with our ground truth.

Let $\vec{rtt}(o, d_i) = [rtt(o, d_i, t_1) \ \ldots, \ rtt(o, d_i, t_n)]$, denote the two time series for the attack and victim VM. ($i \in \{1, 2\}$, where, the d_1 is the attack VM and d_2 is the victim VM); o and n represent the observer and duration of the observation, respectively. We use two commonly used metrics to measure the distance between the two time series, viz., the Mean Square Error (MSE) [156] and the Pearson Coefficient (PeC) [38].

The mean square error given by:

$$MSE(o, d_1, d_2) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(rtt(o, d_1, t_i) - rtt(o, d_2, t_i))^2} \tag{5.1}$$

Here, we also measure the MSE relative to a time-shifted versions of the the vector; this is to

account for the fact that in practice, the connections to the two VMs cannot be established simultaneously (one is time shifted slightly with respect to the other).

The PeC between the time series is given by,

$$PeC(o, d_1, d_2) = \frac{COV(\vec{rtt}(o, d_1), \vec{rtt}(o, d_2))}{\sigma(\vec{rtt}(o, d_1)) \times \sigma(\vec{rtt}(o, d_2)).} \tag{5.2}$$

COV is the covariance between the two time series.

In our experiments described in Section 5.4.4, the test declares a success (the two VMs co-reside) if the MSE is < 0.15 and the PeC is ≥ 0.8. Note that ideally, if there is an exact match between the time series, the MSE will be 0 and the PeC will be 1; we choose thresholds that are close to these ideal values.

Timing based ECT with multiple observers: In this test, the attacker uses multiple observers (from different vantage points) and examines the RTTs between the observers and the VMs in question (as in the previous test). The attacker only records the minimum RTT observed between each observer and the target VMs for a specific time span. These RTT values are quantized to the nearest decile (i.e., a value of 46 is rounded to 50, while a value of 41 is rounded to 40). A vector of these quantized RTT values are constructed for both the attack and the victim VM. For example, if there are three observers and ρ_j and ρ'_j are the quantized RTT values with respect to the attack and the victim VMs at the j^{th} observer, the vectors $\{\rho_1, \rho_2, \rho_3\}$ and $\{\rho'_1, \rho'_2, \rho'_3\}$ are constructed. The similarity between these vectors is now computed by means of a Hamming distance type measure. The two VMs are considered to be on the same physical machines, if their these distances coincide (similar to the network triangulation [205]). If the two signatures are very similar (but don't exactly match) it is very likely that that the VMs are on the same physical machine. (in

our experiments described in Section 5.4.4, we assume that they are similar if the fraction of elements that match is ≥ 0.8). The accuracy of the approach depends on the number of observers used and the distribution of observers inside and outside the datacenter. It also depends on the duration of the observations. We call this test the "signature based timing test."

ECT based on RTT timing behaviors from multiple vantage points: The two timing tests described above are combined to improve the accuracy of the co-residency determination. This also decreases the time required to get accurate results. In brief, multiple observers are again used; however, each observer applies the first test on the behaviors of the RTTs (analyzes the similarity between the time series) as opposed to using the minimum RTT a signature (as in the second test). If from most (80 % or higher) vantage points, the behaviors of the time series obtained with respect to the attack and victim VMs are deemed similar, the attacker assumes that its process has successfully co-resided with its victim. We call this test the "hybrid timing test."

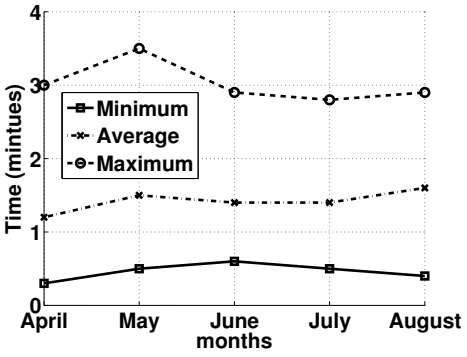

Figure 5.1: Launch times over months

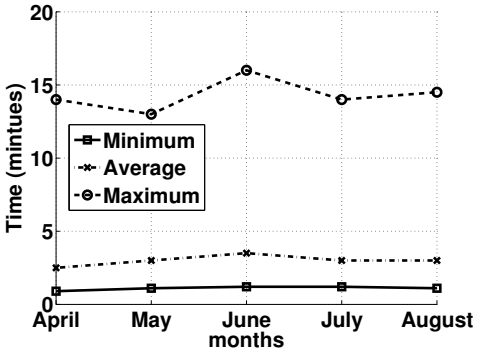

Figure 5.2: Termination times over months

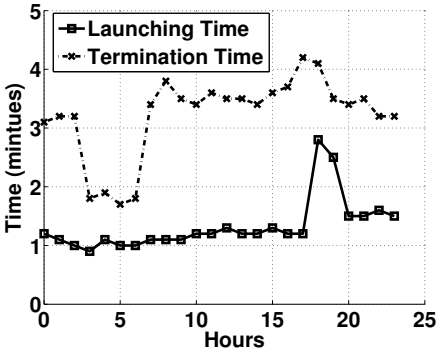

Figure 5.3: Avg. launch and termination times over days

5.4.4 Experimental results

Next, we provide our experimental results on Amazon EC2 with regards to (i) the accuracy and (ii) the time taken for successful co-residency with the different side-channels to verify co-residency.

Launching and Termination times: Launch and termination times are part of the overheads consumed by an attacker while trying to achieve co-residency. They could

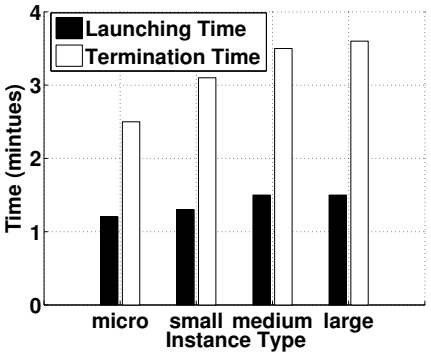

Figure 5.4: Avg. launch and termination times for different VM sizes.

Resource	type	# Avg. VMs	# of hits	Percentage
Disk	ICT	108000	3340	3%
Bus	ICT	108000	22666	21%
CPU	ICT	98000	19540	20%
Core	ICT	98000	8820	9%

Table 5.2: Hit rates with different ICTs

differ from datacenter to datacenter. Note that the attacker may have to try different sizes of attack VMs to achieve co-residency since the cloud providers placement policies may result in the placement of VMs of different sizes on different physical machines (e.g., according to load) [82].

Fig 5.1 shows the minimum, maximum and average launch times over five months. The average launch time is $\approx$ 1.5 minutes. The termination times are longer in general than launch times as shown in Fig 5.2. The average termination time is 2.5 minutes. Upon closer examination, we find that VMs that are on a heavily loaded physical machine (longer response times) take longer to terminate. We also find that larger VMs take longer to terminate (see Fig. 5.4); this is because the deallocation of resources takes longer in such

Application	Sensitivity	Specificity
Taiga	0.94	0.84
MediaServer	0.95	0.88
ownCloud	0.95	0.96

Table 5.3: Sensitivity and specificity with the behavioral test.

Application	Sensitivity	Specificity
Taiga	0.92	0.89
MediaServer	0.93	0.88
ownCloud	0.93	0.95

Table 5.4: Sensitivity and specificity with the signature test.

Application	Sensitivity	Specificity
Taiga	0.95	0.95
MediaServer	0.95	0.95
ownCloud	0.90	0.94

Table 5.5: Sensitivity and specificity with the hybrid test.

Application	Sensitivity	Specificity
Taiga	0.95	0.95
MediaServer	0.95	0.95
ownCloud	0.81	0.95

Table 5.6: Sensitivity and specificity with the Bus based ECT.

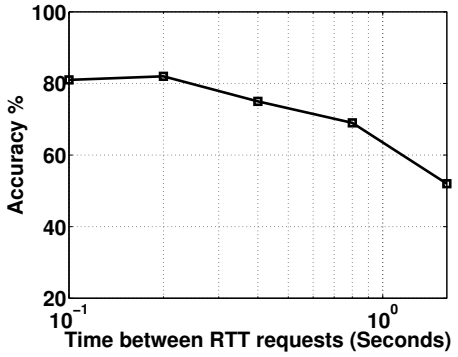

Figure 5.5: Accuracy vs the average time between RTT samples.

cases.

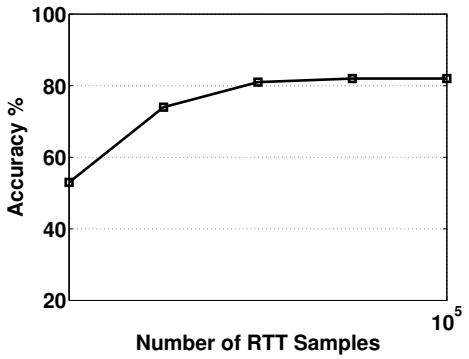

Figure 5.6: The improvement in accuracy by increasing the number of samples.

Hit rate of ICTs: We next quantify the hit rates of the ICT tests on Amazon's EC2. We conducted on average, 3 experiments per hour. As described earlier, in each experiment we launch 20 attack VMs and 20 victim VMs. Table 5.2 shows the hit rates

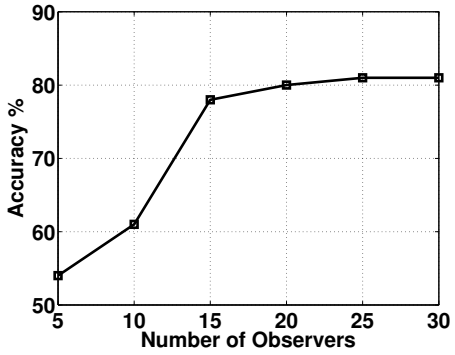

Figure 5.7: The improvement in accuracy as the number of observers increase.

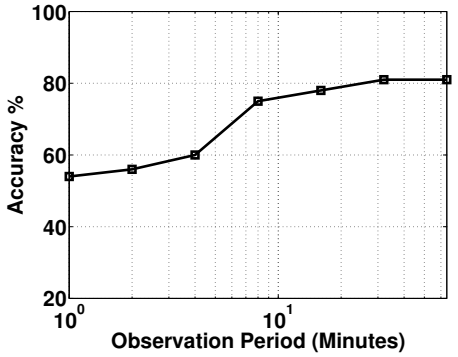

Figure 5.8: The impact of observation period on the accuracy

with the various ICTs discussed earlier. The results show that the disk is shared only 3% of the time. From among this 3%, we find that approximately 86% were sharing the same physical machine (the bus test was also a success). As shown, in approximately 21% of the cases the attack and victim VMs shared the same physical machine. A significant percentage of these also shared the same CPU ($> 90\%$); the LLC cache test was a success. In only 9%

of the cases an attack VM shared a core with a victim VM (L1 cache test succeeded).

Accuracy of the ECTs: Next, we examine the accuracy of the various ECTs described in the previous section. The results are summarized in Tables 5.3-5.6. We report the results with 3 different services. Each service was tested for 5 consecutive days. We approximately conducted 50 runs per day. We re-launch the victim VMs every 10 runs, while the attacker VMs were re-launched every run. We used the bus contention based ICT to establish the ground truth; in other words, that test was also conducted and the result we obtained was considered the truth. This is because, first, the work in [247] shows that this test has very high accuracy with regards to determining the co-residency of two processes. Second, the bus test is the only test for which we have both the ICT and ECT tests. Finally, if the bus test is successful, as discussed above, it is very likely that the LLC cache test will also be successful.

We examine the efficacy of each ECT by measuring the number of true positives (TP), number of true negatives (TN), number of false positives (FP) and number of false negatives (FN). We compute the true positive rate (aka *sensitivity* [97]) to be $\frac{TP}{TP+FN}$. Similarly, the true negative rate (aka *specificity* [97]) is $\frac{TN}{FP+TN}$.

The results show that the bus based ECT and the hybrid timing test exhibit similar (very high) sensitivity and specificity. They are both more accurate than behavioral and signature based timing tests. Note however that designing the requests for the bus contention based ECT is complex. It needs to be tailored to the type of service running on the victim VM. Furthermore, some services do perform heavy memory transfers; if a victim VM hosts such a service, (e.g., ownCloud) it is difficult for an adversary to successfully

carry out a bus based ECT. This demonstrates that for some workloads, the bus contention ECT may not be effective; the hybrid timing tests seem to work well with all the workloads we considered. Finally, we argue that the RTT based methods are less invasive and thus harder to detect.

A microscopic view: In Figs. 5.9 and 5.10, we show snapshots of the normalized time series at a single observer (using the hybrid timing based ECT), when we have a mismatch and a match, respectively. As evident, with a match or hit, the difference in the normalized response times obtained with respect to the victim and the attack VM is much smaller than 0.1 for each sample. With a mismatch, this can be as high as 0.6. In the rare cases with false positives (the ICT yields a mismatch), we find that the normalized RTT is slightly higher (≈ 0.2 for some of the sampled points); this could be a result of the two VMs being close to each other (same rack) but not on the same physical machine.

Properly configuring the ECT tests: The work in [247] discusses how to properly configure the parameters for an effective bus contention ECT (we follow the same approach). We omit the details in the interest of space. Instead we focus on determining how to appropriately configure our new behavioral and signature based timing tests in the next set of experiments. The results reported here are from experiments done over 30 days; we perform 30 runs per day.

Configuring the behavioral timing test: We vary both the total number of samples taken (to construct the time series) and the average time between samples. As seen in Fig. 5.5, an average period of 200 msec yields the highest accuracy from among the values we considered. While this value could change depending on the dynamics inside the cloud (how

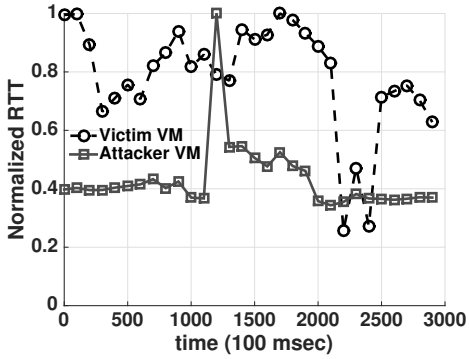

Figure 5.9: Time series snapshot of responses from attack and victim VMs upon a miss

often paths change etc.) we find that roughly choosing this average time between probes is enough; slightly higher or lower sampling rates do not cause significant degradations in performance. Note here that since, the path from the observer to the cloud provider remains fairly stable all the packets experience similar delays on this part of the path; the delays within the cloud are more dynamic. Fig. 5.6 shows the sensitivity of the accuracy to the number of samples taken to form the time series. We observe that going beyond 1000 samples yields little improvement in the accuracy. With these values for the average time between samples and the number of samples, it takes around 200 seconds (3.5 minutes) to perform the behavioral timing test. To decrease the false positives and negatives for the test, the experiments are repeated thrice with at least a 2 - 5 minute pause time, between the tests. Thus the total time taken is between 15 and 26 minutes per attempt.

Configuring the signature based timing test: We vary the number of observers

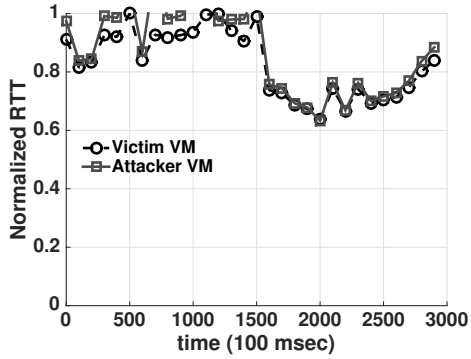

Figure 5.10: Time series snapshot of responses from attack and victim VMs upon a hit

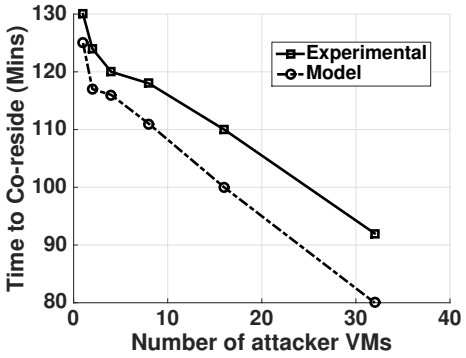

Figure 5.11: Average time to co-reside with any of 8 victim VMs.

considered while creating a signature. As seen in Fig. 5.7, the accuracy does significantly improve if we increase the number of observers above 15. Beyond this, the accuracy is pretty much stable and equal to 81%. To remove fluctuations in RTTs resulting from traffic variations and intra-cloud dynamics (paths on which traffic is routed may change dynamically [113]), we need a larger set of samples in order to obtain accurate signatures.

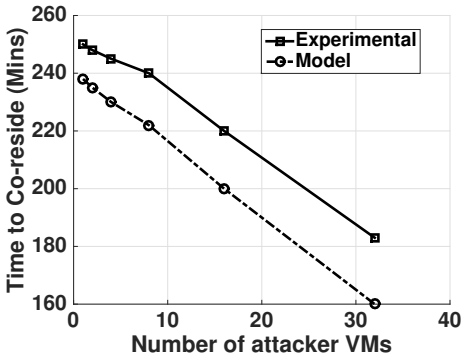

Figure 5.12: Average time to co-reside with any of 4 victim VMs.

We find that typically we need to collect observations over 20 minutes for each run. However, using the hybrid test where we essentially perform a behavioral RTT test at each observer improves the accuracy without the need for a longer observation period; conducting the same behavioral test as before at 16 observers, we find that the accuracy improves to $\approx$ 86 % (the time taken is less than 20 mins).

Experimentally computed times for co-residency: In Table 5.7 we present the experimentally determined, average times with the different ECTs. The times shown include the (a) the launch time, (b) the time taken to determine co-residency (for the tests described) and (c) the termination time. We note that the average times in all cases are about 2 hours. We find that 75 % of the tests took more than 72-93 minutes depending on the ECT in use. More than 95 % took $>$ 20 minutes. The minimum times taken to successfully determine co-residency could be small depending on the test (as seen above). However, the attacker has to really get lucky and must be able to colocate with the targeted victim VM with very few launches of his VMs. We discuss risk assessment and various policies on VM migration (when should a VM be migrated to minimize the risk of long

ECT Test	Average	75%	85%	95%
Behavioral	110	72	45	21
Signature	135	93	53	26
Hybrid	120	78	45	23
Bus	105	76	65	33

Table 5.7: Average and percentiles (mins) of times taken for co-residency.

co-residency with an attacker?) in the Section 5.6.

5.5 Modeling co-residency times

Prior to successfully co-residing with a victim process, an attacker may have to iteratively launch attack processes, check for co-residency (perform the co-residency tests), and upon failure, terminate the processes and relaunch the set of processes. Here, we assume that the attacker has a single account on the cloud, and goes through this procedure until he is able to successfully co-reside with the victim. The time taken for successful co-residency depends on (i) the number of VMs the victim has on the cloud (a web provider may have multiple replicas of his web server running [66]) (ii) the number of attack processes launched in each iteration and (iii) the cloud provider's policy in placing a customer's VMs. In our experiments described earlier, we assumed that the attacker is able to launch 20 VMs in an iteration and the victim has 20 processes running on EC2. Note that Amazon's EC2 limits the number of VMs one can launch with a single account to 20. The provider's policy on VM placement here is unknown.

It is hard to consider all possible cases and perform experiments to characterize the times taken for establishing co-residency. Thus, we seek to develop a simple model that allows us to estimate this time, based on the number of attack and victim VMs; we show that while this model is not very precise, it provides good enough (rough) estimates that can

be used to guide migration decisions. If u is the victim, and the probability of successfully co-residing an attack process with any of the m replica VMs the victim is running, in a given attempt, is $p_c(u)$, the expected time for successful co-residency is given by:

$$E_{CR}[p_c(u)] = (t_l + t_d + t_c) \sum_{j=1}^{J} j(1 - p_c(u))^{j-1} p_c(u) \tag{5.3}$$

where, J is the maximum number of attempts the attacker makes at co-residency. Since we assume a presistent attacker, we set $J = \infty$. With this it is easy to see that:

$$E_{CR}[p_c(u)] = (t_l + t_d + t_c) \frac{1}{p_c(u)} \tag{5.4}$$

Let $p_c(u, m)$ be the probability of successful co-residency with the m^{th} victim VM replica in an attempt. We assume that a victim's VM replica does not share the same physical machine with another replica; conservatively, this maximizes the attacker's chances since he has a better chance of *hitting* a victim VM replica in each attempt. Then, the probability of co-residing an attack process with any of the replica VMs is given by:

$$p_c(u) = \sum_{m} p_c(u, m). \tag{5.5}$$

It is hard to determine $p_c(u)$ without knowing the placement policy of the provider. If one assumes a completely random placement policy (meaning that a physical machine is chosen at random for placement), this probability is $p_c(u, m) = \frac{1}{\mathcal{N}}$, where, $\mathcal{N}$ is the number of available physical machines.

If a plurality of the attacker's VMs are placed on the same physical machine, then he is at an inherent disadvantage (the number of physical machines on which he can check for co-residency in that attempt, is reduced). Thus, we conservatively (to minimize the co-residency time) assume that the attack VMs are always placed on different machines (as

with the victim VMs). This policy is not far from what is likely to happen in reality (e.g., see [82]). For example, a provider may place the VMs of a customer on different physical machines for reliability (robustness to machine failures). If we assume that such a policy is in place, it is easy to see that one can conservatively bound the probability $p_c(u, m)$ by:

$$p_c(u, m) = \frac{1}{\mathcal{N}} + \frac{1}{\mathcal{N}-1} \cdots + \frac{1}{\mathcal{N}-\mathcal{A}} \leq \frac{\mathcal{A}}{\mathcal{N}-\mathcal{A}} \tag{5.6}$$

where, $\mathcal{A}$ is the number of attacker VMs. Thus, if there are $\mathcal{L}$ victim VM replicas, $p_c(u) = \frac{\mathcal{L}\times\mathcal{A}}{\mathcal{N}-\mathcal{A}}$.

Evaluating our model for co-residency time estimation: Next, we compare the co-residency estimates using our model, with that from experiments on EC2. Figs. 5.11 and 5.12 depict the times taken to co-reside with any of the victim VMs, where the victim has deployed 8 and 4 VMs respectively. The number of attacker VMs are varied (for the experiments, we have two accounts and can have up to 40 VMs in total). The value of $\mathcal{N}$ can be estimated based on the number of IP addresses made available on the provider's launch interface and the maximum number of VMs that can be hosted per machine; we use $\mathcal{N} = 500$ since EC2 provides around 4000 available IP addresses and the Xen hypervisor allows 8 VMs per physical machine. The model takes as input, the average (possibly offline) measurements of t_l, t_d and t_c with one attack VM. We see that with different number of attack VMs, we are able to get relative good (but rough) estimates of the co-residency times with the model. This shows that the model can be useful in predicting how long attackers with differing capabilities will take, to successfully co-reside with a victim.

5.6 Determining when to migrate

Next, we seek to develop guidelines on when a VM should be migrated. Towards this, we first propose a set of indicators that capture "the risk of a VM co-residing with an adversarial VM." Note that without knowing the capabilities of an adversary it is hard to quantify risk; thus, the risk indicators are based on what can be directly measured either by the customer (user) or the provider.

5.6.1 Risk indicators

To assess risk, we consider a set of measurable indicators, the variations in which implicitly indicate an increase in risk. These indicators are: **(i)** The time that a victim VM spends on a physical machine relative to the time taken by an adversary to successfully achieve co-residency. As evident, the longer the time spent on the same physical machine, the more probable it is that an adversary has successfully co-resided on the same machine. **(ii)** The level of utilization of the memory bus on the physical host machine. This is the same side channel used by an attacker using the bus contention ECT to ascertain co-residency. From the perspective of the victim, a heavy utilization of the bus can be the result of the bus contention based ECT. It is quite possible that such heavy utilization is because of benign congestion; we argue that even then, migration would help in improving performance. Note that with the timing based ECTs that we discover, the second risk indicator is not useful; in other words, migration has to be based on the time spent by the victim VM on the physical machine.

Time indicator: The first very simple risk indicator is the time for which the VM has resided on the current physical machine and is represented by $\tau = t - t_i$ where, t is the current time, t_i is the time at which the VM was first placed on that physical machine. If one assumes that the timing based ECTs are used, the time indicator is the only metric that can be used to guide migration decisions.

Heavy memory bus utilization indicator: A heavy utilization of the memory bus may indicate that the bus contention based ECT is underway. We sample the utilization of the bus periodically at intervals t_s. If this utilization, on machine m is greater than a threshold for a specific sample (say j), we set a boolean variable associated with that resource $S(m, j)$ to 1 (it is set to 0 otherwise). A composite risk indicator $V(m, K)$ is obtained by jointly considering say K consecutive samples. Specifically,

$$V(m, K) = \sum_{j=k}^{k+K} S(m, j), \tag{5.7}$$

for any k. If this risk indicator yields a value of K, then all K consecutive samples indicated that the bus experienced a high utilization; this would indicate that the VM is at risk of being subjected to a bus contention ECT.

Threshold for determining heavy bus contention: Typically, for specific platforms, there are specifications for the maximum values associated with this heavy utilization indicator. For example, for SDRAM, the specification says that the maximum memory access time is 70 - 150 ns depending on the vendor [131]. One could set the threshold to be a certain fraction of the specified maximum value (e.g., the threshold could be 0.8 of the maximum specified time). Clearly, the higher the threshold, the less likely it is that an alert is issued (leading to a low true positive rate with regards to detecting a threat); on the other

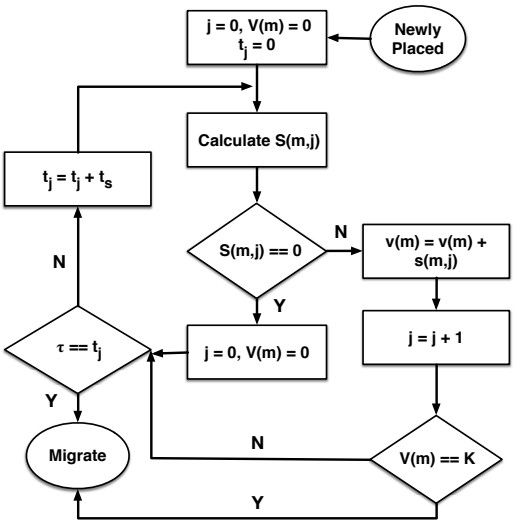

Figure 5.13: Migration Guidelines

hand, setting too low a threshold would incur a higher cost (because of possibly frequent VM migrations). Thus, this threshold could be set based on the user's risk averseness and the costs she is willing to bear. In our work, we conduct an extensive empirical experiments and we set the threshold to a pre-defined value ($Th = 0.8$) so as to keep the false positive rate below 1 %. Similar thresholds are used with the bus contention ECT [247]. We assume that this threshold will be fixed and other parameters are tuned to determine when VMs are to be migrated as discussed next.

5.6.2 Migration guidelines

Next, we try to develop guidelines for migration based on the risk indicators. We assume that the provider does not unilaterally take decisions to migrate a VM (since some users may be unwilling to experience downtimes towards reducing risk). Instead, it monitors the bus utilization at some preset time intervals t_s (not controlled by the user) and based

on user preferences (with regards to certain parameters as discussed later), migrates her VMs.

Our guidelines for performing migrations are characterized using the flow chart in Fig. 5.13. A user's virtual machine enters a safe state when it is placed on a physical machine. The value of $S(m, jt_s)$ on that physical machine m, is checked by the provider at each sampling instance of jt_s (sampling is done every t_s time units i.e., $j = 1, 2, \dots$). If this value is 1, the VM enters the monitor state. If the VM remains in the monitoring state for K consecutive monitoring instances, this implies that $V(m, K) = K$ and and it should be migrated. The machine returns to the safe state if at any point while in the monitoring state, the the value of $S(m, (j + l)t_s)$ (where $l < K$) becomes zero (i.e., the utilization of the bus gets back below the chosen threshold). If the VM remains on the physical machine (regardless of whether how long it spends in the safe or the monitoring state) for τ seconds a decision is made to migrate. We wish to point out here, that there is an implicit assumption that $Kt_s << \tau$.

For ease of discussion, as mentioned earlier, let us assume that the threshold Th is fixed. The two parameters that define the user's cost and risk averseness are K and τ. If the values chosen for these parameters are too small, the number of false positives with respect to detecting a co-residency threat increases; unnecessary high migration costs are experienced. On the other hand, if the values chosen are too high, an attacker can succeed in its attempt to co-reside and do so for long periods.

Costs: In the best case, the user does not migrate for a period of τ seconds. If the size of her VM is X MB, her bandwidth expense in this case will be $\frac{8X}{\tau}$ Mbps. She will

Interval	Bandwidth	Downtime	Efficiency
1 hour	25.6M	1.4s	0.008
2 hour	13.2M	0.8s	0.05

Table 5.8: Average cost and attack efficiency with proactive migration (1 victim VM and 1 attack VMs).

Interval	Bandwidth	Downtime	Efficiency
1 hour	25.6M	1.4s	0.03
2 hour	13.2M	0.8s	0.08

Table 5.9: Average cost and attack efficiency with proactive migration (1 victim VM, 2 attack VMs).

experience a downtime every τ seconds. In the worst case, she will continuously observe bus contention, and will migrate every Kt_s seconds. Here, the bandwidth cost will be $\frac{8X}{Kt_s}$; she will experience a downtime every Kt_s seconds. In practice, if there is no attack or if there is a timing based ECT, the former (best case) will hold true. If there is a bus-contention based ECT, the times between the migration instances will be somewhere in between (and not excluding) the best and the worst case scenarios.

Victim VMs	Bandwidth	Downtime	Attack Efficiency
1	13M	0.75s	0.11
2	14M	0.73s	0.11

Table 5.10: Average cost and attack efficiency for migration based on heavy memory utilization (varying victim VMs).

Victim VMs	Bandwidth	Downtime	Attack Efficiency
1	16M	1.1s	0.03
2	16M	1.2s	0.028

Table 5.11: Average cost and attack efficiency for migration based on both residence time and heavy memory utilization (Attack VMs = 2X Victim VMs).

5.7 Evaluations

In this section, we experimentally evaluate the VM migration in terms of (a) reducing the times for which an attack VM co-resides with a victim and (b) the incurred costs; the migrations are based on the guidelines put together in Section 5.6.2. Unfortunately, Amazon EC2 or other cloud providers do not yet offer a service wherein a user can control (or request) when VM migrations are performed; therefore, our evaluations are on an in house private cloud.

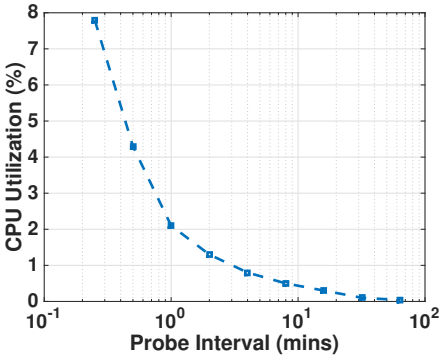

Figure 5.14: The CPU utilization costs with memory probing.

Evaluation Scenarios: We consider the two best ECTs that can be used by the attacker (the hybrid timing based ECT and the bus contention ECT) to ascertain co-

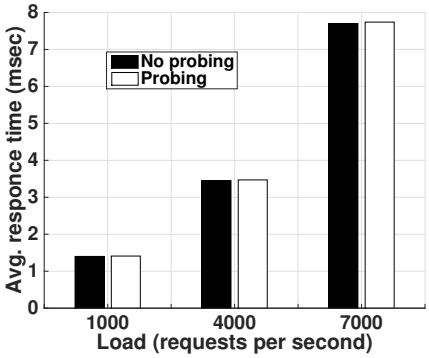

Figure 5.15: The average response times with and without memory probes.

residency. We consider the two risk indicators separately and jointly, to invoke migrations.

Our private cloud testbed: Our private cloud consists of 13 Servers (11 DELL and 2 HP), two Cisco 20-Port gigabit switches and 9 DELL hosts. It can host up to 140 micro VMs or 70 small VMs, simultaneously (equivalent to t2.micro and t2.small on EC2, respectively). We run the KVM hypervisor on top of Ubuntu 14.04. On the VMs, we run Centos 7 and Ubuntu 15 images. We deploy Apache CloudStack [71] to provision the VMs. We perform live migration by using virt-manager (KVM + QEMU). We host Taiga, ownCloud and Mediaserver on the VMs and use 9 hosts to initiate requests to the deployed VMs (background traffic).

Although our testbed is much smaller than commercial clouds, it suffices for a proof-of-concept implementation and showcasing the effectiveness of the VM migration based on our guidelines. On commercial clouds, we believe that VM migration will be even more effective than what we show (because of scale).

Evaluation results: Next, we present our results. Migration costs are averages over 24 hour periods unless specified otherwise. In the first set of results we consider the

reactive attacker model described in Section 5.3; this is what the attacker can do to-day on commercial clouds. Subsequently we consider the cases where it can make choices of whether or not to migrate (static and periodic attackers).

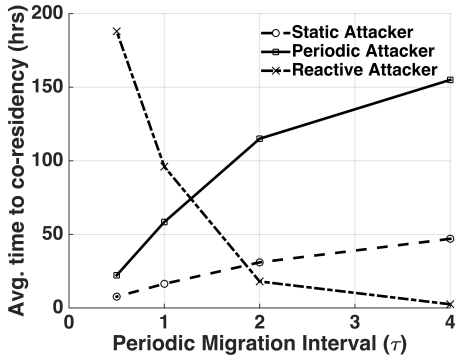

Figure 5.16: Time taken by an attacker to co-reside with a victim VM (inter co-residency time).

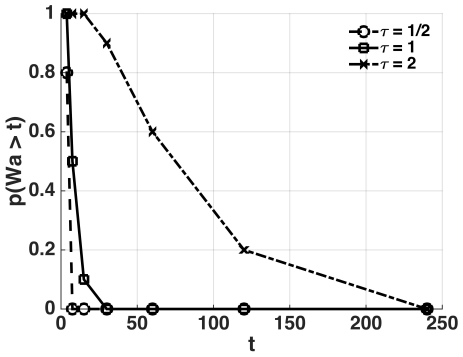

Figure 5.17: Probability that the co-residency time is $> t$ for an reactive attacker; attacker uses hybrid timing based ECT.

Evaluations with a reactive attacker: First, we present our evaluations with

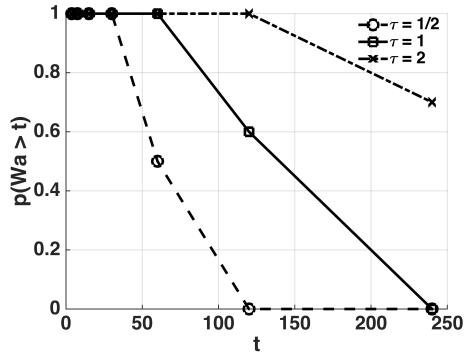

Figure 5.18: Probability that the co-residency time $> t$ for a static attacker; attacker uses hybrid timing based ECT.

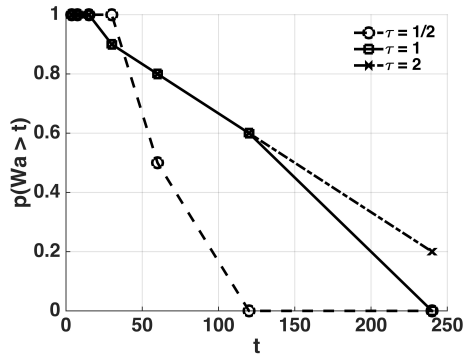

Figure 5.19: Probability that the co-residency time $> t$ for a periodic attacker; attacker uses hybrid timing based ECT.

a reactive attacker.

Migration based on time of residency: First, we consider migration based on only the time of residency (value of τ). Here, we do not trigger alerts from the heavy bus contention utilization indicator. We consider the case where the hybrid timing based ECT strategy is used by an *reactive* attacker[3]. We migrate a VM if the time spent on a physical

[3]Since the bus contention utilization indicator is not used, the results are very similar when the attacker used the bus contention ECT.

machine is equal to $\tau = \beta \times f(T_{CR})$, where f is some monotonically increasing function of the time taken by the attacker to successfully co-reside with the victim. β determines how conservative we are in migrating a VM; a smaller of β invokes more frequent migrations and thus, incurs higher cost. For simplicity, we consider that the function f provides the mean value of T_{CR} (based on the values from from Section 5.4 we set $f(T_{CR}) = 105$). Two values of β, which correspond to inter-migration times τ of approximately 60 ($\beta \approx 0.6$) and 120 mins ($\beta \approx 1.1$), are considered. The lower the β value indicates high user averseness to risk, while the higher the β means a lower risk averseness (and that cost is more important to the owner of the victim VM). The victim VMs are either Taiga, Mediaserver and ownCloud.

We conduct the experiments over a period of 15 days (5 days per type of VM). The cost incurred by the victim is measured in terms of (a) the downtime that it experiences and (b) the bandwidth consumed. The bandwidth consumed corresponds to the memory state (in MB) transferred during the live migration of the victim VM. To capture the security provided, we compute the ratio of the time for which the attack VM co-resides with the victim VM to the total duration of the experiment; we call this the attack efficiency. We have two victim VMs. We populate the machines with 35 additional VMs which are randomly placed, in order to reflect a real operational setting (the cloud has a utilization of approximately 30 %). We consider two and four attacker VMs (i.e., 1X and 2X the number of victim VMs). In this experiment we assume an reactive attacker who performs the hybrid timing based ECT to ascertain co-residency; with this approach, he can re-launch his VMs 1.7 times an hour, on average.

We summarize the costs (downtimes and the traffic generated by migration) and

the attack efficiency in Tables 5.8 and 5.9 with different numbers of attacker VMs. As expected, the traffic volume is doubled if the migration periodicity doubles. The average downtimes are also doubled. However, the attacker efficiency is less than 1% if the VMs are migrated every hour, compared to 5 % if the period is increased to 2 hours. This is because the drop in the attacker success rate is not linear with increased migration frequency (it is better). We see that the costs in terms of downtimes (< 2 s) and bandwidth (of the order of MB over 24 hours) are reasonable.

Migration based on heavy memory utilization: In our next experiments, we assume that the bus contention ECT is used by an reactive attacker. We only migrate a VM if the heavy bus contention risk indicator is triggered. Note here that if the attacker uses the hybrid timing based ECT, the victim's VM will never be migrated in this case. We set the access time threshold to (100ns) (about 0.8 of the maximum specified time on our platforms). The value of K is set to 10. First, we set $\tau = \infty$, i.e., we only use the heavy memory access time risk indicator as a trigger. Table 5.10 summarizes the results. The costs of migration decrease significantly compared to the case where migration is proactively performed based on the time indicator (recall results in Tables 5.8 and 5.9). However, since migration is only performed upon detecting long memory access times, the attacker is able to co-reside with the victim VM for slightly longer periods (in quite a few cases the heavy utilization is not consistently above the chosen threshold); thus, an increase in the attack efficiency is observed.

Jointly considering the time and heavy utilization indicators: Next, we perform proactive migration once every $\tau = 2$ hours (high β); in addition we perform reactive

migration if there is an indication of heavy memory usage i.e., the heavy bus memory bus utilization indicator issues an alert. The reactive attacker employs the bus contention based ECT. Note here that if the attacker were to use the hybrid timing based ECT, the heavy utilization indicator will never kick in and the results will be identical to that of where only the timing based indicator is used to trigger migrations; we have verified that this is the case. The results are in Table 5.11. We see that there are slight increases in the costs in terms of downtimes and bandwidth compared to the case with only proactive migrations with the same β (see Table 5.9, row 2). This is because, additional migrations are now invoked on top of proactive migrations; however, the risk in terms of attack efficiency is reduced by a factor of nearly 3. This suggests that the effectiveness of our migration guidelines; combining the indicators provides better protection with a modest increase in cost (for the same β).

Sampling overheads of memory access utilization: Fig. 5.14 shows the overhead of monitoring memory access times with different sampling rates. This overhead depends on factors such as the type of application, the allocated resources, and the workload. We perform experiments with ownCloud, varying the interval between the memory probes, between 0.25 and 64 minutes. Each probe test lasts for 15 seconds. Approximately 7% of the CPU cycles were consumed even with the smallest probing interval. We perform a similar experiment with Taiga. In Fig. 5.15, we show the average response times to web requests while varying traffic load, with a probing interval of 0.25 minutes. We see that the response times are relatively unaffected. This demonstrates that clients can monitor the risk indicators with relatively very little impact on performance in the cases of the applications

we consider.

Performance with different attacker models: In the next set of experiments, we consider the three different attacker models that we described in Section 5.3 viz., the reactive attacker, the periodic attacker and the static attacker. We recall that an reactive attacker terminates and relaunches its VMs if a co-residency test fails. A periodic attacker simply chooses a period for migration (like a victim VM who migrates based on the timing indicator). A static attacker simply stays put on a single physical machine and awaits the arrival or return of the victim VM (i.e., chooses not to be migrated). As mentioned in Section 5.3, a periodic (static) attacker continuously checks for co-residency since it is unaware of when the victim VM is placed on its physical machine. Implicit in these experiments is the assumption that migrations are allowed on the cloud for all users (the attacker as well as the victim); the users make decisions on whether to migrate or not. We first consider the scenario where the attacker uses the hybrid timing based ECT. Later we consider the bus contention ECT.

The attacker uses the hybrid timing based ECT: In Fig. 5.16, we show the average time taken by an attacker to co-reside with its victim VM for different values of τ; we assume that the periodic attacker migrates its VM at the same rate as the victim. The figure captures how often an attacker co-resides with the victim (but not how long he stays with the victim). We see that frequent migrations cause the victim to come back to the same physical machine occupied by a static or periodic attack VM often. Infrequent migrations would cause the inter-coresidency times to increase. In the case of an reactive attacker, the frequent migrations hurt the time taken to get a co-residency hit (as one would expect).

Reducing the time taken to co-reside with the victim does not necessarily translate to a longer co-residency time. Let us represent the time for which the attack VM co-resides with the victim be represented by W_a. In the next three plots, we present the complementary CDF of W_a, i.e., the probability that $W_a > t$, for the three attacker models, respectively. We immediately see that frequent migrations translate to lower overall co-residency times in all cases. The reactive attacker is hurt the most by frequent migrations. With $\tau = 1/2$ hour, or 1 hour, the migrations occur even before it can successfully carry out a co-residency attempt in many cases. Combined with the fact the fact that its average time to achieve co-residency is high in these regimes (as seen in Fig. 5.16), it is the least effective strategy for the attacker. The static attacker gains time since it does not have to terminate and relaunch his process; we find that if a victim VM, is placed on his machine, the attacker stays with it for the period of the migration (which is to the attacker's advantage). The periodic attacker does better than the reactive attacker; however, it does not do as well as the static attacker since, once if the victim co-resides with it (moves to the physical machine it is occupying), it may be migrated itself. *In summary, the above results suggest that if users are allowed to migrate their VMs, staying put on the same physical machine is the best strategy for an attacker. Performing frequent migrations (to the extent permitted by cost) is the best strategy for the victim.*

The attacker uses the Bus Contention ECT: In Figs. 5.20, 5.21 and 5.22, we show the complementary CDF of W_a when the attacker uses the bus contention ECT. Our migration guidelines are in place. The value of K is 10. We see that regardless of the attacker strategy, a migration is invoked after $Kt_s = 5$ mins with high probability since the high

memory bus utilizaiton indicator is triggered. Thus, the co-residency times are minimal. The value of τ has little impact since migrations are triggered in response to bus contention. The co-residency times are now much smaller from the attacker perspective, compared to the case where it used the bus contention based ECT. *The results not only demonstrate the efficacy of our migration guidelines with regards to minimizing the co-residency periods again, but also demonstrate that the bus contention ECT is much less effective than our hybrid timing based ECT from the perspective of the attacker.*

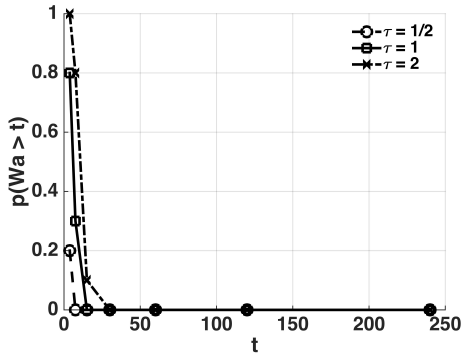

Figure 5.20: Probability that the co-residency time is $> t$ for an reactive attacker; attacker uses bus contention ECT.

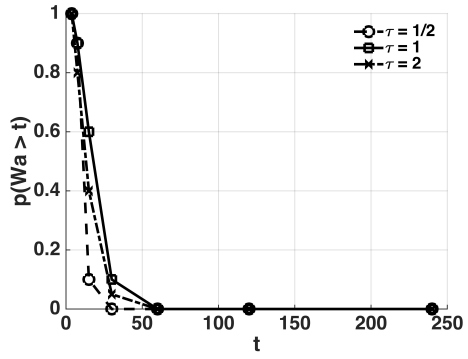

Figure 5.21: Probability that the co-residency time $> t$ for a static attacker; attacker uses bus contention ECT.

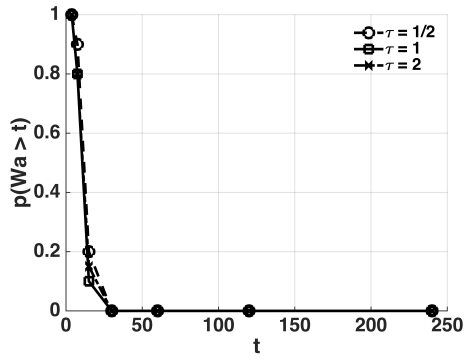

Figure 5.22: Probability that the co-residency time $> t$ for a periodic attacker; attacker uses bus contention ECT.

5.8 Discussion: Implications on side channel attacks targeting information leakage

In our work, the primary metric of interest was the time for which an adversarial VM co-resides with a victim VM (we tried to minimize this time subject to some constraints on performance costs). Reducing the co-residency time directly minimizes the potency of

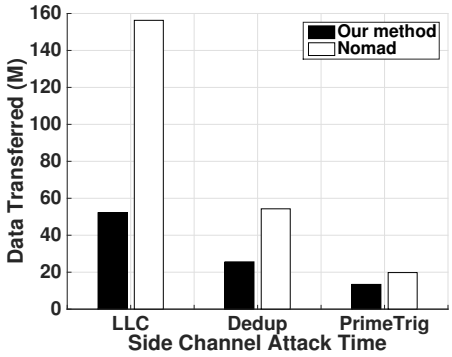

Figure 5.23: A rough comparison of the bandwidth costs with Nomad.

SCA	Avg. Time	Shared Resource
Prime+Trigger+Probe [211]	341	Core
LLC Prime+Probe [164]	27	CPU
Deduplication [114]	45	Storage

Table 5.12: Average times (mins) to carry out side channel attacks.

other side channel attacks that target information leakage; for such attacks to succeed an attacker will need to co-reside with the victim for the duration of the attack. In this section, we provide a discussion on how our work applies if one assumes that such attacks are carried out after co-residency is achieved; note that we only provide rough characterizations of the benefits relating to such attacks here, and a more systematic study is left for the future.

There are numerous side-channel attacks studied in the literature (see Section 5.2). In Table 5.12, we list three representative prior attacks and the average times taken to carry out the attack successfully (to achieve a desired extent of information leakage) as reported in those papers; each side channel attack targets a shared resource and uses a side-channel relating to that resource. The details can be found in the respective citations. The two things we wish to point out are the following. First, these attacks assume that an attack

VM has already co-resided with a victim VM prior to lauching the attack. However, the attacker will first need to co-reside with the victim and this can take significant time, as discussed in Section 5.4. Second, the times taken for these "information leakage" attacks could be shorter than the time taken by an attacker to achieve co-residency on today's cloud platforms (comparing Table 5.7 with Table 5.12). Thus, if one were to use VM migration as a countermeasure against such side-channel attacks which target information leakage, one could potentially use less frequent migrations if one were to account for the time taken to achieve co-residency in addition to launching the information leakage attack (since achieving co-residency is a pre-requisite for the latter attack).

Bandwidth savings by accounting for time taken for co-residency: Nomad [179], implicitly, only considers the time taken for a successful information leakage attack, to invoke migrations. By ignoring the time taken to achieve co-residency, Nomad invokes migrations more frequently than necessary. To compute a rough estimate of the additional costs due to more frequent migrations with Nomad, we consider the three information leakage attacks listed in Table 5.12 and assume that they take the times reported to succeed. Nomad is assumed to invoke a migration just prior to the attack succeeding. With our approach, we migrate proactively with a value of τ equal to the sum of the average time taken for co-residency (105 mins) and the time taken for the information leakage attack as reported in the papers listed in Table 5.12. In Fig. 5.23, with this setup, we compare the bandwidth costs with Nomad with that of our migration approach. The results show that, by not accounting for the time taken for co-residency, Nomad increases the bandwidth costs by 30 - 150% compared to our approach. A more holistic implementation (with both

co-residency and an information leakage attack) to validate the gains in real scenarios, is left for future work.

Decrease in information leakage rates: It is evident that VM migration can decrease information leakage rates (also shown in [179]). It has been shown that without any migration, after co-residing with a victim, an attacker can extract a secret key of length 2048 bits in 27 minutes (corresponding to a leakage rate of 1.26 bps). Our experiments show frequent VM migrations would require an attacker to perform co-residency repeatedly to get this information. Let us assume a strong attacker who can resume his attack once he again co-resides with the victim. With proactive migrations our experiments show that the attacker efficiency is ≈ 1 %. This means that he spends about $24 \times 60 \times 0.01 \approx 14$ minutes per day with the victim; thus, it now takes nearly two days to extract a secret key of the same length (a leakage rate of 5.7×10^{-6} bps).

Note that the effectiveness of proactive migrations (as computed above) are limited by our set up; we have only 13 physical machines. On commercial clouds, where the number of machines could be much higher, the process will be even more effective. In addition, if the attacker cannot immediately resume his attack after he co-resides with a victim VM at a later time, the information leakage rate will be further reduced; in the extreme case if the attacker has to restart his attack, the information cannot be retrieved.

5.9 Conclusions

In this paper, we consider an attacker who seeks to co-reside his VM with a victim VM on the cloud. Achieving such co-residency could allow the attacker to launch

various side-channel attacks that target information leakage. Our goals are to (a) get a comprehensive understanding of the ways and the effectiveness with which an attacker can achieve co-residency and (b) develop migration guidelines for the victim VM that can help minimize its co-residency time with an attacker VM, given constraints on performance costs. Towards achieving (a) we consider both previous side-channel attacks and design our own (more effective) attacks towards ascertaining co-residency with a victim, and evaluate the process of co-residency extensively on Amazon's EC2. Based on these experiments, we formulate a set of migration guidelines and evaluate these extensively with different attacker strategies on our in house cloud. We show that our guidelines can limit the attacker efficiency (fraction of the time it co-resides with the victim) to about 1 % with very modest bandwidth and downtime costs (MB of bandwidth and seconds of downtime per day, per VM migrated).

Bibliography

[1] 10 Ways to Deal With Mobile Data Capacity Crunch. White paper, http://www.amdocs.com/Whitepapers/10-Ways-to-Deal-with-Mobile-Data-Capacity-Crunch.pdf.

[2] Advanced Encryption Standard (AES) (RFC 3826). `http://www.ietf.org/rfc/rfc3826.txt`.

[3] AES using Bouncy Castle API. `www.itcsolutions.eu`.

[4] Click Modular Router. http://read.cs.ucla.edu/click/.

[5] COPE on the Nokia N800 Phones. http://blogs.forum.nokia.com/blog/ frank-fitzeks-forum-nokia-blog/2008/10/06/network-coding.

[6] COPE WiKi. http://piper.csail.mit.edu/dokuwiki/doku.php?id=cope.

[7] Hierarchical Token Bucket. http://luxik.cdi.cz/∼devik/qos/htb/.

[8] HomePlug Support for IEEE Standards. `http://bit.ly/1AThvHO`.

[9] IEEE 802.11n Standard. http://bit.ly/126Nw94.

[10] lpsolve reference guide. http://lpsolve.sourceforge.net/5.5/.

[11] MIT Roofnet. http://pdos.csail.mit.edu/roofnet.

[12] Neco Defense Systems. `http://www.necodefence.com/rfj.php`.

[13] Onoe Rate Control. http://madwifi.org/browser/trunk/ath_rate/onoe.

[14] PhySim-WiFi. http://dsn.tm.kit.edu/english/ns3-physim.php.

[15] PKI 6650 Wideband Jammer. `http://bit.ly/10us5My`.

[16] Raptor Forward Error Correction (RFC 5053). `http://bit.ly/13VJYZq`.

[17] Rice University WARP Project. `http://warp.rice.edu`.

[18] RT2860. http://www.ralinktech.com/ralink/Home/Support/Linux.html.

[19] Rude/Crude measurement tool. http://rude.sourceforge.net/.

[20] SampleRate Bug. http://madwifi.org/ticket/989.

[21] SESP jammers. http://www.sesp.com/.

[22] TCP/UDP Bandwidth Measurement Tool. `http://iperf.fr/`.

[23] The GSM Jammers. `http://bit.ly/TWz15d`.

[24] The MadWiFi driver. http://madwifi.org.

[25] The Network Simulator 2. http://www.isi.edu/nsnam/ns/.

[26] TreeBagger Classification Model. `http://www.mathworks.com/help/stats/treebagger.html`.

[27] Tsung. `http://bit.ly/1KSCxdc//`.

[28] UCR Wireless Testbed. http://networks.cs.ucr.edu/testbed.

[29] Wireless Open-Access Research Platform. http://warp.rice.edu/.

[30] Wireshark. `http://www.wireshark.org`.

[31] Video Traces for Network Performance Evaluation. `http://trace.eas.asu.edu/tracemain.html`, 2008.

[32] 3GPP TS 33.401. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE);. Security architecture. R.12, 2012.

[33] ANSI/IEEE 802.11-Standard. 1999 edition.

[34] Y. Abdallah, M. A. Latif, M. Youssef, A. Sultan, and H. El Gamal. Keys through arq: Theory and practice. *IEEE Trans. Info. For. Sec.*, 2011.

[35] M. Acharya, T. Shamra, D. Thuente, and D. Sizemore. Intelligent Jamming Attacks in 802.11b Wireless Networks. In *OPNETWORK conference*, 2005.

[36] Onur Aciiçmez. Yet another microarchitectural attack:: exploiting i-cache. In *ACM Computer Security Architecture*, 2007.

[37] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In *Cryptographic Hardware and Embedded Systems*. 2010.

[38] Per Ahlgren, Bo Jarneving, and Ronald Rousseau. Requirements for a cocitation similarity measure, with special reference to pearson's correlation coefficient. *American Society for Information Science and Technology*, 54(6), 2003.

[39] R. Ahlswede, N. Cai, S-Y. R.Li, and R. W. Yeung. Network information flow. In *IEEE Trans. Inform. Theory, pp. 1204-1216*, July 2000.

[40] N. Ahmed and S. Keshav. SMARTA: A Self-Managing Architecture for Thin Access Points. In *ACM CONEXT*, 2006.

[41] A. Akella, G. Judd, S. Seshan, and P. Steenkiste. Self-Management in Chaotic Wireless Deployments. In *ACM MOBICOM*, 2005.

[42] Harry R. Anderson. *Fixed Broadband Wireless System Design: The Creation of Global Mobile Communications*. 2003.

[43] ANSI/IEEE802.11-Standard. 1999 edition.

[44] A. O. F. Atya, K. S., S. V. Krishnamurthy, M. A. Khojastepour, and S. Rangarajan. Voltnet: Effective power line networking in multi-flow environments (technical report). In *http://bit.ly/1wMMoam*, 2014.

[45] Ahmed Osama Fathy Atya, Azeem Aqil, Shailendra Singh, Ioannis Broustis, Karthikeyan Sundaresan, and Srikanth V Krishnamurthy. Mitigating malicious interference via subcarrier-level radio agility in wireless networks. In *ICNP*, 2013.

[46] Amittai Aviram, Sen Hu, Bryan Ford, and Ramakrishna Gummadi. Determinating timing channels in compute clouds. In *ACM Cloud Computing Security Workshop*, 2010.

[47] M. Baldi, M. Bianchi, N. Maturo, and F. Chiaraluce. A physical layer secured key distribution technique for ieee 802.11g wireless networks. *Wireless Communications Letters, IEEE*, 2013.

[48] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. On the Performance of IEEE 802.11 under Jamming. In *IEEE INFOCOM*, 2008.

[49] P. Bellavista, A. Corradi, and L. Foschini. The MUM Middleware to Counteract IEEE 802.11 Performance Anomaly in Context-aware Multimedia Provisioning. In *International Journal of Multimedia and Ubiquitous Engineering, Vol. 2, No. 2*, July 2007.

[50] J. Bicket. Bit-rate Selection in Wireless Networks. In *MS Thesis, Dept. of Electr. Engin. and Comp. Science, MIT*, 2005.

[51] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: The insecurity of 802.11. In *ACM MobiCom '01*.

[52] T. Bostoen and O. Van de Wiel. Modelling the low-voltage power distribution network in the frequency band from 0.5 MHz to 30 MHz for broadband powerline communications (plc). In *2000 international Zurich seminar on broadband communications*, pages 171–178, 2000.

[53] D. Boswarthick, O. Hersent, and O. Elloumi. M2M Communications: A Systems Approach. In *Wiley, ISBN: 978-1-1199-9475-6, 1st edition*, 2012.

[54] P. Bremaud. Markov Chains, Gibbs Field, Monte Carlo Simulation and Queues. Springer-Verlag 1999.

[55] I. Broustis, J. Eriksson, S. V. Krishnamurthy, and M. Faloutsos. A Blueprint for a Manageable and Affordable Wireless Testbed: Design, Pitfalls and Lessons Learned. In *IEEE TRIDENTCOM*, 2007.

[56] I. Broustis, J. Eriksson, S. V Krishnamurthy, and M. Faloutsos. Implications of Power Control in Wireless Networks: A Quantitative Study. In *PAM*, 2007.

[57] I. Broustis, K. Papagiannaki, S. V. Krishnamurthy, M. Faloutsos, and V. Mhatre. MDG: Measurement-Driven Guidelines for 802.11 WLAN Design. In *ACM MOBICOM*, 2007.

[58] I. Broustis, G. Paschos, D. Syrivelis, L. Georgiadis, and L. Tassiulas. NCRAWL: Network Coding for Rate Adaptive Wireless Links. In *Technical Report, arXiv:1104.0645v1*, 2011.

[59] I. Broustis, K. Pelechrinis, D. Syrivelis, S. V. Krishnamurthy, and L. Tassiulas. Quantifying the Overhead due to Routing Probes in Multi-Rate WMNs. In *IEEE WCNC*, 2010.

[60] T. X. Brown, J. E. James, and A. Sethi. Jamming and Sensing of Encrypted Wireless Ad Hoc Networks. In *ACM MOBIHOC*, 2006.

[61] A. Brusilovsky, I. Faynberg, and Z. Zeltsan. Password-Authenticated Key (PAK) Diffie-Hellman Exchange. RFC 5683, 2010.

[62] V. Cakulev, G. Sundaram, and I. Broustis. IBAKE: Identity-Based Authenticated Key Exchange. RFC 6539, 2012.

[63] Joseph Camp and Edward Knightly. Modulation rate adaptation in urban and vehicular environments: Cross-layer implementation and experimental evaluation. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, 2008.

[64] Haowen Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE S & P*, 2003.

[65] P. Chaporkar and A. Proutiere. Adaptive Network Coding and Scheduling for Maximizing Throughput in Wireless Networks. In *MOBICOM*, 2007.

[66] Fangfei Chen, Katherine Guo, John Lin, and Thomas La Porta. Intra-cloud lightning: Building cdns in the cloud. In *IEEE INFOCOM*, 2012.

[67] R.-T. Chinta, T.F. Wong, and J.M. Shea. Energy-Efficient Jamming Attack in IEEE 802.11 MAC. In *IEEE MILCOM*, 2009.

[68] S. Choi, K. Park, and C. Kim. On the Performance Characteristics of WLANs: Revisited. In *ACM SIGMETRICS*, 2005.

[69] T Charles Clancy. Efficient ofdm denial: Pilot jamming and pilot nulling. In *IEEE ICC*, 2011.

[70] D. Clark. Powerline communications: Finally ready for prime time? In *IEEE Internet Computing*, 1998.

[71] Apache CloudStack. Open Source Cloud Computing. `https://goo.gl/TT2S3R`, 2016.

[72] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High Throughput Path Metric for MultiHop Wireless Routing. In *ACM MOBICOM*, 2003.

[73] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *NDSS*, 2015.

[74] E. Dalhman, S. Parkvall, and Johan Skold. 4G: LTE/LTE-Advanced for Mobile Broadband. In *Elsevier*, 2011.

[75] B. DeCleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, and C. Zhang. Secure group communications for wireless networks. In *IEEE MILCOM 2001.*

[76] T. Denoeux. A k-nearest neighbor classification rule based on dempster-shafer theory. *Systems, Man and Cybernetics, IEEE Transactions on*, 25(5):804–813, May 1995.

[77] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.2. RFC 5246,2008.

[78] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory,*, 1976.

[79] Q. Dong, J. Wu, W. Hu, and J. Crowcroft. Practical network coding in wireless networks. In *ACM MOBICOM*, 2007.

[80] R. Draves, J. Padhye, and B. Zill. Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks. In *ACM MOBICOM*, 2004.

[81] J. Dunn, M. Neufeld, A. Sheth, D. Grunwald, and J. Bennett. A Practical Cross-Layer Mechanism For Fairness in 802.11 Networks. In *IEEE BROADNETS*, 2004.

[82] Amazon EC2. AWS Security White paper. `http://goo.gl/GK0sz8`, 2011.

[83] Amazon EC2. T2 Instance Requirements. `http://goo.gl/GWMxxI`, 2016.

[84] A. Eryilmaz and D. D. Lun. Control for inter-session network coding. In *in Proc. of Information Theory and Applications Workshop (ITA2007)*, 2007.

[85] D. B. Johnson et al. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, 2001.

[86] D. Raychaudhuri et al. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *IEEE WCNC*, 2005.

[87] D. S. Lun et al. Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding. In *IEEE INFOCOM*, 2005.

[88] D. S. Lun et al. Minimum-Cost Multicast over Coded Packet Networks. In *IEEE Trans. Inform. Theory, 52(6):931-938*, 2006.

[89] E. Rozner et al. ER: Efficient Retransmission Scheme for Wireless LANs. In *ACM CONEXT*, 2007.

[90] J. Bicket et al. Architecture and Evaluation of an Unplanned 802.11b Mesh Network. In *ACM MOBICOM*, 2005.

[91] J. Camp et al. A Measurement Study of Multiplicative Overhead Effects in Wireless Networks. In *IEEE INFOCOM*, 2008.

[92] J. Kim et al. A Memory Copy Reduction Scheme for Networked Multimedia Service in Linux Kernel. In *EurAsia-ICT, LNCS 2510, pp. 188195*, 2002.

[93] S. Chachulski et al. Trading Structure for Randomness in Wireless Opportunistic Routing. In *ACM SIGCOMM*, 2007.

[94] S. Deb et al. Network Coding for Wireless Applications : A Brief Tutorial. In *IWWAN*, 2005.

[95] Y. Xi et al. Adaptive Multirate Auto Rate Fallback Protocol for IEEE 802.11 WLANs. In *IEEE MILCOM*, 2006.

[96] J. Le *et al.* How Many Packets Can we Encode? - An Analysis of Practical Wireless Network Coding. In *IEEE INFOCOM*, 2008.

[97] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8), 2006.

[98] C. Fragouli, D. Katabi, A. Markopoulou, M. Medard, and H. Rahul. Wireless network coding: Opportunities & Challenges. In *IEEE Military Communications Conference 2007*, October 2007.

[99] C. Fragouli, J. Widmer, and J. Y. Le Boudec. Network Coding: an Instant Primer. In *ACM SIGCOMM CCR, vol. 36, pp. 63-68*, Jan. 2006.

[100] J. Fuemmeler, N. Vaidya, and V. V. Veeravalli. Selecting Transmit Powers and Carrier Sense Thresholds for CSMA Protocols. Technical report, October 2004. University of Illinois at Urbana-Champaign.

[101] C. L Fullmer and JJ Garcia-Luna-Aceves. *Solutions to Hidden Terminal Problems in Wireless Networks*, volume 27. ACM, 1997.

[102] S. Galli. A simplified model for the indoor powerline channel. In *Power Line Communications and Its Applications, 2009. ISPLC 2009. IEEE International Symposium on*, pages 13–19, March 2009.

[103] J. Geier. Assigning 802.11b Access Point Channels. In *WiFi planet*, 2002.

[104] E. Gelal, G. Jakllari, and S. V. Krishnamurthy. Exploiting Diversity Gain in MIMO Equipped Ad hoc Networks. In *Asilomar Conf. on Signals and Systems*, 2006.

[105] L. Georgiadis, M. Neely, and L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1:1–147, 2006.

[106] L. Georgiadis and L. Tassiulas. Broadcast erasure channel with feedback - capacity and algorithms . In *Network Coding Workshop 2009*, June 2009.

[107] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *IEEE INFOCOM*, 2005.

[108] M. Gotz, M. Rapp, and K. Dostert. Powerline channel characteristics and their effect on communication system design. *Communications Magazine, IEEE*, 42(4):78–86, Apr 2004.

[109] R. Gummadi, D. Wetheral, B. Greenstein, and S. Seshan. Understanding and Mitigating the Impact of RF Interference on 802.11 Networks. In *ACM SIGCOMM*, 2007.

[110] R. Gummadi, D. Wetheral, B. Greenstein, and S. Seshan. Understanding and Mitigating the Impact of RF Interference on 802.11 Networks. In *ACM SIGCOMM*, 2007.

[111] M. Halldorsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. In *Algorithmica*, 1997.

[112] M. Hansen, R. Dubayah, and R. Defries. Classification trees: An alternative to traditional land cover classifiers. *International Journal of Remote Sensing*, 17(5):1075–1081, 1996.

[113] Fang Hao, TV Lakshman, Sarit Mukherjee, and Haoyu Song. Enhancing dynamic cloud-based services using network virtualization. In *ACM Virtualized Infrastructure Systems and Architectures*. ACM, 2009.

[114] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE S&P*, 8(6), 2010.

[115] L. Harte. Introduction To EVDO: Physical Channels, Logical Channels, Network and Operation. In *Althos*, J uly 2004.

[116] H. Hartley. The modified gauss-newton method for the fitting of non-linear regression functions by least squares. *Technometrics*, 1961.

[117] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[118] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley., 1991.

[119] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance Anomaly of 802.11b. In *IEEE INFOCOM*, 2003.

[120] T. Ho, Y. Chang, and K. J. Han. On constructive network coding for multiple unicasts. In *in Proc. of 44th Allerton conference on Communication, Control and Computing*, September 2006.

[121] T. Ho and R. Koetter. Online Incremental Network Coding for Multiple Unicasts. In *DIMACS Working Group on Network Coding*, 2005.

[122] T. Ho, M. Medard, M. Effros, and D. Karger. On Randomized Network Coding. In *Allerton Conf. on Commun., Control and Computing*, pages 11–20, 2003.

[123] H. Holma and A. Toskala. HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications. In *Wiley*, 2006.

[124] G. Horvat, Z. Balkić, and D. Žagar. Power line communication throughput analysis for use in last mile rural broadband. In *20th Telecommunications Forum TELFOR 2012*, 2012.

[125] W. Hu, T. Wood, W. Trappe, and Y. Zhang. Channel Surfing and Spatial Retreats: Defenses Against Wireless Denial of Service. In *ACM Workshop on Wireless Security*, 2004.

[126] W. Hu, T. Wood, W. Trappe, and Y. Zhang. Surfing and Spatial Retreats: Defenses Against Wireless Denial of Service. In *ACM Workshop on Wireless Security*, 2004.

[127] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, 2004.

[128] Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space aslr. In *IEEE S&P*, 2013.

[129] L. Iannone and S. Fdida. Sdt. 11b: Un Schema a Division de Temps Pour Eviter l'anomalie de la Couche MAC 802.11b. In *CFIP*, April 2005.

[130] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Lucky 13 strikes back. In *ACM Information, Computer and Communications Security*, 2015.

[131] Bruce Jacob, Spencer Ng, and David Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[132] A. Jacobs. The pathologies of big data. *ACM Commun.*, 52(8), August 2009.

[133] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, MobiCom '03, pages 66–80, New York, NY, USA, 2003. ACM.

[134] A. Jardosh, K. Mittal, K. N. Ramachandran, E. M. Belding, and K. C. Almeroth. IQU: Practical Queue-Based User Association. In *ACM MOBICOM*, 2006.

[135] I Javed, A Loan, and W Mahmood. An energy-efficient anti-jam cognitive system for wireless ofdm communication. *International Journal of Communication Systems*, 2013.

[136] Frank Kargl, Joern Maier, and Michael Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*. ACM, 2001.

[137] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-Level Network Coding for Wireless Mesh Networks. In *ACM SIGCOMM*, 2008.

[138] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in The Air: Practical Wireless Network Coding. In *ACM SIGCOMM*, 2006.

[139] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in The Air: Practical Wireless Network Coding. In *IEEE/ACM Trans. on Networking, Vol. 16, Iss. 3*, June 2008.

[140] B. Kauffmann, F. Baccelli, A. Chainteau, V. Mhatre, K. Papagiannaki, and C. Diot. Measurement-Based Self Organization of Interfering 802.11 Wireless Access Networks. In *IEEE INFOCOM*, 2007.

[141] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, ISBN 0-13-345711-7.

[142] F Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 1998.

[143] R. Khalili, M. Ghaderi, J. Kurose, and D. Towsley. On the performance of random linear network coding in relay networks. In *IEEE MILCOM*, 2008.

[144] R. Khalili and K. Salamatian. On the achievability of cut-set bound for a class of erasure relay channels. In *IWWAN*, 2004.

[145] R. Khalili and K. Salamatian. A new relaying scheme for cheap wireless relay nodes. In *WIOPT 2005.*, 2005.

[146] Abdallah Khreishah, Issa M Khalil, and Jie Wu. Polynomial time and provably efficient network coding scheme for lossy wireless networks. In *IEEE MASS*, 2011.

[147] H. Kim, S. Yun, I. Kang, and S. Bahk. Resolving 802.11 Performance Anomalies through QoS Differentiation. In *IEEE Comm. Letters, Vol. 9, No. 7*, July 2005.

[148] T. S. Kim, S. Vural, I. Broustis, D. Syrivelis, S. V. Krishnamurthy, and T. La Porta. A Framework for Joint Network Coding and Transmission Rate Control in Wireless Networks. In *IEEE INFOCOM*, 2010.

[149] Tae-Suk Kim, Hyuk Lim, and Jennifer C. Hou. Improving Spatial Reuse Through Tuning Transmit Power, Carrier Sense Threshold, and Data Rate in Multihop Wireless Networks. In *ACM MobiCom*, 2006.

[150] R. Koetter and M. Medard. An Algebraic Approach to Network Coding. In *IEEE/ACM Trans. on Networking*, 2003.

[151] M. Korki, N. Hosseinzadeh, H.L. Vu, T. Moazzeni, and Chuan Heng Foh. A channel model for powerline communication in the smart grid. In *IEEE Power Systems Conference and Exposition*, 2011.

[152] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press, 1995.

[153] R. Kumar, S. Tati, F. De Mello, S. V. Krishnamurthy, and T. La Porta. Network Coding Aware Rate Selection in Multi-Rate IEEE 802.11. In *IEEE ICNP*, 2010.

[154] P. Kyasanur and N. Vaidya. Selfish MAC Layer Misbehavior in Wireless Networks. In *IEEE Trans. on Mob. Computing, Vol. 4, No. 5*, September 2005.

[155] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *USITS*, volume 1, pages 11–11, 2001.

[156] N Levinson. The wiener RMS (root mean square) error criterion in filter design and prediction. 1947.

[157] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal Jamming Attacks and Network Defense Policies in Wireless Sensor Networks. In *IEEE INFOCOM*, 2007.

[158] Peng Li, Debin Gao, and Michael K Reiter. Stopwatch: a cloud architecture for timing channel mitigation. *ACM Information and System Security (TISSEC)*, 17(2), 2014.

[159] S. R. Li, R. W. Yeung, and N. Cai. Linear network coding. In *IEEE Trans. Inform. Theory*, 2003.

[160] Z. Li and B. Li. Network coding: The Case for Multiple Unicast Sessions. In *Allerton Conference on Communications*, pages 1–9, 2004.

[161] Ari Liberman García. *The evolution of the Cloud: the work, progress and outlook of cloud infrastructure.* PhD thesis, Massachusetts Institute of Technology, 2015.

[162] G. Lin and G. Noubir. On Link Layer Denial of Service in Data Wireless LANs. In *Wireless Communications and Mobile Computing*, May 2003.

[163] C. H. Liu and F. Xue. Network Coding for Two-Way Relaying: Rate Region, Sum Rate and Opportunistic Scheduling. In *IEEE ICC*, 2008.

[164] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE S&P*, 2015.

[165] Y. Liu, P. Ning, H. Dai, and A. Liu. Randomized Differential DSSS: Jamming-Resistant Wireless Broadcast Communication. In *IEEE INFOCOM*, 2010.

[166] R. Liyong, C. Bo, and W. Jing. A novel packet-pair-based inferring bandwidth congestion control mechanism for layered multicast. *SIGOPS*, 37(4), Oct 2003.

[167] M. Luby. Lt codes. In *IEEE FOCS*, 2002.

[168] D.S. Lun, M. Medard, R. Koetter, and M. Effros. Further results on coding for reliable communication over packet networks. In *ISIT*, 2005.

[169] Jun Luo, J.H. Andrian, and Chi Zhou. Bit error rate analysis of jamming for ofdm systems. In *Wireless Telecommunications Symposium*, April 2007.

[170] M. Lzaro-gredilla and M. K. Titsias. Variational heteroscedastic gaussian process regression. In *In 28th International Conference on Machine Learning (ICML-11*, pages 841–848. ACM, 2011.

[171] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The Next Frontier For Innovation, Competition, and Productivity, May 2011.

[172] L. Massoulié and J. Roberts. Bandwidth Sharing: Objectives and Algorithms. *IEEE/ACM Trans. Netw.*, 10(3):320–328, 2002.

[173] V. Mhatre and K. Papagiannaki. Optimal Design of High-Density 802.11 WLANs. In *ACM CONEXT*, 2006.

[174] V. Mhatre, K. Papagiannaki, and F. Baccelli. Interference Mitigation through Power Control in High Density 802.11 WLANs. In *IEEE INFOCOM*, 2007.

[175] M. Minsky. Steps toward artificial intelligence. In *Computers and Thought*, pages 406–450. McGraw-Hill, 1961.

[176] A. Mishra, V. Brik, S. Banerjee, A. Srinivasan, and W. Arbaugh. A Client-Driven Approach for Channel Management in Wireless LANs. In *IEEE INFOCOM*, 2006.

[177] A. Mishra, V. Shrivastava, D. Agarwal, and S. Banerjee. Distributed Channel Management in Uncoordinated Wireless Environments. In *ACM MOBICOM*, 2006.

[178] T. Mladenov, S. Nooshabadi, and K. Kim. Implementation and evaluation of raptor codes on embedded systems. *IEEE Transactions on Computers*, 2011.

[179] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *ACM CCS*, 2015.

[180] R. Morris, E. Kohler, J. Janotti, and M. F. Kaashoek. The Click Modular Router. In *ACM Symposium on Operating Systems Principles*, 1999.

[181] T. B Murdoch and A. S Detsky. The inevitable application of big data to health care. *JAMA*, 309(13):1351–1352, 2013.

[182] R. Murty, J. Padhye, R. Chandra, A. R. Chowdhury, and M. Welsh. Characterizing the end-to-end performance of indoor powerline networks. *Harvard University Microsoft Research*, 2008.

[183] V. Navda, A. Bohra, S. Ganguly, and D. Rubenstein. Using Channel Hopping to Increase 802.11 Resilience to Jamming Attacks. In *IEEE INFOCOM mini-conference*, 2007.

[184] G. Noubir. On Connectivity in Ad Hoc Network under Jamming Using Directional Antennas and Mobility. In *Wired/Wireless Internet Communications, Vol. 2957/2004, pp. 186-200, 2004.*

[185] G. Noubir and G. Lin. Low-power DoS Attacks in Data Wireless LANs and Countermeasures. In *ACM MOBIHOC (poster)*, 2003.

[186] G. Noubir and G. Lin. Low-power DoS Attacks in Data Wireless LANs and Countermeasures. In *ACM MOBIHOC*, 2003.

[187] G. Noubir, R. Rajaraman, B. Sheng, and B. Thapa. On the Robustness of IEEE 802.11 Rate Adaptation Algorithms Against Smart Jamming. In *ACM WISEC*, 2011.

[188] M. Obitko, V. Jirkovsky, and J. Bezdicek. Big data challenges in industrial automation. In *Industrial Applications of Holonic and Multi-Agent Systems*, 2013.

[189] C. Orakcal and D. Starobinski. Rate Adaptation in Unlicensed Bands under Smart Jamming Attacks. In *CrownCom*, 2012.

[190] S. Pal, S. R. Kundu, K. Basu, and S. K. Das. IEEE 802.11 Rate Control Algorithms: Experimentation and Performance Evaluation in Infrastructure Mode. In *PAM*, 2006.

[191] P. Palanisamy and T.V.S. Sreedhar. Performance analysis of raptor codes in ofdm systems. In *IEEE ICETET*, 2008.

[192] C. Papadimtriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[193] Jeongho Park, Dongkyu kim, Changeon Kang, and Daesik Hong. Effect of partial band jamming on ofdm-based wlan in 802.11g. In *ICASSP '03)*, volume 4, April 2003.

[194] K. Pelechrinis, I. Broustis, S. V. Krishnamurthy, and C. Gkantsidis. ARES: An Antijamming REinforcement System for 802.11 Networks. In *ACM CoNEXT*, 2009.

[195] K. Pelechrinis, I. Broustis, S. V. Krishnamurthy, and C. Gkantsidis. A Measurement-Driven Anti-Jamming System for 802.11 Networks. *IEEE/ACM ToN*, 2011.

[196] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy. Denial of Service Attacks in Wireless Networks: The case of Jammers. In *IEEE Comm. Surveys and Tutorials*, 2011.

[197] K. Pelechrinis, C. Koufogiannakis, and S. V. Krishnamurthy. On the Efficacy of Frequency Hopping in Coping with Jamming Attacks in 802.11 Networks. *IEEE Trans. Wireless Commun.*, 2010.

[198] Colin Percival. Cache missing for fun and profit, 2005.

[199] Cell phone Jammer. Wikipedia. http://en.wikipedia.org/wiki/Cell_ phone_ jammer.

[200] M. Portoles, Z. Zhong, and S. Choi. IEEE 802.11 Downlink Traffic Shaping Scheme for Multi-User Service. In *IEEE PIMRC*, 2003.

[201] A. Proao and L. Lazos. Selective Jamming Attacks in Wireless Networks. In *IEEE ICC*, 2010.

[202] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *ACM Cloud Computing Security Workshop*, 2009.

[203] A. Ramamoorthy, J. Shi, and R. Wesel. On the Capacity of Network Coding for Wireless Networks. In *Allerton Conf. Commun., Control and Computing*, 2003.

[204] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: closing digital side-channels through obfuscated execution. In *USENIX Security*, 2015.

[205] Bharat Rao and Louis Minakakis. Evolution of mobile location-based services. *ACM Communications*, 2003.

[206] S. Rayanchu, S. Sen, J. Wu, S. Banerjee, and S. Sengupta. Loss-Aware Network Coding for Unicast Wireless Sessions: Design, Implementation, and Performance Evaluation. In *ACM SIGMETRICS*, 2008.

[207] Shravan Rayanchu, Sayandeep Sen, Jianming Wu, Suman Banerjee, and Sudipta Sengupta. Loss-aware network coding for unicast wireless sessions: design, implementation, and performance evaluation. In *ACM SIGMETRICS Performance Evaluation Review*, 2008.

[208] T. Razafindralambo, I. G. Lassous, L. Iannone, and S. Fdida. Dynamic Packet Aggregation to Solve Performance Anomaly in 802.11 Wireless Networks. In *ACM MSWiM*, October 2006.

[209] Chris Regan. Bagging and selection. 2002.

[210] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Measurement-based models of delivery and interference in static wireless networks. In *ACM SIGCOMM*, 2006.

[211] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM CCS*, 2009.

[212] M. Roger. IEEE 802.16 WirelessMAN Standard: Myths and Facts. In *Wireless Communications Conference.*, 2006.

[213] S. Sancha, F.J. Canete, L. Diez, and J.T. Entrambasaguas. A channel simulator for indoor power-line communications. In *ISPLC.*, 2007.

[214] A Sang, X. Wang, M. Madihian, and R.D. Gitlin. A flexible downlink scheduling scheme in cellular packet data systems. *Wireless Communications, IEEE Transactions on*, 5(3), March 2006.

[215] H. Sari, Y. Levy, and G. Karam. An Analysis of Orthogonal Frequency-Division Multiple Access. In *IEEE GLOBECOM '97.*

[216] T. Sartenaer and P. Delogne. Powerline cables modelling for broadband communications. In *Proc. IEEE Int. Conf. Power Line Communications and Its Applications*, pages 331–337, 2001.

[217] B. Scheuermann, W. Hu, and J. Crowcroft. Near-Optimal Co-ordinated Coding in Wireless Multihop Networks. In *ACM CONEXT*, 2007.

[218] H. Seferoglu and A. Markopoulou. Opportunistic Network Coding for Video Streaming over Wireless. In *Packet Video*, 2007.

[219] H. Seferoglu and A. Markopoulou. Distributed Rate Control for Video Streaming over Wireless Networks with Intersession Network Coding. In *Packet Video*, 2009.

[220] H. Seferoglu, A. Markopoulou, and U. Kozat. Network coding-aware rate control and scheduling in wireless networks. In *ICME*, 2009.

[221] Hulya Seferoglu, Athina Markopoulou, and KK Ramakrishnan. I 2 nc: intra-and inter-session network coding for unicast flows in wireless networks. In *IEEE INFOCOM*, 2011.

[222] Sambu Seo, Marko Wallat, Thore Graepel, and Klaus Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Mustererkennung 2000*, pages 27–34. Springer, 2000.

[223] S. Sesia, I. Toufik, and M. Baker. LTE - The UMTS Long Term Evolution: From Theory to Practice. In *Wiley; 2nd Ed.*, Aug. 2011.

[224] A. Sezgin. Capacity Achieving High Rate Space-Time Block Codes. In *IEEE Comm. Letters, Vol. 9, No. 5*, May 2005.

[225] C. Shahriar, M. La Pan, M. Lichtman, T.C. Clancy, R. McGwier, R. Tandon, S. Sodagari, and J.H. Reed. Phy-layer resiliency in ofdm communications: A tutorial. *IEEE Communications Surveys Tutorials*, PP(99), 2014.

[226] S. Sharma, Yi Shi, Jia Liu, Y.T. Hou, and S. Kompella. Is network coding always good for cooperative communications? In *IEEE INFOCOM*, 2010.

[227] A. Shokrollahi. Raptor codes. *IEEE Trans. Inf. Theory*, 2006.

[228] D. Slater, Patrick T., Radha P., and Brian J. M. A coding-theoretic approach for efficient message verification over insecure channels. In *ACM WiSec*, 2009.

[229] D. Smetters, D. Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. 2002.

[230] Soekris-net4826. http://www.soekris.com/net4826.htm.

[231] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security*, 2001.

[232] F. Song and Y. Li. Cross-layer optimization for ofdm wireless networks-part i: theoretical framework. *Wireless Communications, IEEE Transactions on*, 4(2):614–624, 2005.

[233] Kannan Srinivasan, Mayank Jain, Jung Il Choi, Tahir Azim, Edward S. Kim, Philip Levis, and Bhaskar Krishnamachari. The $\mathcal{K}$; factor: Inferring protocol performance using inter-link reception correlation. In *ACM MOBICOM*, 2010.

[234] M. Stahlberg. Radio Jamming Attacks Against Two Popular Mobile Networks. In *Helsinki U. of Tech. Seminar on Network Security*, 2000.

[235] M. Strasser, S. Capkun, C. Popper, and M. Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *IEEE S & P*, 2008.

[236] M. Strasser, C. Ppper, and S. apkun. Efficient uncoordinated fhss anti-jamming communication. In *ACM MOBIHOC*, 2009.

[237] K. Sundaresan and K. Papagiannaki. The Need for Cross-Layer Information in Access Point Selection Algorithms. In *ACM IMC*, 2006.

[238] J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[239] Trusty Tahr. Ubuntu 14.04.3 LTS. `http://goo.gl/mN2Ben`, 2014.

[240] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37:1936–1948, 1992.

[241] D. J. Thuente and M. Acharya. Intelligent Jamming in Wireless Networks With Applications to 802.11b and Other Networks. In *IEEE MILCOM*, 2006.

[242] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.*, 1:211–244, September 2001.

[243] M. Tlich, A. Zeddam, F. Moulin, F. Gauthier, and G. Avril. A broadband bowerline channel generator. In *IEEE International Symposium on Power Line Communications and Its Applications, 2007. ISPLC'07.*, pages 505–510, 2007.

[244] J. A. Torres, R. M. Davis, J. D. R. Kramer, and R. L. Fante. Efficient Wideband Jammer Nulling When Using Stretch Processing. In *IEEE Trans. Aerospace and Electr. Systems, Vol. 36, No. 4*, October 2000.

[245] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *ACM CCS*, 2012.

[246] Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift. Scheduler-based defenses against cross-vm side-channels. In *Usenix Security*, 2014.

[247] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *USENIX Security*, 2015.

[248] R. Vedantham, S. Kakumanu, S. Lakshmanan, and R. Sivakumar. Component Based Channel Assignment in Single Radio, Multi-channel Ad Hoc Networks. In *ACM MOBICOM*, 2006.

[249] L. F. M. Vieira, A. Misra, and M. Gerla. Performance of Network Coding in Multi-Rate Wireless Environments for Multicast Applications. In *IEEE GLOBECOM*, 2007.

[250] C. Vlachou, A. Banchs, J. Herzen, and P. Thiran. Analyzing and Boosting the Performance of Power-Line Communication Networks. In *ACM CoNEXT*, 2014.

[251] C. Vlachou, A. Banchs, J. Herzen, and P. Thiran. On the MAC for Power-Line Communications: Modeling Assumptions and Performance Tradeoffs. In *IEEE ICNP*, 2014.

[252] C. Vlachou, A. Banchs, J. Herzen, and P. Thiran. Performance analysis of MAC for power-line communications. ACM SIGMETRICS, 2014.

[253] C. Vlachou, J. Herzen, and P. Thiran. Fairness of MAC protocols: IEEE 1901 vs. 802.11. In *IEEE International Symposium on Power Line Communications and Its Applications*, 2013.

[254] N. V. Vladimir. The nature of statistical learning theory, 1995.

[255] A. Vlavianos, E. Law, I. Broustis, S. V. Krishnamurthy, and M. Faloutsos. Assessing Link Quality in IEEE 802.11 Wireless Networks: Which is the Right Metric? In *IEEE PIMRC*, 2008.

[256] Chih-Chun Wang. On the capacity of wireless 1-hop intersession network codinga broadcast packet erasure channel approach. *IEEE Transactions on Information Theory*, 58(2), 2012.

[257] D. J. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. In *The Computer Journal*, 1967.

[258] D. B West. *Introduction to Graph Theory*. Prentice-Hall, 2000.

[259] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Inc., 2005.

[260] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *ACM MOBICOM*, 2006.

[261] Y. Wu, P. A. Chou, and S. Y. Kung. Information Exchange in Wireless Networks with Network Coding and Physical-layer Broadcast. In *Microsoft-TR-2004-78*, 2004.

[262] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security*, 2012.

[263] W. Xu, K. Ma, W. Trappe, and Y. Zhang. Jamming Sensor Networks: Attacks and Defense Strategies. In *IEEE Network*, May/June 2006.

[264] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *ACM MOBIHOC*, 2005.

[265] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *ACM Cloud Computing Security Workshop*, 2011.

[266] Zhang Xu, Haining Wang, and Zhenyu Wu. A measurement study on co-residence threat inside the cloud. In *USENIX Security*, 2015.

[267] Yuval Yarom and Katrina E Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. *IACR Cryptology ePrint Archive*, 2013.

[268] J. Yee and H. P-Esfahani. Understanding Wireless LAN Performance Tradeoffs. In *http://www.commsdesign.com*, 2002.

[269] V. Yenamandra and K. Srinivasan. Vidyut: Exploiting power line infrastructure for enterprise wireless networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, 2014.

[270] Y. Yuan and M. J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125 – 139, 1995.

[271] Hua Zhang and Ye Li. Anti-jamming property of clustered ofdm for dispersive channels. In *IEEE MILCOM '03.*, volume 1, Oct 2003.

[272] Q.T. Zhang, X. Y. Zhao, Y. X. Zeng, and S. H. Song. Efficient Estimation of Fast Fading OFDM Channels. In *IEEE ICC '06*.

[273] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *IEEE S&P*, 2011.

[274] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In *ACM CCS*. ACM, 2012.

[275] T. Zijnge, R. Schiphorst, X. Shao, and C.H. Slump. Raptor codes for use in opportunistic error correction. In *Symposium on Information Theory(WIC10)*, 2010.

[276] M. Zimmermann and K. Dostert. A multipath model for the powerline channel. *Communications, IEEE Transactions on*, 50(4):553–559, Apr 2002.

[277] S. Zvanovec, P. Pechac, and M. Klepal. Wireless LAN Networks Design: Site Syrvey or Propagation Models? In *Radioengineering, Vol. 12, No. 4*, Dec. 2003.