PROGRAMMABLE MEDIUM ACCESS CONTROL FOR WIRELESS NETWORKS

A Dissertation

by

MUN ON SIMON YAU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	P. R. Kumar
Committee Members,	Gregory H. Huff
	Srinivas Shakkottai
	Radu Stoleru
Head of Department,	Miroslav M. Begovic

May 2018

Major Subject: Computer Engineering

ABSTRACT

The field of wireless communication is changing rapidly, with increasing numbers of wireless applications being touted as everyday staples in the future. Some examples of these applications include Virtual Reality (VR), Augmented Reality (AR), Internet Of Things (IoT), tactile internet, smart power grids, unmanned traffic management systems, remote medical procedures, disaster network deployment, sensor networks, and cloud computing, to name just a few. However, for many of these applications to fulfill their stated potential, the required system performance that wireless networks need to have cannot be achieved with current deployments. One of the major factors in the performance of wireless networks is the Medium Access Control (MAC) layer.

The MAC layer serves as the junction between the Physical (PHY) and Network (NET) layers, and its efficiency is key to the performance of communication networks. This is especially critical for wireless networks, since wireless transmissions introduce interference into the system which can be detrimental to the performance of wireless networks. While the MAC layer consists of both software and hardware components, researchers overwhelmingly only perform computer simulations to verify the performance of their protocols, leaving the potential impact of hardware on the performance largely untested. Unlike the Physical (PHY) and Network (NET) layers, there is little experimental verification of the performance of novel protocols suggested for the MAC layer. Hence, protocols developed for the MAC layer do not get verified at a level commensurate with the claims of the protocols, thus generating no momentum to drive the adoption of new protocols. The end result is that the protocols in current use are little more than variants of the original ALOHA protocol proposed in 1970. So motivated, in this dissertation, I present a platform for implementation and experimentation of next-generation wireless MAC protocols, along with a novel architecture that uses a hardware-software decoupling principle

to achieve flexibility without loss in performance and show how these platforms can aid prototyping of MAC protocols in the future. In addition to that, I will also present a MAC protocol for mmWave networks that can be implemented directly on the IEEE 802.11ad standards.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a dissertation committee consisting of Professor P. R. Kumar, Professor Srinivas Shakkottai, and Professor Gregory Huff of the Department of Electrical and Computer Engineering, and Professor Radu Stoleru of the Department of Computer Science and Engineering.

The scheduling code for Chapter 2 was jointly developed by Kartic Bhargav, Rajarshi Bhattacharyya, Ping-Chun Hsieh, and myself. Experiments in this chapter were performed by Rajarshi, Ping-Chun and myself. In Chapter 3, Ping-Chun developed the host code for running experiments for a fixed period of time. He also developed the timeout mechanism which was used primarily for the Max-Weight protocol described in this chapter. Bharadwaj Satchidanandan developed the simulation code for the mmWave protocol in Chapter 4. All other work conducted for the dissertation was completed independently by me.

Funding Sources

This work was made possible in part by the National Science Foundation (NSF) under Contract Numbers CCF-1619085 and CNS-1302182.

Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NSF.

NOMENCLATURE

3GPP	3rd Generation Partnership Project
A-BFT	Association Beamforming Training
AP	Access Point
AR	Augmented Reality
ATI	Announcement Time Interval
BHI	Beacon Header Interval
BI	Beacon Interval
BSS	Basic Service Set
BTI	Beacon Transmission Interval
CE	Channel Estimation
CEF	Channel Estimation Field
CPU	Central Processing Unit
CSMA	Carrier-Sense Multiple Access
DCF	Distributed Coordination Function
DTI	Data Transmission Interval
FIFO	First-In-First-Out
FPGA	Field-programmable Gate Array
ICN	Information-centric Networking
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things

LAA	License-Assisted Access
LoS	Line-of-sight
LTE	Long Term Evolution
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
mmWave	Millimeter Wave
MU-MIMO	Multi-user Multiple-Input-Multiple-Output
NET	Network
NOMA	Non-orthogonal Multiple Access
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
PCF	Point Coordination Function
PHY	Physical
PIFS	PCF Interframe Space
RF	Radio Frequency
SC-PHY	Single-Carrier PHY
SDN	Software-defined Networking
SDR	Software-defined Radio
SIFS	Short Interframe Space
SNR	Signal-to-Noise Ratio
STF	Short Training Field
T_tc	Topological Coherence Time
TDMA	Time-Division Multiple Access
TRN	Beamforming Training Field

ULA	Uniform Linear Array
USRP	Universal Software Radio Peripheral
VR	Virtual Reality
WARP	Wireless Open-Access Research Platform
WLAN	Wireless Local Area Network
WMP	Wireless MAC Processors

TABLE OF CONTENTS

Page

ABSTRACT ii

CONTRIBUTORS AND FUNDING SOURCES iv

NOMENCLATURE v

TABLE OF CONTENTS viii

LIST OF FIGURES x

LIST OF TABLES xiii

1. INTRODUCTION 1

2. PULSE: A PLATFORM FOR ULTRA-LOW LATENCY WIRELESS SCHEDULING EXPERIMENTATION 4

    2.1 Introduction 4
    2.2 Related Work 8
    2.3 Design of PULSE 9
        2.3.1 Basic MAC Functions for Wireless Scheduling 10
        2.3.2 Mechanism-Policy Separation 10
        2.3.3 Flexible MAC Through Host-FPGA Separation 11
    2.4 Implementation of PULSE 12
        2.4.1 Packet Generation 13
        2.4.2 Queueing, Deadlines and Types of Flows 13
        2.4.3 Scheduling and Transmission Procedures 15
        2.4.4 Retransmission 17
    2.5 Measuring Host-to-FPGA Interfacing Latency and Round-Trip Latency 19
    2.6 Experimental Results 24
        2.6.1 Capacity Regions 26
        2.6.2 Throughput Performance 26
        2.6.3 Changing the Arrival Process 28
    2.7 Conclusion 30

3. WIMAC: PLATFORM FOR RAPID IMPLEMENTATION OF MEDIUM ACCESS CONTROL PROTOCOLS 32

3.1 Introduction 32
3.2 Related Work 35
3.3 Key Design Principles for MAC Platform 37
3.4 Architectural Design of The Platform 39
3.4.1 Identifying Distinguishing Features 39
3.4.2 Designing MAC Protocols in Software and Decoupling 40
3.4.3 Enabling Reconfiguration on the Fly 42
3.5 Implementation of Different Classes of MAC Protocols 44
3.5.1 Link Performance 44
3.5.2 CSMA 45
3.5.3 CHAIN 46
3.5.4 Max-Weight Scheduling Policy 50
3.5.5 A Meta Protocol 56
3.6 Extensibility of Platform 59
3.7 Conclusion 60

4. MEDIUM ACCESS CONTROL FOR DIRECTIONAL MILLIMETER WAVE WIRELESS NETWORKING 62

4.1 Introduction 62
4.2 Related Work 65
4.3 Design Methodology of TrackMAC 66
4.3.1 Topological Coherence Time 67
4.3.2 TrackMAC: A Novel Directional MAC Protocol 68
4.4 Implementation of TrackMAC Within the IEEE 802.11ad Specifications 72
4.5 Simulation Results 76
4.6 Conclusions and Future Work 81

5. CONCLUSIONS 83

REFERENCES 85

LIST OF FIGURES

FIGURE Page

2.1 The scheduling problem with N flows. 5

2.2 Architecture of PULSE. 14

2.3 Packet transmission in PULSE. 18

2.4 Empirical CDF of Host-to-FPGA latency for payload size = 1500 bytes and data rate = 54Mbps. 21

2.5 Empirical CDFs of round-trip latency for various data rates and payload sizes. 22

2.6 Timely-throughput vs. Deadline for 20 packets generated every 11 ms. 23

2.7 Experimental setup with host machines and USRP-2153R's being used as wireless AP and clients. 23

2.8 Capacity regions for the different Policies with 5 ms deadline and 0.99 delivery ratio. 25

2.9 Throughputs for K_{RT}=4 and K_{NRT}=4. 31

2.10 Throughputs for K_{RT}=3 and K_{NRT}=6. 31

2.11 Throughputs for K_{RT}=3, K_{NRT}=5 for client 1, K_{RT1}=4, K_{NRT}=6 for client 2. 31

2.12 Throughputs for K_{RT}=7, K_{NRT}=4 for client 1, K_{RT1}=3, K_{NRT}=5 for client 2. 31

2.13 Throughputs for the second arrival process with P(K_{RT1}) = 0.8, P(K_{NRT1}) = 0.3, P(K_{RT2}) = 0.1, P(K_{NRT2}) = 0.5. 31

2.14 Deficit growth for the K_{RT}=4 and K_{NRT}=4 for both clients. 31

3.1 Example of a generic function block in WiMAC. 41

3.2 Example of reconfiguration packet. 43

3.3 System throughput under CSMA and CHAIN vs different CWmin: (a) CWmin = 4, (b) CWmin = 8, (c) CWmin = 16, (d) CWmin = 32. 47

3.4 Example of piggyback transmissions with three clients and an AP in CHAIN. 48

3.5 Piggyback transmission block in WiMAC. 49

3.6 How (a) downlink and (b) uplink transmissions are done in Max-Weight. . . . 51

3.7 Block used in Max-Weight to obtain the TX queue length of the node. 52

3.8 Block used in Max-Weight to associate different clients with their respective (weighted) queue lengths. 52

3.9 Example of Max-Weight scheduling with two clients. 54

3.10 System throughput under Max-Weight scheduling versus different timeout thresholds. 57

3.11 How different functional blocks interact with each other in the Meta Protocol. 58

3.12 System throughput under different protocols. The system uses CSMA from 0 to 54s, then CHAIN until 98s, and finally the Meta Protocol until the end. 59

3.13 A possible implementation of a sleep block in hardware. 61

4.1 A mmWave network operating in infrastructure mode. 67

4.2 Division of time into macroslots, of macroslots into microslots. 69

4.3 Structure of a Beacon Interval. 72

4.4 Overlay of 802.11ad PHY packets on microslots and macroslots on 802.11ad BI. 74

4.5 802.11ad PHY packet structure. 75

4.6 Azimuth pattern of a $\lambda/2$-spaced 64-element ULA at 60.48GHz. 77

4.7 Tracking performance of the AP for three different STA translational speeds (5m/s, 10m/s, and 20m/s). Plotted against time are (i) the azimuth direction of a particular randomly chosen STA out of the 25 associated STAs, and (ii) the azimuth direction in which the AP points its transmit beam when it communicates with that particular STA. For STA translational speeds of 5m/s and 10m/s, the AP is able to track the movement of the STA, whereas for STA translational speed of 20m/s, the AP is unable to do so. 78

4.8 Average received signal power vs. translational speed of STAs. The abscissa denotes the avg. speed at which the STAs move, and the ordinate denotes the avg. power level at which the MAC payload is received. The averaging is performed over STAs. 79

4.9 Azimuth pattern of the ULA with steering weights corresponding to 3.2° azimuth. 80

4.10 Tracking performance of AP and STAs for different rotational speeds. The abscissa denotes the avg. rotational speed of the STAs and the ordinate represents the angle between the directions of the transmitter's beam and the intended receiver's beam, averaged over both STAs and time. The network was simulated for one beacon interval. 81

4.11 Average received signal power vs. rotational speed of STAs. The abscissa denotes the avg. speed at which the STAs rotate, and the ordinate denotes the avg. power level at which the MAC payload is received. The averaging is performed over STAs. 82

LIST OF TABLES

TABLE Page

2.1 Partial list of mechanisms in PULSE 12
2.2 Host-to-FPGA latency results 21
2.3 Round-trip latency results 22
2.4 Loss ratios of real-time flows 29
3.1 Link throughput for different modulation schemes and coding rates 45
3.2 System throughput under Round-Robin scheduling and Max-Weight scheduling 55

1. INTRODUCTION

Wireless networks have come a long way since their inception, and have become the *de facto standard* for devices that connect to the internet. As the number of wireless devices continues to increase, wireless network systems have to meet increasingly stringent performance demands to accommodate these devices. Furthermore, the applications that are being proposed for next-generation wireless devices go beyond the standard metrics of throughput and fairness. For example, Virtual Reality (VR) and Augmented Reality (AR) require high throughputs with strict latency guarantees, while unmanned traffic management systems require high reliability. On the other hand, Internet of Things (IoT) and sensor networks typically require low energy rather than low latencies, while remote medical procedures, and to a certain extent smart power grids, typically require both high reliability and low latencies [1]. These applications, spanning a wide range, have different performance goals which can be in conflict with each other, and next-generation wireless networks will need to provide high system performance as well as the flexibility to support all the different system requirements. To achieve this goal of performance and flexibility, researchers have been working on key enabling technologies to increase the performance and efficiency of wireless networks.

For example, at the Physical (PHY) layer, advancements have been made in Radio Frequency (RF) electronics on the RF front end, as well as the data flow and processing speeds of the encoding and decoding chains of radios. These improvements allow for wireless communication to be performed in extremely high frequency bands, such as mmWave bands (from 30GHz to 300GHz), which were previously thought to be unusable [2, 3]. By leveraging these bands for communication, the amount of spectrum available for wireless communication can be increased by several orders of magnitude. In addition to that,

researchers have shown that the spectral efficiency and Signal to Noise Ratio (SNR) of wireless communication can be greatly increased by using Multiple Input Multiple Output (MIMO) [4, 5], Non-Orthogonal Multiple Access (NOMA) [6, 7] and advanced interference cancellation techniques. These technologies also serve to improve the efficiency of the Medium Access Control (MAC) layer. Multiuser MIMO (MU-MIMO) [8] builds upon the principles of MIMO to schedule multiple users at the same time, whereas interference cancellation allows full-duplex wireless communication on the same frequency bands. MIMO also allows for beamforming signals in a desired direction, and can be used to reduce interference to surrounding nodes.

The MAC layer also functions as the junction between the PHY layer and the Networking (NET) layer, and is therefore responsible for scheduling data from the NET layer to the PHY layer. In most commercial deployments today, the MAC layer works on a simplistic First In First Out (FIFO) basis, with minimal control over how to schedule the different flows from the NET layer. Researchers have proposed many scheduling policies to improve the efficiency and flexibility of the MAC layer, such as in [9, 10, 11, 12, 13, 14, 15, 16, 17].

At the network layer, Software Defined Networking (SDN) [18, 19], virtualization and network slicing [20, 21, 22, 23, 24, 22] have given increased control for network utilization and configuration options. Furthermore, research is also being done to shift the traditional paradigm of connecting two endpoints to one of connecting users with their desired content through Information-centric networking (ICN) [25]. With ICN, content is cached within multiple nodes in the network as opposed to data centers. This can potentially reduce the latency users that experience when requesting content of interest.

These advancements, however, are not without their challenges. For the PHY layer technologies, researchers have prototyped hardware systems to prove the claims of their improvements. Similarly, since the NET layer technologies are typically implemented

in software, NET layer researchers are able to prototype their technologies in software and verify the claims of improvements that can be made about the NET layer performance. When it comes to the MAC layer, matters are not as simple since the MAC layer is very tightly intertwined with the PHY layer. In fact, apart from MAC layer technologies that leverage the use of PHY layer technologies, such as MU-MIMO and Orthogonal Frequency-Division Multiple Access (OFDMA), MAC layer protocol performance evaluation largely rests on computer simulation for verification of the claims of performance, or any claims of improving performance over other suggested protocols. There has been precious little experimentation of MAC protocols. Moreover, since the MAC layer serves as the junction between the PHY and NET layer, it deals with both hardware and software, and without experimentation, some problems with MAC protocols can easily be overlooked when only simulations are performed for verification, as I will show in Chapter 3.

In this dissertation, I will focus on the challenges associated with prototyping next generation MAC protocols, and will present a platform for experimentation with scheduling protocols for low-latency applications, an architecture for flexible MAC designs which can reduce prototyping time for next-generation MAC protocols. I will also introduce a key concept, called the in mmWave MAC protocol design, called *topological coherence time*, and propose a protocol for a scheduled MAC protocol for mmWave networking.

2. PULSE: A PLATFORM FOR ULTRA-LOW LATENCY WIRELESS SCHEDULING EXPERIMENTATION

2.1 Introduction

In this chapter, I will focus on the scheduling problem for heterogeneous flows in next-generation wireless networks; in particular on how researchers can experiment with protocols that provide delay guarantees in wireless networks. A strict latency requirement is currently one of the most critical challenges for next-generation wireless networks. Emerging applications, such as VR [26], factory IoT, and tactile Internet [27], require an end-to-end latency between 1 to 10 millseconds (ms) to provide seamless user experience. However, existing wireless networks cannot provide such stringent latency guarantees. For example, the round-trip time of Long Term Evolution (LTE) is estimated to be at least 20 ms, including the transmission time, scheduling overhead, and processing delay [28]. In practice, the current LTE technology can only support voice or video streaming applications with round-trip time in the range of 20-60 ms [29]. For Wi-Fi networks, due to the nature of random access, the round-trip time could vary from several ms to hundreds of ms depending on the traffic load [30]. Therefore, compared to the current technology, the latency budget is expected to be at least one order of magnitude smaller in next-generation wireless networks.

To provide strict per-packet latency guarantees, numerous theoretical solutions have been proposed to accommodate *per-packet deadline constraints* in wireless scheduling. [31] and [32] propose a theoretical framework to study wireless network scheduling with per-packet deadline constraints. In the framework of [31], N flows are scheduled by a single Access Point (AP) as shown in Figure 2.1. Each packet in the flow is associated with a deadline. Packets that are not delivered on time are dropped. In [32] this is extended

to multi-hop wireless networks.

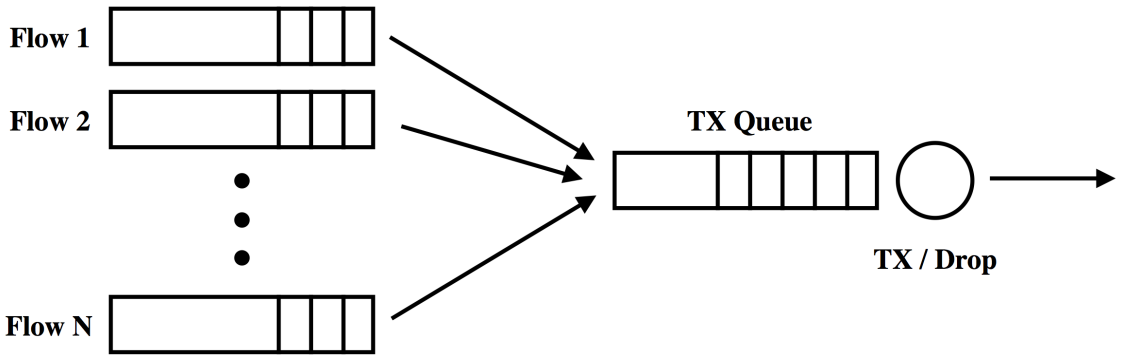

Figure 2.1: The scheduling problem with N flows.

Subsequently, this framework has been applied to many other scenarios, such as utility maximization [14], scheduling for both latency-constrained and best-effort traffic [15], broadcast traffic [16], and multicast traffic [17], as described in [33]. The performances of these protocols are usually measured by *timely-throughput*, i.e., the time average of the amount of data delivered within their deadlines. While the above proposals are promising, there has been no implementation of these ultra-low-latency wireless protocols. In this chapter, we allow for N flows, which consist of real-time or non-real-time flows. Real-time flows contain packets with deadlines, and non-real-time flows contain packets without deadlines. (For packets without deadlines, queue lengths become an important metric for measuring performance of the protocol.)

To prototype these ultra-low-latency wireless protocols and achieve timely-throughput that closely matches what is achieved in practice, a powerful software-defined radio (SDR) platform with dedicated architectural design is required to provide enough latency budget as well as to minimize the software and the hardware processing and interfacing over-

head. Various SDR architectures have been proposed to mitigate the interfacing overhead between the software host and the hardware, which can include Field-Programmable Gate Arrays (FPGA). Just to name a few, [34] proposes a split-functionality framework to significantly reduce the communication overhead between the software host and the hardware. Similarly, [35] introduces Decomposable MAC Framework to identify basic functional components according to both timeliness and degree of code reuse. However, neither of them considers per-packet deadline constraints nor provides any experimental results for ultra-low-latency wireless networks.

This platform aims to bridge the gap between theory and implementation for wireless networks with strict per-packet latency constraints. We propose PULSE, a software-defined wireless platform for prototyping ultra-low-latency scheduling protocols. To achieve the required per-packet latency performance, PULSE needs to address the following challenges:

1. **Enforce per-packet deadline on a software-defined wireless platform.** PULSE aims to support per-packet latency as low as 1 ms. When packets arrive at the software host, they are first queued and start waiting for transmission according to some scheduling policy. The deadline of each packet in the queue needs to be tracked and checked before transmission. A packet that already missed its deadline should be dropped from the queue. Moreover, due to the nature of SDRs, packet transmission is carried out on the hardware while packet scheduling is often done on the software host. Therefore, it is not directly clear how deadlines should be maintained and checked on a software-defined platform if there is no synchronization between the software and the hardware. Moreover, it usually requires non-trivial efforts to provide an accurate reference timer shared by the software host and the hardware. In Section 2.4.3, I present a simple design that tracks packet deadlines accurately

using only timestamps of the software host, without any synchronization between the software host and the hardware.

2. **Low interfacing latency between the software host and the hardware.** There are three major factors that affect the end-to-end latency: (i) queuing delay on the software host, (ii) interfacing latency between the software host and the hardware, and (iii) the hardware processing time. Queuing delay depends mainly on the scheduling policy, and the hardware processing time can usually be made small due to the high clock rate supported by current technology. Therefore, with a proper choice of the scheduling policy and the hardware component, interfacing latency between the software host and the hardware needs to be minimized in order to achieve ultra-low latency. In Section 2.5, I present a simple experiment that demonstrates the interfacing latency of PULSE is indeed small compared to packet deadlines.

3. **Achieve realistic per-flow timely-throughput.** Ultra-low latency needs to come with realistic timely-throughput. Given the same physical data rate, the overhead of enforcing per-packet deadlines could be quantified by the difference in total MAC-layer throughput between the networks with and without packet deadlines. In Section 2.6.1, through an experimental study on system capacity, PULSE is shown to achieve almost the same MAC-layer throughput as that with no deadlines.

4. **Support functions working on heterogeneous time scales.** MAC layer functions operate on very different time scales. For example, an acknowledgement (ACK) response needs to be performed within tens of microseconds. The transmission time of a typical data packet of 1500 Bytes at 20Mbps is between 0.5 to 1 ms. The target per-packet deadline is between 1 to 10 ms. The parameters of wireless protocols usually change over a period of at least several seconds to several minutes. In Section 2.3, I describe the separation principles of PULSE which inherently incorporate

the heterogeneity in time scale.

5. **Support various ultra-low-latency downlink applications.** For example, VR requires latency as low as 1 ms with moderate timely-throughput while factory automation needs ultra-low packet loss rate with latency of about 5-10 ms. PULSE is able to support applications with totally different performance requirements and provide a programmable environment for different wireless protocols.

To tackle the above challenges, PULSE follows three major design principles. First, as a software-defined wireless testbed, PULSE follows the Host-FPGA separation principle for both high flexibility and performance. Next, PULSE addresses the heterogeneous time scales of MAC functions by applying Mechanism-Policy separation. Third, to support a broad class of scheduling policies, we borrow ideas from both WiFi as well as LTE standards, and build up a set of basic MAC functions required by most of the wireless protocols.

The rest of the chapter is organized as follows. Section 2.2 summarizes the related works on low-latency wireless networks. Section 2.3 describes the design principles of PULSE. The implementation of PULSE is detailed in Section 2.4. Section 2.5 discusses the interfacing latency. Section 2.6 provides an extensive experimental study on ultra-low-latency protocols. Finally, Section 2.7 concludes the chapter.

2.2 Related Work

Most of the existing experimental studies for wireless LANs focus primarily on maximizing system throughput or throughput-based network utility. For example, to achieve maximium throughput, the well-known backpressure algorithm has been tailored and implemented for various scenarios, such as multi-hop wireless networks [36], Time Division Multiple Access (TDMA)-based MAC protocol [37] and wireless networks with intermittent connectivity [38]. Besides, for wireless LAN with random access, [39] implements an

enhanced version of 802.11 Distributed Coordination Function (DCF) and demonstrates that it achieves near-optimal throughput as well as fairness with the original DCF. However, all of the above studies provide no support for packets with latency constraints. To address latency requirement for industrial control applications, RT-WiFi, a WiFi-compatible TDMA-based protocol, has been proposed and implemented on commercial 802.11 interface cards [40]. However, it cannot achieve both ultra-low latency and satisfactory timely-throughput performance for each user at the same time due to the nature of TDMA.

On the cellular side, several preliminary studies of 5G provide candidate solutions to enhance the low-latency capability via either numerical and experimental evaluation. [41] studies the trade-off between latency budget and required bandwidth by applying the conventional OFDM framework to 5G networks through numerical analysis. However, these numerical results do not take the possible signaling and processing overheads into account. [42] provides experimental study for latency performance of 5G millimeter-wave networks with beamforming. However, this solution relies heavily on the beam-tracking technique and frame structure employed by cellular networks and cannot be directly applied to wireless LAN applications. [43] demonstrates a wireless testbed that is potentially capable of supporting millisecond-level end-to-end latency requirement. However, it supports only single link and does not take wireless scheduling issue into account.

2.3 Design of PULSE

Our objective was to develop a platform for testing ultra-low latency protocols that require scheduling on a per packet basis, with a focus on downlink protocols. In this section, I will explain the basic design principles upon which PULSE was built to achieve the goals described in the previous sections.

2.3.1 Basic MAC Functions for Wireless Scheduling

Our platform borrows ideas from both the WiFi and LTE standards. From the WiFi side, we use features such as Carrier Sense for loose synchronization between the nodes and some robustness to interference. We also use the same interframe space timing intervals as WiFi for transmission (and reception) of packets. Furthermore, ACKs are sent immediately following transmission of data packets after a Short Interframe Space (SIFS) period, specified in [44] which allows us to know in a short period of time whether the packet was transmitted successfully. On the other hand, we use a completely centralized framework for scheduling the different queues, which is similar to what LTE does. By using ideas from WiFi and LTE, we are able to obtain a deployment that is both lightweight and can be incorporated more easily into our framework for ultra low latency scheduling.

2.3.2 Mechanism-Policy Separation

In our design of PULSE, we utilize a mechanism-policy separation used in [45]. Mechanisms are functions or hardware blocks used to handle the low-level operations of packet transmissions over the network, whereas policy refers to the high-level specification of the scheduling protocol itself. This mechanism-policy separation builds on the framework of Wireless MAC Processors, introduced by Tinnirello et al in [46].

Each mechanism has a set of inputs, outputs, events, conditions to check and possible actions that can be performed. Inputs and outputs of a mechanism take the form of register values (e.g. channel state and average energy), or an array of bytes (e.g. my address and packet).

Mechanisms provide the set of actions and events, and also act as condition checkers, whereas the policy side specifies the set of enabling functions, the parameters for the conditions, the set of update functions and the transition relations for the state machine of the scheduling protocol. Maintaining the distinction between the different mechanisms

and their associated events, actions, and conditions, allows us to design new mechanisms more cleanly, and reuse previously developed mechanisms. In addition to the mechanisms used in [45] and [46], we implement mechanisms to allow for deadline checking, packet dropping, controlling the packet arrival process, as well as features to increment and decrement some notion of "deficit" according to user-specified conditions, where a "deficit" is roughly the difference between what is aimed at versus what has been delivered. Deficits can be used in various ways to achieve some performance guarantees for delay-sensitive traffic, or to control the ratio of service times between real-time and non-real-time flows. We will focus more on these mechanisms since they allow us to achieve certain guarantees on delay-sensitive traffic. These mechanisms can also be changed at runtime, allowing us to switch between different protocols on-the-fly. The implementation details of these mechanisms are laid out in Section 2.4. It should be noted that in the design of our testbed, mechanisms are implemented both on the FPGA as well as on the host machine. Furthermore, the inputs and outputs for each mechanism can be modified accordingly to allow for cross-layer designs. For example, the update deficit mechanism can use the packet's Modulation and Coding Scheme (MCS) as an input for the function to increment or decrement the deficits.

2.3.3 Flexible MAC Through Host-FPGA Separation

For flexible MAC scheduling decisions, we employed a Host-FPGA separation, where high-level MAC functions such as packet scheduling and packet dropping are done on the host, and low-level functions such as packet encoding/decoding, carrier sensing, ACK processing, and CRC checking are done on the FPGA. This is done to allow for easy changes in packet scheduling decisions, while still being able to achieve the latency requirements for the platform.

Table 2.1: Partial list of mechanisms in PULSE

Mechanism	Input	Ouptut	Description
Packet Generation	Interarrival times Probability of generation Max packets generated	Packets	Generates packets based on user specified parameters.
Prepend Deadlines	Current tick count Deadline Packets Queue references	N/A	Prepends deadlines to incoming packets and puts the packets into the correct queues.
Packet Dropping	Queue references Current tick count	Packet dropped?	Drops packets if deadlines have elapsed.
Update States	Queue references Packet dropped? ACK received count ACK timeout count Current state	Updated states	Updates the states associated with each flow.
Scheduling Decision	Current state Policy to use	Flow to schedule (or retransmit)	Decides which flow to schedule based on specified policy.
Retransmission	Current tick count Queue references ACK received count ACK timeout count Re-TX attempts	N/A	Stores transmitted packet, and retransmits if ACK was not received and deadline has not elapsed.

2.4 Implementation of PULSE

The implementation details of PULSE is discussed in this section, namely, the flow of events, and the design of mechanisms for the nodes. A partial list of all mechanisms can be found in Table 2.1.

Since our platform focuses on downlink scheduling, the mechanisms shown here are geared towards that, although most can be reused for uplink scheduling as well. Starting with the National Instruments 802.11 Application Framework, we have implemented additional mechanisms on the Host machine and on a USRP-2953R for flexible scheduling

protocols. To support packets with strict deadlines, PULSE presents a data transmission procedure which enables per-packet scheduling on the host while keeping the FPGA design simple. The overall system architecture is summarized in Figure 2.2.

2.4.1 Packet Generation

For packet generation, we implement a packet generation mechanism that allows three main parameters that can be tweaked to replicate the different kinds of traffic that a user might want to experiment with: 1) interarrival times, 2) probability that packets get generated, 3) number of packets that are generated. Packets can be generated with interarrival times up to 1 ms in resolution, with a certain number of packets being generated at each time. In addition to that, packets can be generated deterministically or stochastically. At each interrival time, users can specify the probability that a certain number of packets get generated. Furthermore, the number of packets can be set to be a fixed number, or can take on a random number based on a distribution. For our purposes, in the random case, we use a simple uniform distribution for the number of packets that are generated. The maximum number of packets that can be generated is specified by the user. These arrival patterns and rates are by no means exhaustive, but it is sufficient for us to justify the performance of the platform and of particular protocols that we have implemented. (The details of the results are in Section 2.6.)

2.4.2 Queueing, Deadlines and Types of Flows

Each data flow, either real-time or non-real-time, is associated with a queue on the Host. To add deadlines to the packets, we implement a simple mechanism to prepend deadlines to each packet arriving at the queue. The input of this block is the current tick count of the system, the deadline, the incoming packet, and all the references to the queues associated with each flow. This block takes the current tick count of the system, adds the correct tick count corresponding to the deadline of the packet and prepends it to the packet

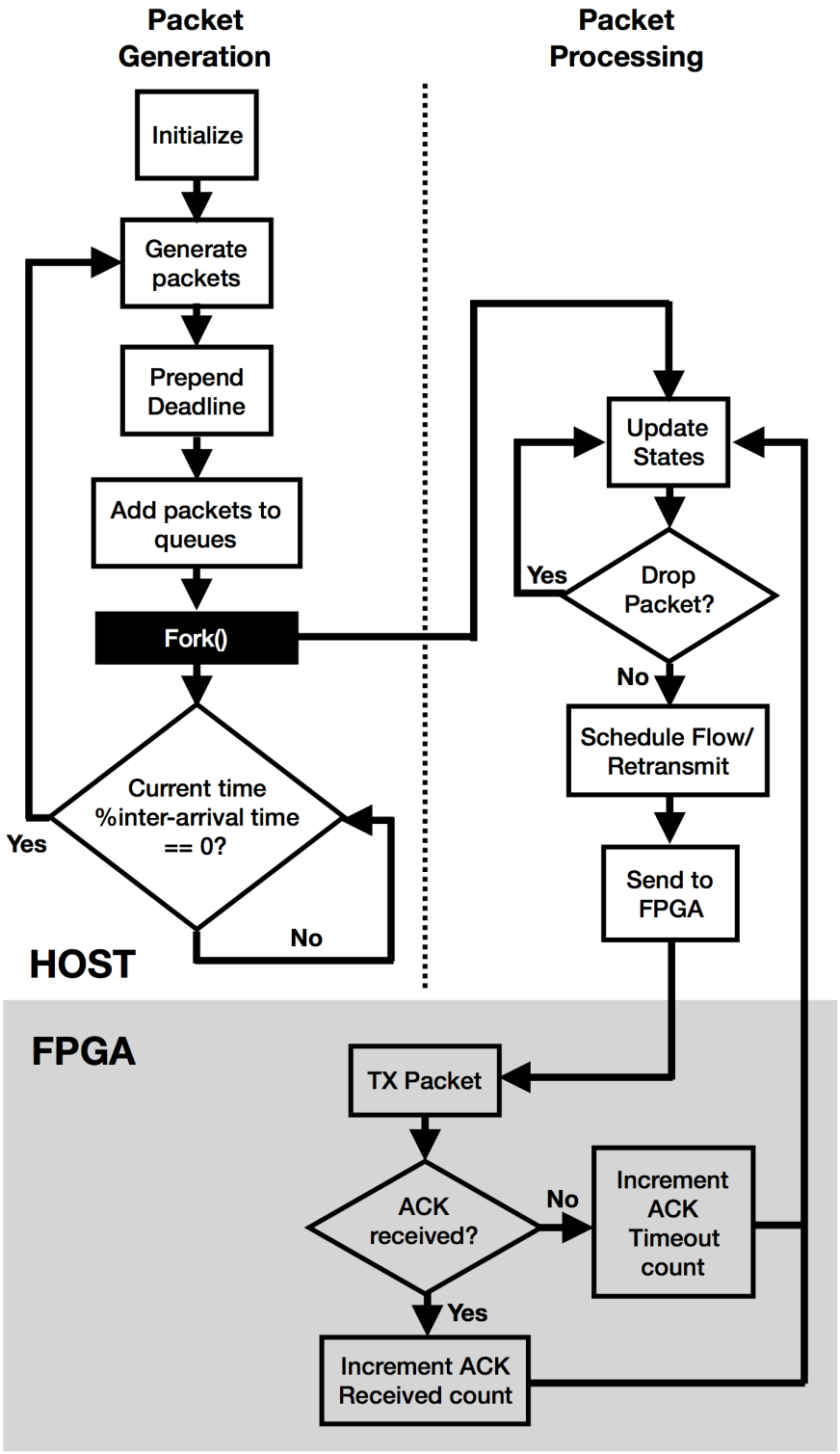

Figure 2.2: Architecture of PULSE.

before adding it to the correct queue. For real-time flows, when a packet is generated (according to the Packet Generation mechanism), the corresponding *deadline identifier*, in the format of Host reference timer, is prepended onto the packet and then the packet is queued for scheduling. Since the absolute packet deadlines of each flow are assumed to form a non-deceasing sequence, every queue on the Host is always inherently arranged in an earliest-deadline-first manner. (Note, this may not be true if packet deadlines are changed frequently in a non-increasing manner. However, applications typically just require a baseline performance guarantee, so this is valid for most cases. In the event that deadlines for packets are not arranged in an earliest-deadline-first manner, we can sort the packets while packets are being transmitted.) On the other hand, for non-real-time flows, the packet deadlines are set to be negative one for simplicity and better modularity in design.

2.4.3 Scheduling and Transmission Procedures

Once packets are generated and the packets are in their respective queues, scheduling is performed and repeated on a per-packet basis. In PULSE, scheduling is aided by the use of several mechanisms to achieve our desired latency goals. First, we have a packet dropping mechanism which scans the head-of-line packet of each queue, and drops the packets that have expired. This block only requires the references of each queue, and the current tick count of the system as inputs. It also has an output to inform other mechanisms whether or not a packet has been dropped. Next, we have a mechanism to update the state of the flows. Its job is to update the deficits associated with each flow based on whether or not an ACK was received and if a packet has been dropped from the queue. It uses the references of the queues, ACK received count, ACK timeout count, and the output of the packet dropping mechanism as its inputs. Lastly, we have a scheduling decision block that decides which flow to schedule based on the current states of the flows. This mechanism

uses the deficits associated with each flow, the ACK counters for each of the flow, and the output of the packet dropping mechanism as inputs. All scheduling-related mechanism blocks are executed on the Host.

Scheduling and transmission executions are triggered when at least one queue is non-empty, and the scheduling state is UNLOCKED. The scheduling state becomes LOCKED as soon as a packet has been scheduled and is being processed for transmission. The state becomes UNLOCKED again when either of the two following FPGA events happen: an ACK reception or an ACK timeout. A successful ACK reception happens when an ACK packet is decoded successfully. This in turn increments the number of ACKs that have been received. The Host machine polls the register that stores the number of ACKs that have been received, and when the numbers differ from one iteration of the while loop to the next, the Host registers an ACK reception. For ACK timeouts, we implement a mechanism that starts a counter after a packet has been transmitted on the FPGA. If an ACK is not received within a specified time period (set to be 75 microseconds), the count for timeout is increased on the FPGA. As in the case of ACK reception, if the count for ACK timeouts that the Host polls from the FPGA differs from one while loop iteration to the next, an ACK timeout event is registered. The timing of loop executions is described further in Section 2.5. In every loop cycle, exactly one packet is scheduled for transmission. Before making a scheduling decision, the Host first "cleans up" the queues and updates the deadline-related state information by dropping expired packets in each queue using the mechanisms detailed above. Given the computing power of the Host, this cleanup can be finished almost instantaneously compared to the transmission time of a packet. (We should note that there will be more overhead if a batch of packets have expired, since we are only dropping one packet per loop execution, but this time is also negligible relative to the packet transmission time.) Given the queues in a clean state, the flow scheduler sets the priorities of the flows according to the scheduling policy and schedules the flow with a

non-empty queue and the highest priority. The scheduled queue then sends the head-of-line packet to the Prepare ICP Packet block, which removes the deadline identifier and appends the ICP header to the packet. The ICP header carries the information required by the FPGA, such as packet length, modulation and coding scheme, source MAC address, destination MAC address, and flow identifier, etc. Upon receiving the scheduled packet from the Host, the FPGA simply triggers the required channel access procedure of the MAC layer as well as the physical transmission procedure in the PHY layer. By placing all the scheduling complexity on the Host, the design of PULSE can be easily reproduced on an existing wireless interface card. The AP packet transmission procedure is summarized in Algorithm 1 and the function blocks associated with the transmission procedure are depicted in Figure 2.3.

2.4.4 Retransmission

As reliable transmission is often required by mission-critical low-latency applications, PULSE also supports retransmission for both real-time and non-real-time flows to recover packet losses. Note that while MAC-layer retransmission is usually implemented in the hardware for conventional SDRs to minimize latency, the retransmission block of PULSE is located in the Host for two reasons: (i) Packet deadlines need to be checked before any retransmission. Since deadlines are tracked in the Host, it is straightforward to handle retransmission in the Host. (ii) The Host-to-FPGA interfacing latency is low enough for supporting retransmission in the Host.

In PULSE, retransmission is built on top of the scheduling and data transmission procedure described in Section 2.4.3. An additional retransmission mechanism with retransmission queue is created on the Host for temporarily storing the duplicate of the scheduled packet in the current loop cycle. The inputs to this block are the references to all the queues, the ACK counts for the flows, and the maximum retransmission attempts which

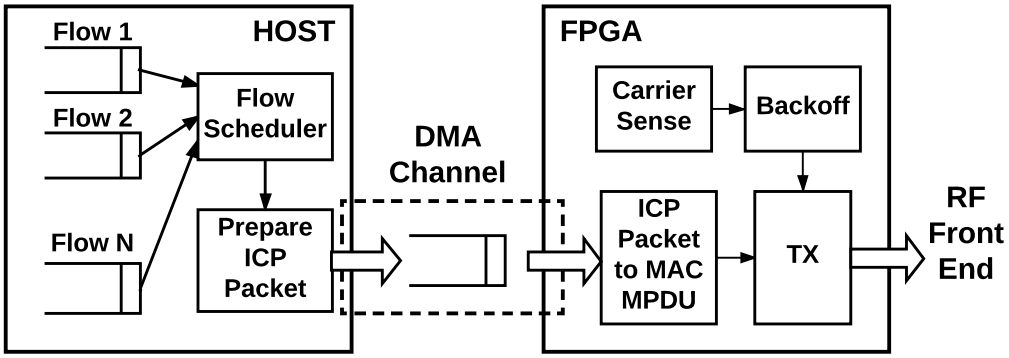

Figure 2.3: Packet transmission in PULSE.

can be easily configured in the Host according to the user specified scheduling policy. If an ACK timeout is received and the maximum retransmission count is not met, then the scheduler attempts to retransmit the same packet in the next cycle; otherwise, the duplicate packet is removed from the retransmission queue.

The ICP packet is sent to the SDR's FPGA fabric via a Direct Memory Access (DMA) Channel. All PHY layer processing required to transmit the packet is performed on the FPGA using the mechanisms provided by the 802.11 AF [47]. While the platform supports mechanisms for random backoff, we set backoffs to zero since our scheduling algorithms are completely centralized.

After the packet is transmitted, the Received Packet mechanism determines whether an ACK was received or not. If an ACK was received, then the ACK count is incremented. Otherwise, the ACK timeout count is incremented. Received packets are then sent back to the Host for processing, again via a DMA Channel. Concurrently, the Host polls the FPGA registers for transmitted packet count, ACK count, and ACK timeout count, and updates the deficits accordingly based on these counts.

Algorithm 1: AP transmission procedure.

```
1  Initialize the state variables and the Host queues;
2  while station is ON do
3      update Host queues and state variables;
4      schedule the flow with the highest priority;
5      send the scheduled packet to Prepare ICP Packet;
6      state of flow scheduler ← LOCKED;
7      while state of flow schedule is LOCKED do
8          if receive ACK response then
9              state of flow scheduler ← UNLOCKED;
10         end
11     end
12 end
```

2.5 Measuring Host-to-FPGA Interfacing Latency and Round-Trip Latency

The interfacing latency between the software and the hardware significantly affects the achievable link throughput of a software-defined wireless testbed. As discussed in Section 2.4, flow scheduling is repeated on a per-packet basis in a loop where scheduling-related function blocks are executed. The time between two consecutive loop executions depends on the round-trip transmission time of each packet plus the Host-to-FPGA interfacing latency. The interfacing latency between Host and FPGA needs to be low enough to support a packet deadline as low as 1 ms. Meanwhile, *round-trip latency*, which is defined as the elapsed time between a packet arrival at the Host and the ACK reception of the packet, inidcates the minimum achievable per-packet deadline of a wireless platform. To measure interfacing latency and round-trip latency, we devise a simple experiment by using the FPGA counter for the timestamps of the packet events.

The experiment can be summarized as follows:

1. Test packets of fixed payload size arrive at the Host periodically. The period is set to be large enough such that at each time there is only 1 test packet waiting for transmission

in the Host queue. This completely eliminates the effect of queueing delay in the Host.

2. When a test packet arrives at the Host, it is given a timestamp denoted by t_h (read by the Host from an FPGA register) and then forwarded immediately to the FPGA through a DMA Channel.

3. When the FPGA detects the new test packet, the FPGA starts processing the ICP header and retrieves t_h from the header. Along with the current FPGA counter denoted by t_f, the Host-to-FPGA interfacing latency can be derived as $t_f - t_h$.

4. The packet is then transmitted. When the corresponding ACK is received, FPGA reads the current timestamp t_r and calculates the round-trip latency as $t_r - t_h$.

In the experiments, both latency metrics for 50000 packets was measured with a fixed interarrival time of 10 ms. Figure 2.4 shows the empirical cumulative distribution function (CDF) of the Host-to-FPGA interfacing latency for packets with 1500 bytes payload at a data rate of 54Mbps. The mean interfacing latency is around 192 μs, and the 90, 95, and 99th percentiles are 233.4, 257.6, and 316.1 μs, respectively. Table 2.2 further summarizes the statistics of the interfacing latency for different data rates and payload sizes. It can be seen that both the mean and the percentiles of the Host-to-FPGA latency are almost invariant, regardless of data rate and payload size. Therefore, PULSE indeed exhibits low and predictable interfacing latency.

Next, Figure 2.5 shows the empirical CDF of round-trip latency, and Table 2.2 summarizes the statistics of round-trip latency for different data rates and payload sizes. Since round-trip latency consists of both transmission time and Host-to-FPGA interfacing latency, it varies with the physical data rate and the payload size. For the six test cases listed in Table 2.2, the maximum 99th percentile round-trip latency is 933.7 μs. Therefore, PULSE is indeed able to guarantee a round-trip latency of less than 1 ms with high

probability even for large packet sizes and moderate physical data rates.

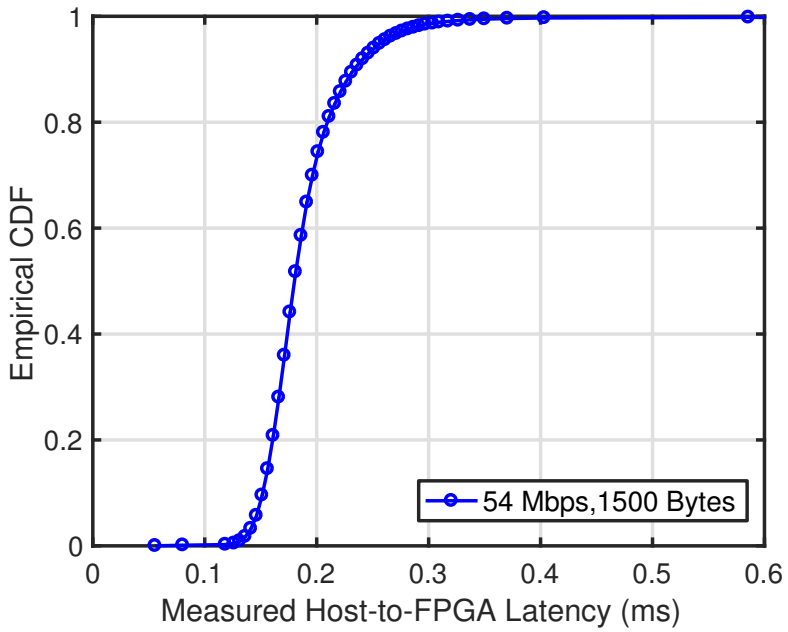

Figure 2.4: Empirical CDF of Host-to-FPGA latency for payload size = 1500 bytes and data rate = 54Mbps.

Table 2.2: Host-to-FPGA latency results

Data rate (Mbps)	Payload size (bytes)	Host-to-FPGA latency (μs)			
		Mean	90%	95%	99%
54	500	185.2	227.6	251.5	301.8
54	1000	189.7	235	259	315
54	1500	192.2	233.4	257.6	316.1
24	500	187.8	228.4	252.7	305.7
24	1000	188.3	230.6	254.3	304
24	1500	189.2	231.5	255	304.6

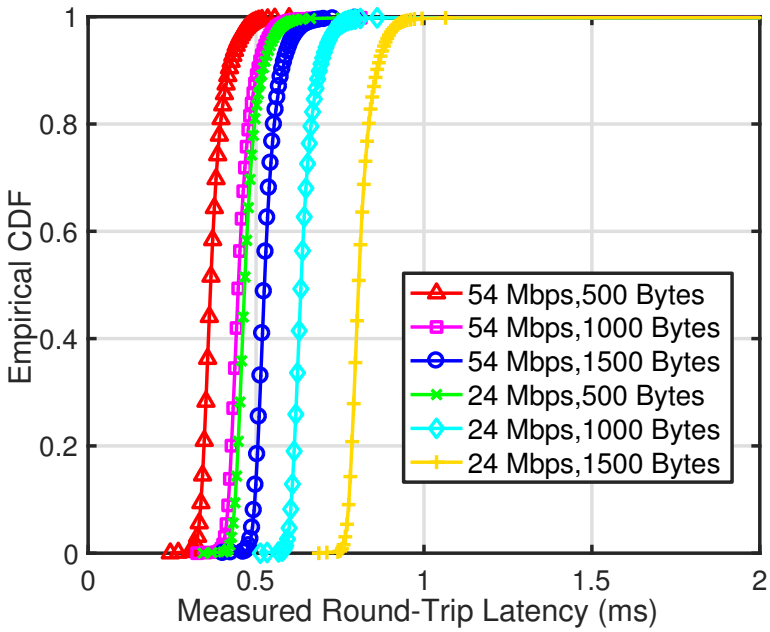

Figure 2.5: Empirical CDFs of round-trip latency for various data rates and payload sizes.

Table 2.3: Round-trip latency results

Data rate (Mbps)	Payload size (bytes)	Round-trip latency (μs)			
		Mean	90%	95%	99%
54	500	377.0	419.2	443.4	494.4
54	1000	459.9	505.1	529.2	585.0
54	1500	536.9	578.5	602.3	660.8
24	500	479.5	520.2	544.4	598.1
24	1000	646.6	688.9	712.7	762.8
24	1500	817.9	860.2	883.9	933.7

As further verification of the system, we also plot the received throughput of clients against packet deadlines. This is to ensure that packets are not transmitted when they have expired. For this test, 20 packets arrive deterministically on the AP every 11 ms. (Note, packets are only expired when the current time exceeds the deadlines. Hence even 0 ms deadlines give some throughput.) As can be seen from Figure 2.6, the throughput increases

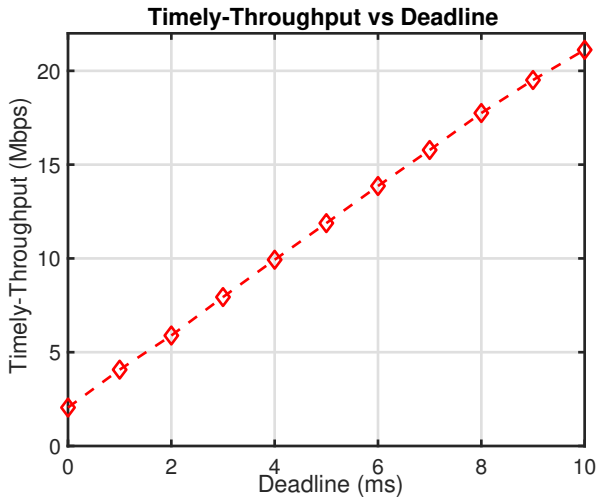

Figure 2.6: Timely-throughput vs. Deadline for 20 packets generated every 11 ms.

linearly as deadline increases, which shows that expired packets are indeed getting dropped by the AP.

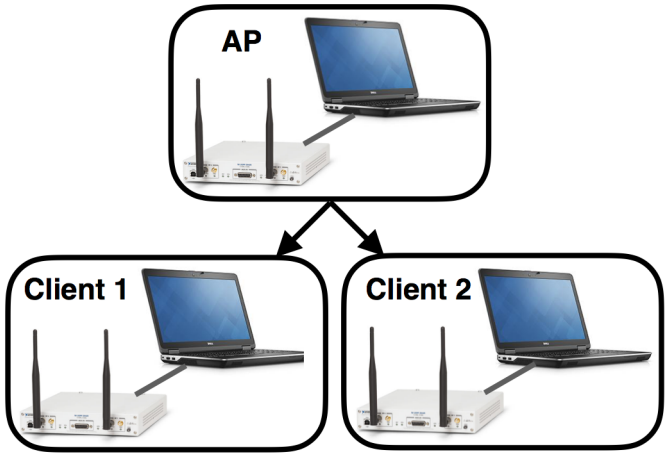

Figure 2.7: Experimental setup with host machines and USRP-2153R's being used as wireless AP and clients.

2.6 Experimental Results

In this section, I present experimental results for a network with one AP and two downlink clients (as shown in Figure 2.7) in various scenarios. Each client is associated with one real-time flow with per-packet deadlines as well as one non-real-time flow without deadline constraints. Each of the nodes (AP and clients) is a USRP-2153R [48] that is connected to a Windows laptop acting as the Host machine. A specific scenario will have a certain arrival process, as well as predefined requirements for deadlines and delivery ratios for the real-time flows. These experiments were run using 1500B packets and IEEE 802.11a MCS 7 (54Mbps theoretical link data rate). We consider four scheduling policies: Largest Deficit First (LDF) [31], Longest Queue First (LQF), Round Robin (RR), and Random. Based on the definition of deficit introduced in [31], LDF schedules the real-time flows with the largest deficit and selects non-real-time flows with the largest queue length if the real-time flows are empty. Ties are broken randomly. LQF, as the name suggests, schedules the flow with the longest queue, with ties broken randomly. The Random policy randomly picks a flow to schedule from among non-empty queues. RR schedules flows in the following order: real-time flow for client 1, real-time flow for client 2, non-real-time flow for client 1, and then non-real-time flow for client 2. If any of the queues are empty, it schedules the next queue. A point to note is that dropping expired packets is not done in most implementations today. To ensure a fair comparison, we decided to enable packet dropping for all policies, which improves the performance for all policies.

The results are presented in this section as follows. In Section 2.6.1, I present the capacity regions of a single client under the different policies for one scenario to show the achievable regions of our system. Then, in Section 2.6.2, the throughputs of the policies under different scenarios is shown to compare the system performance. In these sections, packets are generated every 5 ms and the number of packets generated is uniformly dis-

tributed from 0 to K_{RT} for real-time flows, and K_{NRT} for the non-real-time flows. A different arrival process is used in Section 2.6.3 to have packets arrive more frequently, to show that ultra low latencies with 1 ms deadlines are indeed achievable.

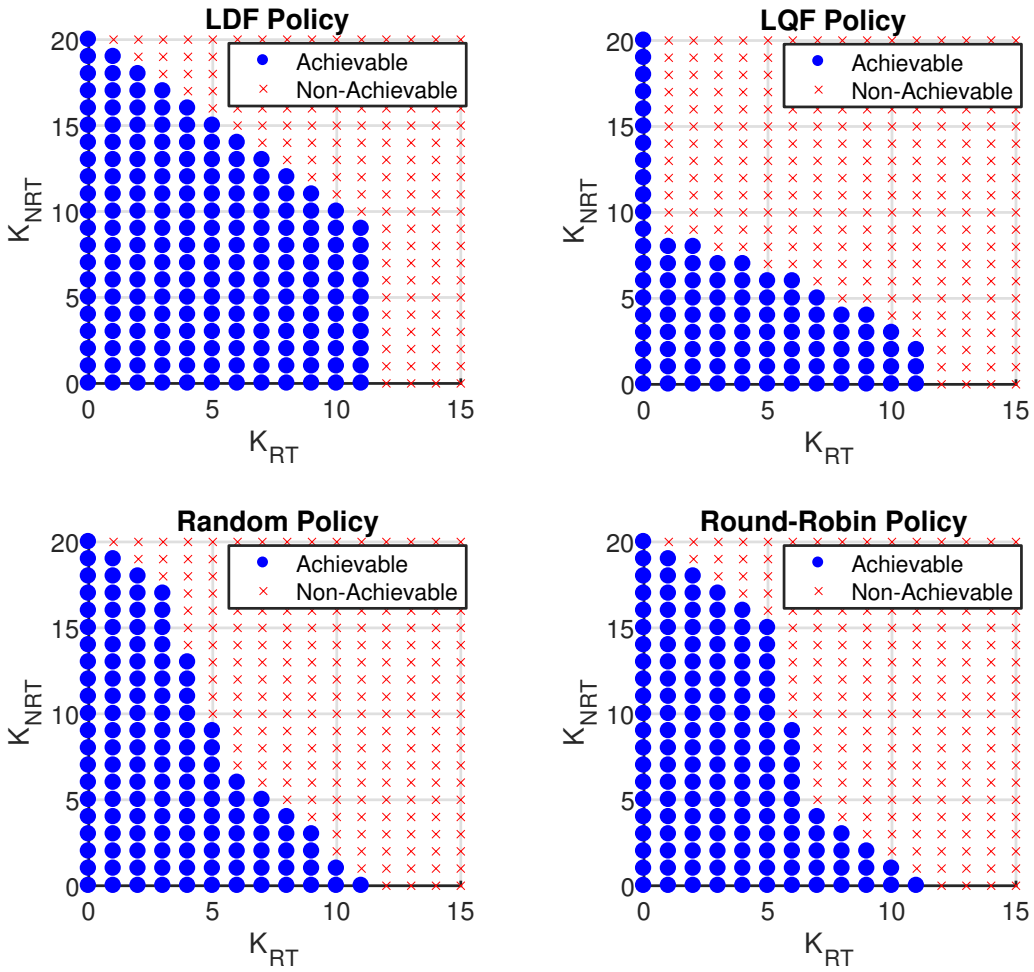

Figure 2.8: Capacity regions for the different Policies with 5 ms deadline and 0.99 delivery ratio.

2.6.1 Capacity Regions

For our capacity region experiments, we defined achievable regions as regions where the system deficits (accumulated when packets are dropped) and queue lengths of the clients do not grow to infinity. Per-packet deadline is set to 5 ms and the delivery ratio is set to 0.99, which means that the deficit increases by 0.99 every time a packet is dropped, and decreases by 0.01 when a packet is successfully delivered. In other words, if we require 99% of real-time packets to arrive in 5 ms, we say the policy can achieve that for any incoming packet rate where the deficits do not grow to infinity. As we can see from Figure 2.8, the LDF policy has a bigger achievable region than LQF, Random and RR. Random and RR have similar capacity regions, but Random performs slightly better than RR when K_{RT} is higher, and worse when K_{RT} is smaller. This is because the amount of time waiting for service is bounded for RR, so for a small number of real-time packets, they will always be served before the packets expire. On the other hand, when K_{RT} is high, RR will not be able to serve the packets in time, but Random has a chance of serving the packets before they expire. LQF works better for higher values of K_{RT} than Random and RR, since it would schedule real-time flows more frequently, but suffers significantly when the value of K_{NRT} is higher than K_{RT}, since non-real-time flows will be scheduled first. LDF always schedules real-time flows first, so the drop off in the capacity region is linear until K_{RT} = 11, after which the deficits start increasing to infinity for this delivery ratio and deadline. In this scenario, all policies can support up to K_{RT} = 11, and serve up to 20 packets every 5 ms.

2.6.2 Throughput Performance

Using the capacity region plots, we can know what arrival rates our platform is capable of supporting. However, this does not tell us much about how system performance is affected in terms of the throughputs for real-time and non-real-time flows when operating

in different scenarios. For brevity, in this section, throughputs for real-time flows will be referred to as timely-throughput, and throughputs for non-real-time flows will just be referred to as throughputs. So, we next ran experiments for multiple clients, each with a real-time and non-real-time flow, under various scenarios to see how each protocol performed. We ran two symmetric scenarios (clients have the same traffic and requirements), and two asymmetric scenarios. In the first scenario, we had K_{RT} = 4 and K_{NRT} = 4 for both clients, with deadlines set to 3 ms and delivery ratio set to 0.98. As we can see in Figure 2.9, while most protocols perform relatively well, LDF has a higher timely throughput and overall throughput for both clients. Next, we increased non-real-time traffic (K_{NRT} = 6) and decreased real-time traffic (K_{RT} = 3), and changed the requirements of the real-time flows to 2 ms deadlines and 0.95 delivery ratios to see how the system performs with lower latencies. This could be an example of two clients downloading a large file while streaming videos. As we can see in Figure 2.10, LQF has the worst performance since it tends to serve non-real-time traffic first, followed by Random and RR. LDF has the best performance in both these symmetrical scenarios.

For the first asymmetrical scenario, we set K_{RT} = 3 and K_{NRT} = 5 for client 1, K_{RT} = 4 and K_{NRT} = 6 for client 2. For client 1, the deadline of real-time packets was set to 2 ms, with 0.97 delivery ratio, whereas client 2 has a deadline of 3 ms with 0.98 delivery ratio i.e. client 1 has a real-time flow requirement where 97% of packets have to arrive in 2 ms and client 2 has a real-time flow requirement where 98% of packets arrive in 3 ms. As we can see in Figure 2.11, LDF outperforms the other policies in terms of timely throughput and overall throughput. This difference is even more apparent when the asymmetry between the requirements of both clients is increased. In our second asymmetrical scenario, we set K_{RT} = 7 and K_{NRT} = 4 for client 1, K_{RT} = 3 and K_{NRT} = 5 for client 2. Client 1's deadline was set to 5 ms with 0.98 delivery ratio, and client 2's deadline was 2 ms with 0.99 delivery ratio. This could be a case where client 1 is streaming a video where

5 ms latency is tolerable but it generates a lot of traffic, and client 2 is running a control application that does not produce as many packets but has stricter latency and delivery ratio requirements. We see that Random and RR do not provide a good timely throughput for client 1, while LQF does not perform well for client 2. LDF outperforms all policies in both clients since it smartly schedules client 1 and client 2's real-time flows so that both requirements are met, without sacrificing overall system throughput. Note that since we are operating on the boundaries of LDF, the deficits of LQF, Random and RR are growing to infinity as well. The deficit evolution from the first symmetrical scenario is shown in Figure 2.14.

Even though the throughputs do not differ by much, the picture becomes very different when the number of real-time packets that were dropped is examined. Using the deficits of the clients, the loss ratios of the real-time flows for all the scenarios above were calculated. (Since the deficit of LDF stays at 0, the most we can conclude is the loss ratio is below 1 - delivery ratio). As can be seen in Table 2.4, the loss ratios of LQF, Random and RR are much higher than LDF. The biggest difference is for the fourth scenario, when LDF can maintain less than 1% loss rate for Client 2, but LDF, Rand and RR experience loss rates of 63.5%, 29.5% and 21.4% respectively.

2.6.3 Changing the Arrival Process

In our final experiment, we modified the arrival process to further evaluate the policies under more stringent deadline requirements. Instead of packets being generated every 5 ms, packets can now be generated every 1 ms. However, instead of generating them uniformly from 0 to K_{RT} (or K_{NRT}), a single packet is generated with some probability, which is specified for each flow by the client. Let P(K_{RTi}) and P(K_{NRTi}) be the probability of generating a packet for client i's real-time flow and non-real-time flow respectively. As in the previous experiment, the contrasts are most stark in the asymmetrical case, which is

Table 2.4: Loss ratios of real-time flows

Arrival Rate	Deadline (ms)		Delivery Ratio		Policy	Loss Ratio (%)	
	C1	C2	C1	C2		C1	C2
K_{RT1} = 4 K_{NRT1} = 4 K_{RT2} = 4 K_{NRT2} = 4	3	3	0.98	0.98	LDF	<2	<2
					LQF	23.51	17.12
					Rand	15.17	15.62
					RR	15.07	16.03
K_{RT1} = 3 K_{NRT1} = 6 K_{RT2} = 3 K_{NRT2} = 6	2	2	0.95	0.95	LDF	<5	<5
					LQF	60.24	54.97
					Rand	26.48	27.22
					RR	19.96	21.27
K_{RT1} = 3 K_{NRT1} = 5 K_{RT2} = 4 K_{NRT2} = 6	2	3	0.97	0.98	LDF	<3	<2
					LQF	61.45	27.35
					Rand	27.2	18.07
					RR	20.09	17.13
K_{RT1} = 7 K_{NRT1} = 4 K_{RT2} = 3 K_{NRT2} = 5	5	2	0.98	0.99	LDF	<2	<1
					LQF	7.18	63.51
					Rand	17.01	29.46
					RR	19.14	21.42

presented in the following scenario. Client 1 has P(K_{RT1}) = 0.8, P(K_{NRT2}) = 0.3 with the real-time flow requirement of 2 ms and 0.99 delivery ratio. This could be a user streaming a video while downloading some files in the background. Client 2 on the other hand has P(K_{RT2}) = 0.1, P(K_{NRT2}) = 0.5 with a deadline of 0 ms and 0.99 delivery ratio. Note that packets are only expired when the current time exceeds the deadlines, and hence even 0 ms deadlines give some throughput. This could be a mission-critical application which does not generate much traffic but requires control packets to be delivered within 1 ms. As we can see in Figure 2.13, client 1's timely throughput is lower for Random and RR, while LQF causes client 2's timely throughput to suffer. However, LDF is able to support both clients and give good timely throughput and throughputs as well. Furthermore, this also shows that the system is capable of delivering packets under 1 ms.

These results show that our system is indeed capable of supporting experimentation of policies for ultra-low latency applications with various packet arrival patterns and deadline requirements.

2.7 Conclusion

Applications today have increasingly stringent requirements, especially in terms of latency and throughput. This presents next generation networks with one of their critical challenges: providing some measure of guarantee for applications with strict latency and throughput requirements. There exist theoretical frameworks to develop protocols that are able to do this, but there is still a gap between theory and implementation of these protocols. We aim to bridge this gap by developing PULSE, which I have shown to be capable of supporting per-packet scheduling for downlink with latencies on the order of 1 ms, with realistic system throughputs. PULSE was developed with reprogrammability in mind, so new scheduling policies can be more easily implemented and experimented on it. Using PULSE, the performance of LDF, LQF, Random and RR was tested under various scenarios and showed that LDF performs equally well or better than the other policies in all scenarios. The difference between LDF and the other policies is even more apparent when the loss ratios of the policies under different traffic loads is observed.

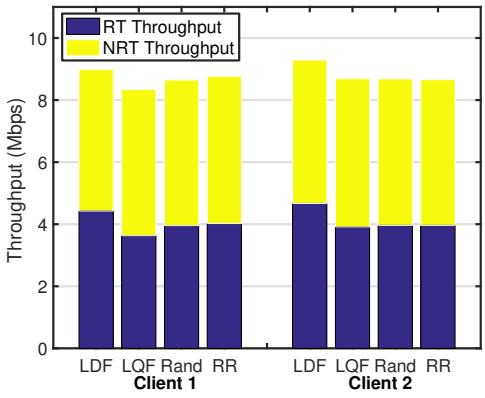

Figure 2.9: Throughputs for K_{RT}=4 and K_{NRT}=4.

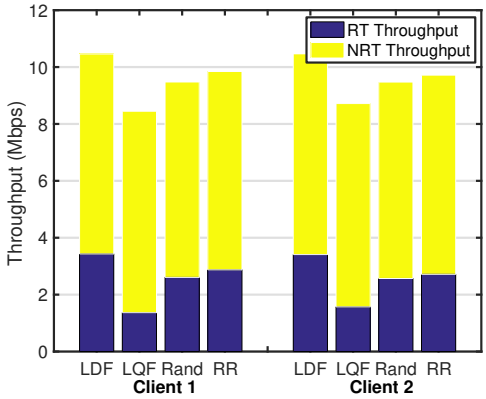

Figure 2.10: Throughputs for K_{RT}=3 and K_{NRT}=6.

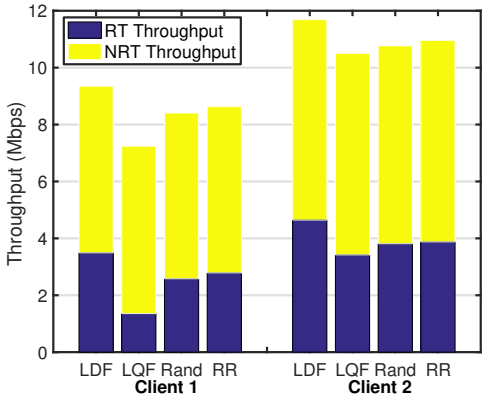

Figure 2.11: Throughputs for K_{RT}=3, K_{NRT}=5 for client 1, K_{RT1}=4, K_{NRT}=6 for client 2.

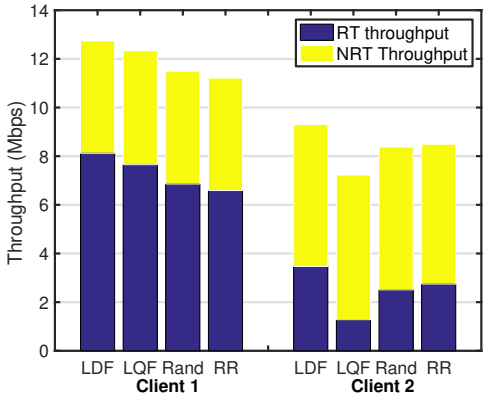

Figure 2.12: Throughputs for K_{RT}=7, K_{NRT}=4 for client 1, K_{RT1}=3, K_{NRT}=5 for client 2.

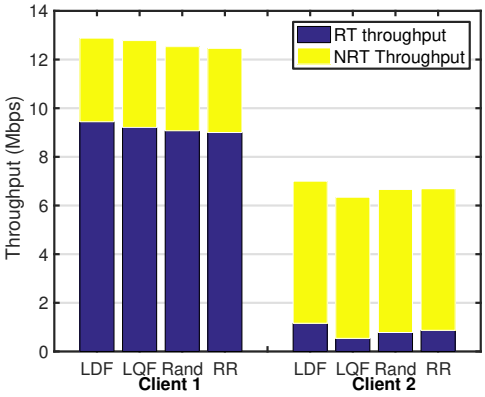

Figure 2.13: Throughputs for the second arrival process with P(K_{RT1}) = 0.8, P(K_{NRT1}) = 0.3, P(K_{RT2}) = 0.1, P(K_{NRT2}) = 0.5.

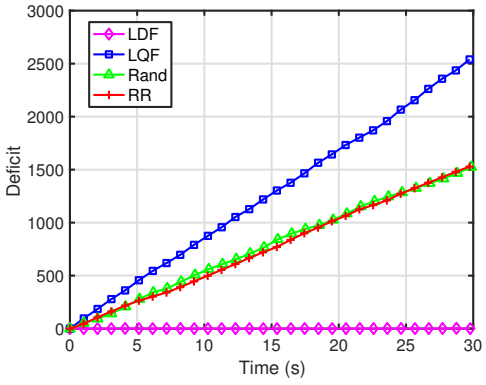

Figure 2.14: Deficit growth for the K_{RT}=4 and K_{NRT}=4 for both clients.

3. WIMAC: PLATFORM FOR RAPID IMPLEMENTATION OF MEDIUM ACCESS CONTROL PROTOCOLS

3.1 Introduction

In the previous chapter, we discussed how we can implement and experiment with scheduling algorithms for next-generation wireless networks. In this chapter, I will show how the performance of these kinds of protocols can be further improved by providing an architecture that allows packet-by-packet scheduling, while also reducing the prototyping time for next-generation MAC protocols. The traffic served by WiFi has been increasing exponentially due to the rapid development of emerging applications, such as multimedia streaming. Moreover, wireless standards organizations are planning to offload more cellular data traffic onto unlicensed bands to relieve the increasingly dense cellular networks. For example, 3rd Generation Partnership Project (3GPP) group has been developing License-Assisted Access (LAA) technology for coexistence of WiFi and LTE, which is expected to be a feature in Release 13 of LTE [49]. Commensurately, wireless networks need to keep evolving to face the challenge of providing higher throughput as well as other critical performance metrics, such as low latency and power saving. A wide variety of wireless protocols have therefore been proposed to meet these requirements. To accelerate the testing and deployment of new innovations, rapid and flexible experimentation of new protocols is becoming increasingly critical for wireless research.

Wireless protocols for different layers of the protocol stack often follow different strategies vis-à-vis experimentation. For PHY layer protocols, it is of utmost importance to meet the stringent timing requirements of high-throughput wireless protocols. Based on hardware design with dedicated FPGAs or digital signal processors, PHY layer experimentation can be conducted to establish the realistic performance of time-critical wireless

protocols with respect to their specifications due to sufficient hardware design flexibility. On the other hand, the network layer is often required to handle complex network topologies and make appropriate routing decisions. Therefore, experimentation of network layer protocols is mostly done through abstraction in the software domain for higher programmability. For example, OpenFlow [50] enables experimentation of network layer protocols by programming the flow tables in Ethernet switches and routers, so that network layer protocols can be tested completely via software. Hence, through extensive abstraction provided in the software domain, the barriers to deploying network layer protocols can be greatly reduced.

Differing from the physical layer and networking layer, the implementation of MAC protocols presents its own challenges. Since the MAC layer serves as the junction between the PHY layer and other high level layers, it is required to interface with the PHY layer closely and smoothly. Therefore, MAC designers are often challenged by the co-design of MAC and PHY which often requires substantial efforts across disciplines. As a consequence, very few of the newly proposed MAC protocols have been experimentally tested or implemented [9, 10, 11, 12, 13]. Furthermore, these rare exceptions are either designed specifically for wireless sensor networks which target low-power and low-rate applications, or are limited to unrealistically low throughput performance. For commodity products, MAC functionalities are often integrated with the PHY components on a single network interface card, which makes them almost impossible to modify. As a result, new wireless MAC protocols often need to be implemented from scratch and thus require much longer time for experimental evaluation.

To reduce the prototyping time for MAC protocols, software-defined Radio (SDR) is a promising solution with the required flexibility. In general, SDRs consist of three major parts: RF front end, baseband hardware, and software host. Typically, hardware development takes longer than software, and this is also true for SDRs. Therefore, it is

preferable to perform as many functions as possible in the software domain for maximum programmability. However, heavy signal exchange between the software and hardware can introduce substantial latency (usually around hundreds of microseconds) and thus degrade the overall throughput. Hence, the flexibility-latency tradeoff dilemma is one major challenge for modern high-throughput MAC implementation.

With the aim of enabling quick prototyping as well as achieving flexibility and low latency, this chapter presents a platform for rapid implementation of wireless MAC protocols for WiFi. The main design philosophy of this platform, called WiMAC, is to decouple the specifications of MAC protocols from the underlying hardware. WiMAC identifies distinguishing features of classes of MAC protocols, and implements these features as separate functional blocks in the hardware. In this way, WiMAC achieves high function reusability in the hardware domain and meets the strict latency requirement of MAC protocols. Second, WiMAC specifies the interaction between these functional blocks completely through software. Therefore, WiMAC offers a flexible way for MAC configuration in software, and an efficient way to stride through MAC design iterations without making any changes in hardware domain. To show the practicality of WiMAC, I present design examples of a broad range of MAC protocols, ranging from Carrier Sensing-based protocols, such as Carrier Sense Multiple Access (CSMA) [51], and coordination-based protocols, like CHAIN [52], to queue length based algorithms, such as Max-Weight [53]. While these protocols are very diverse, WiMAC does make the experimentation easily performable through the proposed decoupling framework. As an additional illustration, we implement an example of a meta protocol, which allows developers to switch between different classes of MAC protocols during runtime with hardware unchanged. In all cases, WiMAC makes feasible rapid implementation, and preserves both design flexibility and realistic throughput performance.

3.2 Related Work

In commercial wireless network interface cards, MAC layer functionality needs to be configured by driver suites that are specific to individual chipsets, such as the *ath* family for Atheros chipsets and the *rt2x00* family for Ralink chipsets. These drivers can only control very limited functions and have no access to the fundamental MAC functions, such as packet format and frame timing. To enable customized MAC on commodity hardware, Neufeld *et al.* propose SoftMAC [54], a software platform built on an Atheros chipset and the corresponding open-source software. By overriding key features of the Institute of Electrical and Electronics Engineers (IEEE) 802.11 protocols [44], SoftMAC offers flexibility to reconfigure important functions such as header formats and backoff schemes. Other platforms like MadMAC [55] and FlexMAC [56] follow similar design strategies to implement customized MAC protocols on commercial 802.11 hardware. However, built on commodity hardware, these platforms suffer from limited scope of redefinable functions and fail to support protocols outside the CSMA-based category. Following a different approach, Doerr *et al.* [57] propose MultiMAC to enable switching between multiple MAC protocols for better performance in a changing environment. Despite the protocol-level adaptability, this platform does not offer the required flexibility in MAC implementation.

To achieve the required flexibility for MAC implementation, a variety of wireless platforms have further leveraged the software radio paradigm. For example, the GNU's Not Unix (GNU) Radio [58] software toolkit offers a flexible environment for protocol implementation. With minimal hardware design, GNU Radio shifts most of the implementation efforts to the software domain. However, this software-based platform is limited to narrow-band protocols due to severe latency, as discussed in [59]. To resolve the latency issue, *Sora* [60] exploits advanced parallel computation on multi-core processors to achieve

throughput comparable to the current-generation Wireless Local Area Network (WLAN) protocols. However, the complexity in the software domain makes MAC implementation difficult for MAC designers. In the family of FPGA-based software radio platforms, Wireless Open Access Research Platform (WARP) [61] is a popular high-performance hardware platform for MAC and PHY research. By using FPGA and PowerPC core, WARP allows full access to both MAC and PHY functions and thus achieves high flexibility. Similar to WARP, Airblue [62] is an FPGA-based cross-layer design platform. By implementing both the MAC and PHY on the FPGA, Airblue is able to run at a speed comparable to commodity IEEE 802.11 hardware. However, these FPGA-based platforms focus primarily on flexibility and latency performance, and therefore do not guarantee quick prototyping of MAC protocols.

To achieve quick prototyping of user-definable MAC protocols, various architectures for SDR have been explored with the aim of realizing fast MAC composition. Nychis *et al.* [63] propose a split functionality architecture, which achieves better code reuse by identifying a core set of time-critical MAC functions, such as backoff, carrier sensing, and precise scheduling in time. These time-critical functions are located as close to the radio hardware as possible to reduce bus latency between the host and radio front end. This architecture is implemented on the GNU Radio and Universal Software Radio Peripheral (USRP) hardware to highlight the latency improvement. Similarly, Ansari *et al.* [64] introduce Decomposable MAC Framework to identify basic functional components according to both timeliness and degree of code reuse. Built on this component-based framework, Zhang *et al.* [65] propose TRUMP, a toolchain that enables runtime composition of MAC protocols. Modelled by the underlying state machines, MAC protocols are described in a meta language and synthesized by a wiring engine from the basic building blocks. However, when new features are added to the system, extensive changes have to be made to the system, such as modifying the compiler and the dependency tables. Furthermore, they do

not show how new functions are added to the system or how different classes of MAC protocols, such as power-saving MAC protocols, can be implemented using TRUMP. In [66] Tinirello *et al.* present Wireless MAC Processor (WMP), which is built specifically on commodity WLAN cards. In addition to modifying the firmware or software of wireless cards, WMP goes one step further to define a set of MAC commands as well as triggering events and conditions, which is similar to the instruction set of a Central Processing Unit (CPU). Similar to [65], WMP reads the MAC state machines written in machine language, and then realizes MAC composition with an execution engine. However, WMP only supports a limited set of MAC protocols since it is difficult to include the required new MAC functionality outside the inherent command set on commodity hardware, such as selecting backoff times based on different probability functions.

3.3 Key Design Principles for MAC Platform

The main goal is to enable rapid implementation of a wide range of MAC protocols to determine the realistic performance of these protocols. In order to do so, we identify key design principles that would help in achieving the desired goal and incorporate them into the design of WiMAC.

- **Focus on "Next Gen" protocols.** Increasingly more applications have requirements beyond maximizing throughput while maintaining fairness. Examples of goals of these protocols include maximizing a network utility function [67], energy savings [68], reliable communication [69], maintaining some level of quality of service (QoS) [70], or even a combination of any of these. These "Next Gen" protocols, require support of very different kinds of functions. The platform should be designed to easily incorporate the features required by these protocols.

- **MAC vs PHY decoupling.** In many cases, the MAC protocol is very tightly coupled with the PHY layer. This is due to the fact that MAC protocols are required

to react to the PHY layer with very stringent timeliness. Changes made in either layer usually necessitate a change in the other layer, with very few exceptions. This makes it extremely challenging to develop MAC protocols that differ significantly from the existing protocols. To ensure the platform is capable of implementing MAC protocols rapidly, we have to ensure that changes in either layer can be made independently of the other.

- **Designing MAC protocols in software.** It is generally easier to develop algorithms in software than it is in hardware. MAC protocols are no different. Being able to design MAC protocols in software will allow for more flexible algorithms. This will enable implementation of a larger class of algorithms. Additionally, designing MAC protocols in software will reduce the time taken to implement these protocols. However, not everything can be done in software, since we still have to meet the quick response time required by MAC protocols.

- **Hardware vs Software decoupling.** Because a wide range of features of MAC protocols require hardware support, a platform that supports different classes of MAC protocols requires a mixture of hardware and software components. On the other hand, we want the ability to develop components in these domains independently from each other; changes in software should not require a change in hardware, or vice versa. Hence, hardware and software components need to be decoupled from each other.

- **Allows cross-layer design.** Cross-layer designs have the potential of increasing the performance of a given system. There have been some proposals for cross-layer MAC protocols such as [71, 72, 73, 74]. Experimentation with such protocols requires the platform to have the capability of supporting cross-layer design. Even though the MAC and PHY layer are decoupled in the design of the platform, it is

flexible enough to allow exchange of information between both layers if so desired.

- **Supports protocol changes at runtime.** This feature increases the range of protocols the platform will be able to support, since dynamic MAC protocols can also be implemented on the platform. Furthermore, it aids comparative experimentation since we can switch between different MAC protocols to observe the performance of each protocol.

3.4 Architectural Design of The Platform

This section provides details on the implementation of the platform, showing how distinguishing features can be implemented generically to achieve decoupling and flexibility in design of MAC protocols while meeting latency requirements of a MAC protocol.

3.4.1 Identifying Distinguishing Features

The first WiMAC task is to identify distinguishing features of different classes of protocols. As one example, one such distinguishing feature is clearly Carrier Sensing. Once Carrier Sense is implemented, it opens up a whole class of protocols that researchers can experiment with, such as CSMA. By identifying such distinguishing features of different classes of protocols, one can quickly obtain an initial set of functions that will allow rapid implementation of a wide range of protocols. As a starting point, we decided to implement the following distinguishing features of different classes of MAC protocols. Other features can be implemented too, as shown later.

- **Carrier Sense:** Determines whether the channel is clear.

- **Power Control:** Tells the node to increase or decrease transmit power, and by how much. Changing transmit power affects the block error rate of transmissions, the interference levels of neighboring nodes, and the network topology.

- **Piggyback Transmissions:** Instead of a node initiating data packet transmissions by itself, piggyback transmissions allow nodes to listen to their surroundings and transmit data packets immediately after a certain packet is heard.

- **Queue Length Information:** Knowing the queue length information can provide us with information about the delay and network utilization.

More details are given in the next section when I describe specific protocols that have been implemented.

3.4.2 Designing MAC Protocols in Software and Decoupling

The second task was to figure out how to overcome the main challenge: how to design MAC protocols easily in software while still achieving hardware performance. Being able to implement every MAC function in software provides the flexibility to implement a wide range of protocols very easily. However, due to the latency between the hardware and software interfaces, this is not possible. MAC functions have to be implemented in hardware in order to meet the timing requirements of a MAC. This makes it more difficult for rapid implementation of MAC protocols, since developing protocols on hardware takes longer than developing in software. Furthermore, hardware designs are typically less agile than software designs.

To achieve the goal of rapid implementation while still meeting the timing requirements, we therefore decided to design WiMAC in the following way:

1. Each MAC function in hardware has controls that allow users to specify parameters that will change how the block functions. For example, in the Backoff block, users can specify how to increment CW, the size of the contention window. The Backoff block then chooses a random number between [0, CW] to pick as its next backoff count. While this provides some flexibility to the protocol, it does not increase the

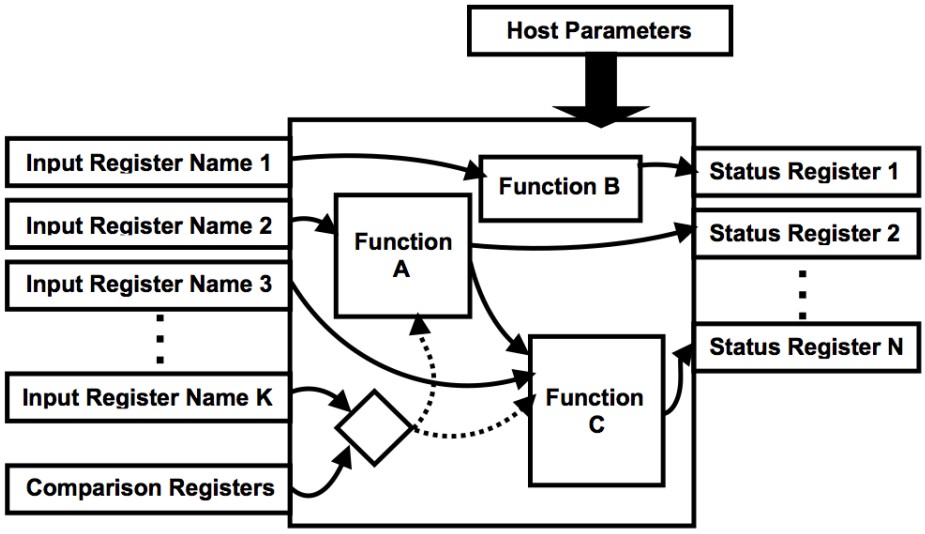

Figure 3.1: Example of a generic function block in WiMAC.

range of protocols the platform is capable of supporting.

2. In addition to controls, we also add three types of registers to each of the blocks. The first type of registers is Status Registers. They allow us to obtain the output of the block for it to interact with other blocks in the MAC. The second type of registers are Input Register Names. These registers tell the blocks what registers to use as inputs to the block. Finally, the third type of registers are Comparison Value Registers. They specify how and what values to compare the input values with.

3. Finally, users specify the Input Register Names that each block should use as inputs, and the comparison values for each of these inputs on the host machine.

In this three pronged way, we are able to achieve the goal of having a flexible platform and still meet the strict timing requirements of a MAC. As an example, suppose we want to specify "Enable piggyback transmissions if queue length is more than 10". Then on the host machine, we set the status register of the Queue length block as the input register

of the Piggyback Transmission block, and the comparison value register to be "more than 10". Now, when the output of Queue Length exceeds 10, the Piggyback Transmission will be enabled.

While significant changes in the hardware will always necessitate changes in the software, we can alleviate the changes that need to be made by fixing the names of the hardware blocks, the number of input and output registers for each block, and the output register names and types for all hardware blocks. By doing so, the same software code can still be used even when hardware changes are made. An example of a function block is shown in Figure 3.1. The number of input registers can vary from block to block, but once the block has been implemented, we do not change the number of inputs. The same applies to the outputs, except we require that output register names and types be the same as well.

3.4.3 Enabling Reconfiguration on the Fly

With the above architecture, it becomes possible to make protocol changes on the fly. To take it one step further, any valid node should be able to change other nodes. We do not specify what the requirements are for a valid node. (This has no security implications since this is a platform for experimentation). For the purposes of this chapter, let us assume all nodes can change every node. This is done in two steps: 1) Creating a reconfiguration packet, and 2) developing a protocol to transmit reconfiguration packets. The reconfiguration packet has the following fields.

- **Packet Type.** This is to specify the type of packet is a reconfiguration packet.
- **Block Name.** Specifies the hardware block to be configured.
- **Input Register Name.** Name of the register to be used as one of the input registers of the block given by Block Name.
- **Comparison Value.** Value to compare the input register values with.

Packet Type	Block Name 1	Input Register 1	Comparison Value 1
Input Register 2	Comparison Value 2	.....	
Input Register K	Comparison Value K	Block Name 2	Input Register 1
Comparison Value 1	Input Register 2	Comparison Value 2	.....
.....		Input Register L	Comparison Value L
.....			
Control Name 1	Control Value 1	Control Name 2	Control Value 2

Figure 3.2: Example of reconfiguration packet.

- **Control Name.** Specifies the name of the control parameter.
- **Control Value.** Changes the value of the control to this Control Value.

Packets start with a Packet Type, followed by a Block Name. For each Block Name, there can be any number of Input Register Names, but each Input Register Name has to be followed by a Comparison Value. Three bits are used in Comparison Value to indicate whether to check for equality, greater than, less than, or do not care. This is followed by however many bits required for the data type. If we do not care about the value, the comparison value register is filled up with the number of zeros corresponding to the type of data in the Input Register. Then come the Control Names and Control Values. Each Control Name has to be followed by a Control Value. Multiple Block Names can be included in the packet. An example of a reconfiguration packet is shown in Figure 3.2.

To transmit these packets, we use a simple protocol. A designated node selects other nodes it wants to transmit reconfiguration packets to. Then, starting with the lowest numbered node, it transmits the packet after Point Coordination Function (PCF) Interframe Space (PIFS) time [44] to avoid collisions with data packets or ACK packets. If it receives an ACK, it moves on to the next node and repeats this process. If it does not receive an

ACK, it retransmits up to a designated number of times. If it still does not receive an ACK after all the retransmissions, it just moves on to the next node. Thus, it is possible for some nodes to remain unconfigured. Unconfigured nodes keep running the previous MAC protocol that they were configured with. The AP then informs the user which nodes successfully received the reconfiguration packet.

Being able to reconfigure packets on the fly provides hardware/software decoupling, and a mechanism to perform comparative tests between different MAC protocols more readily.

3.5 Implementation of Different Classes of MAC Protocols

In this section, I will show how WiMAC can be used to implement completely different classes of MAC protocols. This section is structured as follows. First, the performance of a link with different coding and modulation rates is presented to show that this platform is capable of supporting realistic bit rates. Then, I present the results of the platform implementing CSMA, since it is the most widely used MAC protocol. After that, the implementation of CHAIN [52], is presented as an exemplar of a protocol that uses piggyback transmissions to increase the uplink efficiency. Next, the implementation of Max-Weight, a centralized control algorithm that has been designed for theoretically achieving optimal throughput [53] is presented. Finally, I demonstrate how MAC protocols can be easily changed on the fly on WiMAC through an example of meta protocol.

3.5.1 Link Performance

For the experiments, we used NI USRP 2153R, which is an FPGA-based SDR by National Instruments. The experiments were run at the center frequency of 2.437GHz. Subcarrier format follows the IEEE 802.11a 20MHz standards. To reduce interference from external sources, set the RX gain of each of the nodes was manually set to be 19 dB, and the transmit power to be 10 dBm, with the exception of the meta protocol case. The

performance of a single link is summarized in Table 3.1. For implementation of protocols, the modulation and coding scheme (MCS) was set to 64QAM with 3/4 coding rate.

Table 3.1: Link throughput for different modulation schemes and coding rates

Modulation	Coding Rate	Link Throughput (Mbps)
BPSK	1/2	5.7
	3/4	8.4
QPSK	1/2	11.0
	3/4	16.0
16QAM	1/2	20.7
	3/4	29.5
64QAM	2/3	37.1
	3/4	41.2

3.5.2 CSMA

CSMA [51] is known to be one of, if not, the most widely used wireless MAC protocol nowadays. In CSMA, medium access is achieved by employing the "listen before talk" mechanism. Before transmitting, each wireless node senses the channel to ensure it is clear. If the channel is clear, and the node has a packet to transmit, the node starts decrementing its backoff counter, which is a random number chosen from [0, CW]. The backoff counter stops if the channel becomes occupied either by other wireless nodes, or by external noise. Once the channel is sensed to be clear again, the backoff counter resumes. When the counter reaches 0, the node starts its transmission. CSMA is typically used with Distributed Coordination Function [44]. In DCF, CW is doubled when a node transmits but does not receive an ACK, up to CWmax. When it receives an ACK (after a successful transmission) CW is set to CWmin. Since CSMA is a contention-based protocol, the efficiency of CSMA decreases as more nodes enter the network. The results of CSMA are shown in comparison with CHAIN in Figure 3.3. When CWmin is small,

collisions happen frequently, and thus throughput drops as nodes enter the system. This is the case when CWmin = 4. As CWmin increases, collisions occur less frequently, but the system spends a lot of time idling when there are few nodes in the system. The waste in network resources lead to lower throughputs as shown in Figure 3.3.

3.5.3 CHAIN

To increase the uplink efficiency of a network, CHAIN [52] proposes a mechanism called piggyback transmission. As mentioned in the previous section, this feature allows users to transmit immediately after a certain packet is heard. Specifically, in CHAIN, piggyback transmissions occur after a node decodes an ACK sent to its predecessor. When this happens, the node transmits a data packet after SIFS time. This reduces the amount of time spent on contending for the channel, thus increasing uplink efficiency. When this event does not occur, CHAIN operates just like CSMA with DCF, making it fully compatible with DCF.

An example of this exchange is shown in Figure 3.4. In this set up, three clients C1, C2, and C3 are connected to an AP, with C1 preceding C2, and C2 preceding C3.

- (e1) C1 sends a data packet to the AP.
- (e2) AP responds by sending an ACK to C1. At the same time, C2 overhears the ACK for C1.
- (e3) Since C1 precedes C2, C2 sends a data packet immediately after a SIFS period.
- (e4) AP responds by sending an ACK to C2. At the same time, C3 overhears the ACK for C2.
- (e5) Since C2 precedes C3, C3 sends a data packet immediately after a SIFS period.

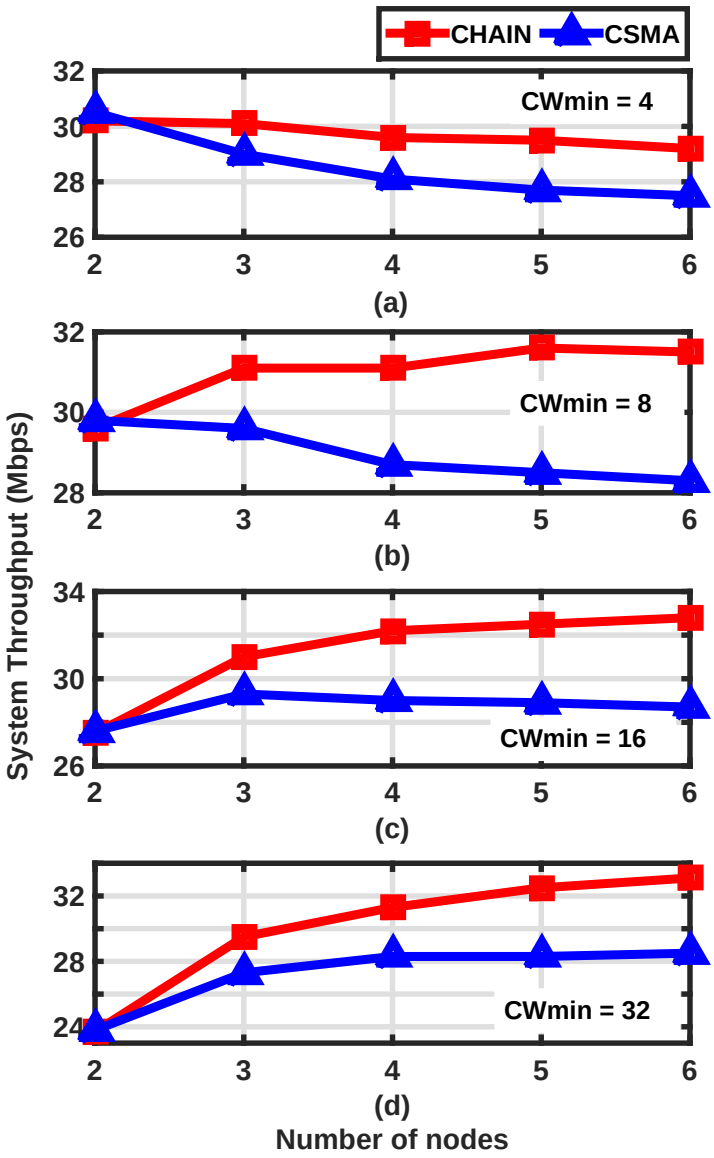

Figure 3.3: System throughput under CSMA and CHAIN vs different CWmin: (a) CWmin = 4, (b) CWmin = 8, (c) CWmin = 16, (d) CWmin = 32.

While we could have decided to set "Enable Piggyback?" as a control on the piggyback transmission block, we decided not to, so that other blocks could enable and disable piggyback transmission based on some parameter instead of just controlling it from the host. As it is, the Piggyback Transmission block was implemented as follows. The input registers are whether to enable piggyback, packet type of last decoded packet, destination address of last decoded packet, and reset output. The output register shows whether an event that triggers piggyback transmission has occured, and, in CHAIN, is connected to the TX packet block. Comparison registers are packet type for piggyback transmissions and the address that triggers piggyback (also known as piggyback address), and they have to be equal in this case.

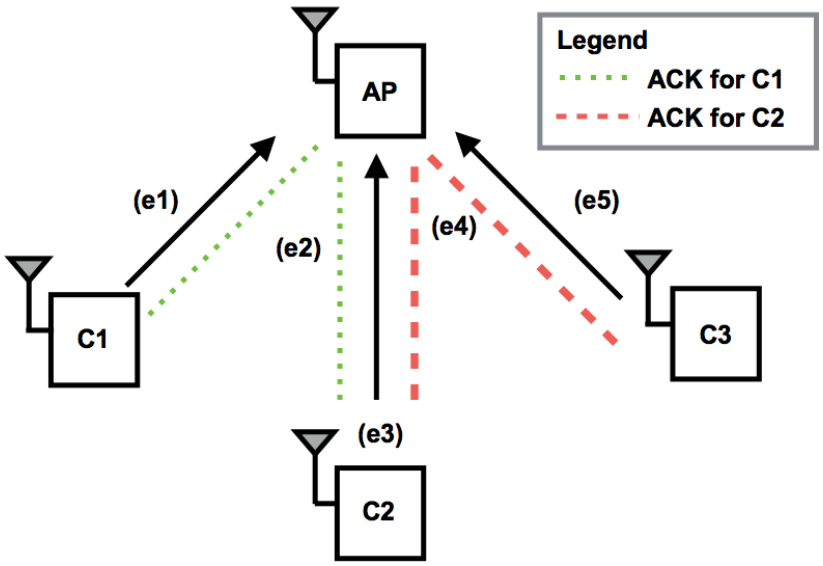

Figure 3.4: Example of piggyback transmissions with three clients and an AP in CHAIN.

To implement piggyback transmissions in CHAIN, the comparison register packet type was set to ACK packets, and comparison register to the precedent address. When a new

packet gets decoded, the packet type is checked to see whether it is an ACK packet, and if it is, whether or not the addresses match. If both of these two conditions are satisfied, the output register sends a piggyback transmission. Then, the TX packet block knows that it needs to send a piggyback transmission, and does so after SIFS time. When a transmission happens, the piggyback transmission block is reset using reset output on the piggyback transmission block. The piggyback transmission block in WiMAC is shown in Figure 3.5.

We compared the uplink results of CHAIN and CSMA for two to six nodes, for CWmin values of 4, 8, 16, and 32. The results are shown in Figure 3.3. As can be seen, CHAIN increases uplink throughput by up to 16.2% in the case of six nodes and CWmin = 32. This is because collision rarely happens in this case, allowing piggyback transmissions to occur fairly often.

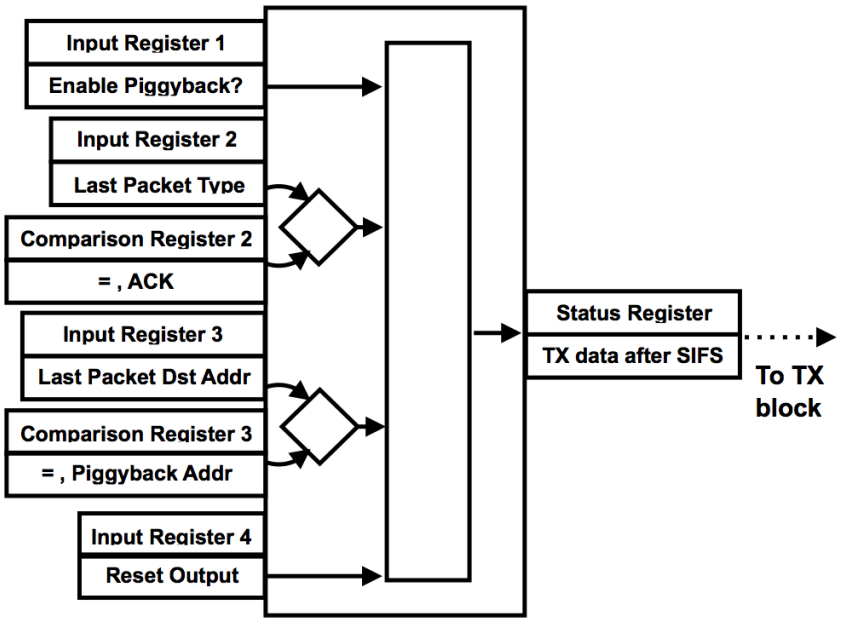

Figure 3.5: Piggyback transmission block in WiMAC.

However, even in the case where CWmin = 4, piggyback transmission help to offset the effects of frequent collisions by a significant amount. No such realistic numbers have ever been presented before, which is a statement generally true of most MAC protocols proposed in the literature.

3.5.4 Max-Weight Scheduling Policy

The class of Max-Weight scheduling policies has been extensively studied in networking theory since the seminal chapter of [75]. As indicated by its name, at each decision time, the Max-Weight scheduling policy chooses the schedule that maximizes the total *weight*, which is usually defined by the state information of each client, such as the queue length of the packet buffer and the transmission rate of the wireless channel. Max-Weight scheduling has been shown to theoretically achieve *throughput optimality*, i.e. the ability to meet the throughput requirements whenever possible, without knowing any information about the packet arrival processes. Max-Weight scheduling has been applied to various setups of wireless networks to achieve optimal throughput performance [76, 77, 78, 79]. Moreover, Max-Weight scheduling has also been proved to achieve good delay performance under some conditions [80, 81]. Despite the progress in theoretical studies, however, there has been no realistic implementation of Max-Weight protocols. The flexibility of WiMAC was utilized to explore the real performance of Max-Weight scheduling.

As a centralized control algorithm, Max-Weight is especially suitable for infrastructure networks. In this chapter, we consider a single-hop wireless network with an AP and multiple clients directly connected to the AP. We define the *weight* of each link as the queue length of the packet buffer of the corresponding link (as in [53]). Under the Max-Weight policy, the AP schedules the uplink packets by selecting the client with the largest queue length at each decision time. For downlink transmission, since the AP has full control over the channel as well as the queue length information of each link, the AP can

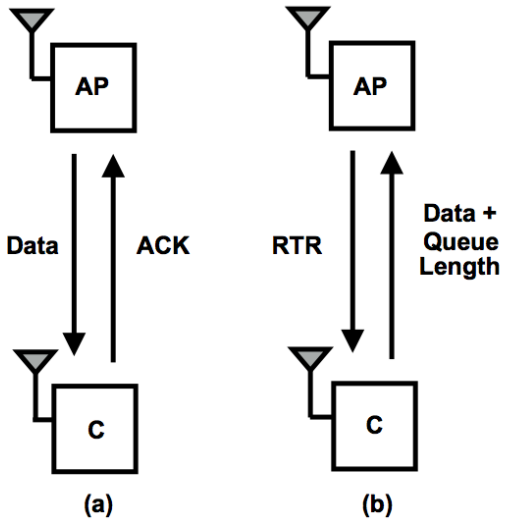

Figure 3.6: How (a) downlink and (b) uplink transmissions are done in Max-Weight.

deliver packets to the clients by using the typical data plus ACK scheme, as shown in Figure 3.6(a).

On the other hand, for uplink transmission, there are two characteristic features in the Max-Weight protocol implementation. First, contrary to contention-based protocols (such as CSMA), under Max-Weight scheduling the AP is responsible for triggering the transmission of all the clients to achieve contention-free medium access. To implement this mechanism, we add a new packet type called Ready to Receive (RTR) for the AP to allocate the wireless medium. As shown in Figure 3.6(b), each client only transmits data to the AP when it receives the RTR request. Note that the transmission triggered by the AP shares similarity with the piggyback transmission in CHAIN. The only difference lies in that, instead of checking for an ACK of a precedent, each client checks for a RTR in the packet header. Accordingly, the same feature that allows piggyback transmissions also allows for AP-triggered transmissions. This example demonstrates the benefit of identifying distinguishing features of MAC protocols.

The second feature of the Max-Weight protocol is using queue length information for

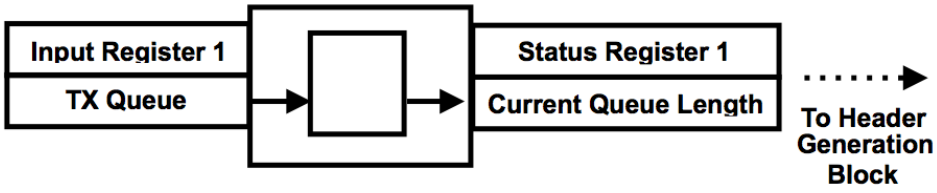

Figure 3.7: Block used in Max-Weight to obtain the TX queue length of the node.

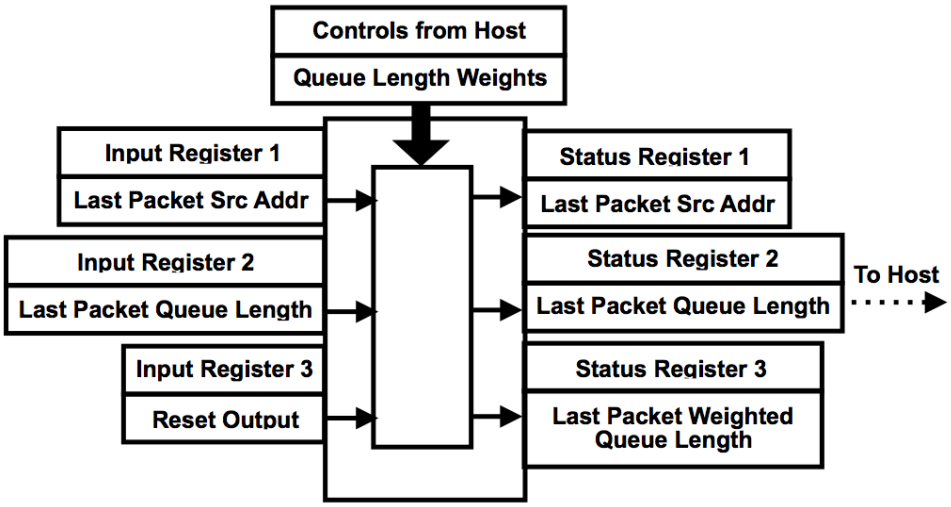

Figure 3.8: Block used in Max-Weight to associate different clients with their respective (weighted) queue lengths.

scheduling transmissions. In order to apply Max-Weight scheduling, the AP needs to know the queue lengths of all the clients before it allocates the channel to a client. One straightforward approach is to poll every client to collect the required queue length information. However, the polling overhead can possibly overwhelm the whole network as discussed in [82]. To deliver the queue length information without incurring polling overhead, the clients are designed to append the queue length information in the header of each data packet. The feature of queue length information is implemented as two functional blocks in hardware:

- For TX: Acquire the queue lengths from the number of elements in the packet buffer. Input is set to the transmit queue, and there are no controls from the host. This block is shown in Figure 3.7. Note, while in Max-Weight there is only one TX queue, the block is implemented to allow support of more than one queue.

- For RX: Read the source address of the received packet and decode the queue length field. Information of the above two fields is stored in output registers. For this block, the inputs are source address, queue length of the last decoded packet, and a reset. Outputs are the source address and queue length of the last decoded packet, and there is also a weighted queue length option that can weight each queue by weights specified from the host. This block is shown in Figure 3.8.

This way, the AP is able to obtain the up-to-date queue length of the scheduled client. However, if a client has not been scheduled for a long time, the queue length kept in the AP may become out-of-date and result in bad schedules. To handle this problem, a Time to Update (ToU) function is added to keep track of the difference between current time and the time of the last transmission for each client. When the time since last RTR of a client exceeds the timeout threshold, the AP transmits RTR immediately to update the queue length.

Figure 3.9 provides an example of how Max-Weight scheduling is performed for uplink. In the set up, there is one AP scheduling for two clients, C1 and C2. QL1 and QL2 refer to the queue lengths of C1 and C2 respectively.

The events in Figure 3.9 are described as follows:

- (e1) The AP starts with QL1 = 15 and QL2 = 11. Under Max-Weight scheduling, the AP schedules C1, which has the largest queue length.

- (e2) After one RTR, QL1 becomes 14 and is still larger than QL2. The AP keeps scheduling C1.

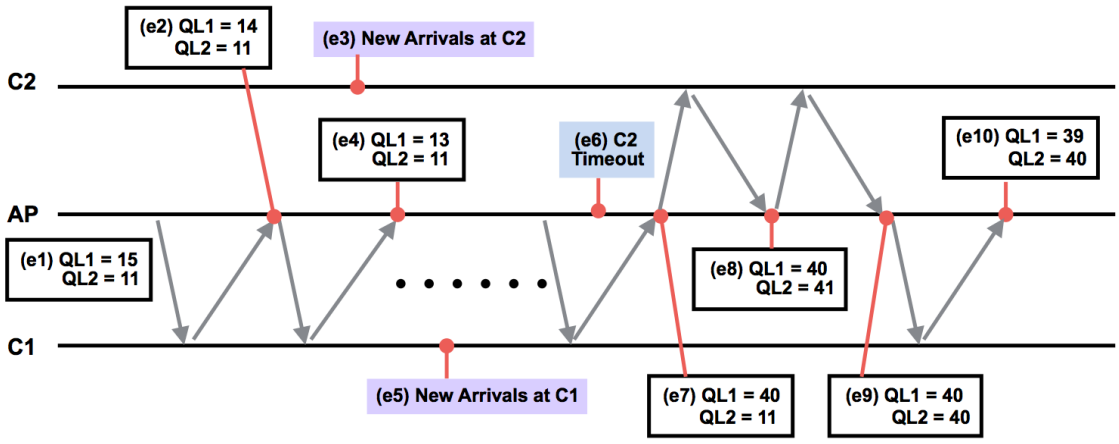

Figure 3.9: Example of Max-Weight scheduling with two clients.

- (e3) New data packets join the buffer of C2. However, QL2 is not updated and stays at 11.
- (e4) After one RTR, QL1 becomes 13 and is still larger than QL2. The AP keeps scheduling C1.
- (e5) New data packets join the buffer of C1.
- (e6) The time since last RTR of C2 exceeds the timeout threshold.
- (e7) QL1 = 40 > QL2 = 11. However, due to (e6), the ToU function triggers RTR to C2.
- (e8) QL2 is updated after (e7). Now, since QL1 = 40 < QL2 = 41, the AP schedules C2.
- (e9) After one RTR, QL2 becomes 40, which equals QL2. The AP breaks the tie by choosing the one with the smallest ID (C1 in this example).

- (e10) After one RTR, QL2 becomes 39, which is less than QL2. The AP schedules C1.

The timeout threshold provides an upper bound for the delay of queue length updates. By using the queue length information with bounded delay, Max-Weight scheduling can still preserve throughput optimality as discussed in [83]. Moreover, since the ToU function handles only high-level instruction of the AP, it is implemented completely in the software domain and can be changed during runtime. This further shows the benefit of decoupling software from hardware in WiMAC.

Table 3.2: System throughput under Round-Robin scheduling and Max-Weight scheduling

Case	System Throughput (Mbps)	
	Round-Robin	Max-Weight
1	9.8	14.6
2	17.0	21.9
3	24.8	30.1

Next, we evaluated the performance of the Max-Weight scheduling policy for an uplink network with one AP and five clients. The data packets join the buffer of each client at a specified rate. The packet generation rates (in packets per second) of the five clients are as follows:

- Case 1: (50, 150, 250, 350, 450)
- Case 2: (75, 225, 375, 525, 675)
- Case 3: (100, 300, 500, 700, 900)

Each packet has a size of 1500 bytes. First, the Max-Weight scheduling is compared with the Round-Robin scheduling policy, which is utilized in the IEEE 802.11 PCF for

contention-free medium access. Table 3.2 shows the overall RX throughput of the AP under these two protocols. In all the three cases, the system throughput is improved by more than 20% under Max-Weight scheduling. This is because Round-Robin scheduling allocates the channel equally to each client, resulting in clients with low packet generation rate wasting much network resources due to their empty buffers. Second, the RX throughput of the AP is shown with different timeout thresholds in Figure 3.10. One may observe that the RX throughput can degrade seriously when timeout is set below 20 ms or above 2 second. When the timeout is small, the AP wastes network resources by sending RTRs to the clients with no packets in the queue. On the other hand, when timeout is too large, the AP makes poor scheduling decisions due to the out-of-date queue length information.

Through this implementation, the issue of *the overhead of obtaining up-to-date information* is highlighted, which is not discussed in the existing literature and simulation results. Hence, WiMAC indeed supports wireless researchers exploring the potential problems through implementation.

3.5.5 A Meta Protocol

In this section, a meta protocol was developed from the initial set of features after implementing WiMAC . For this protocol, a hybrid between CSMA and CHAIN was implemented and deployed as the nodes were running. This protocol was chosen to be implemented since the throughput can be expected to fall somewhere between CSMA and CHAIN, so where the protocol changes occur can easily be seen on the graph. The meta protocol is as follows: Suppose we have an infrastructure topology with one AP and multiple clients. When the queue length of any client is less than a certain threshold, use CSMA. However, when any client's queue length is more than the threshold, the node will enable piggyback transmissions. Intuitively, we want to allow clients with higher queue lengths to transmit more frequently.

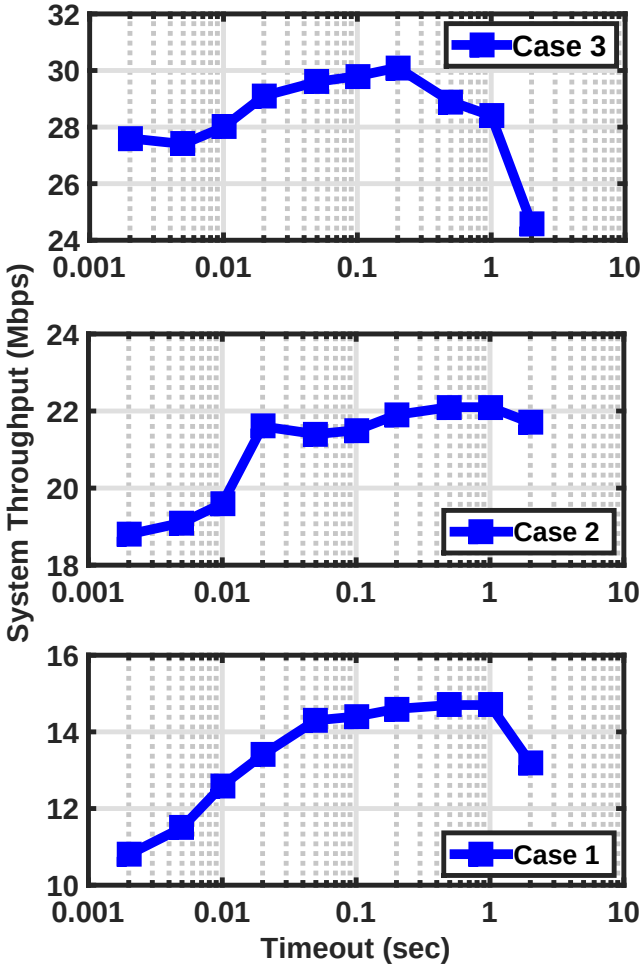

Figure 3.10: System throughput under Max-Weight scheduling versus different timeout thresholds.

The Get Queue Length block, the Decode Header block, and the Piggyback Transmission block were utilized to implement this protocol. First, the TX queue length is obtained and used as one of the inputs on the Piggyback Transmission block. Then the corresponding comparison register is set to check for greater than or equal the queue threshold. Two outputs, packet type and destination address, from the Decode Header block are also used as inputs to the Piggyback Transmission block. The comparison registers were set to check

for equality in both of these cases: ACK packets for the packet type input, and piggyback address for the destination address input. Interactions between these blocks are shown in Figure 3.11.

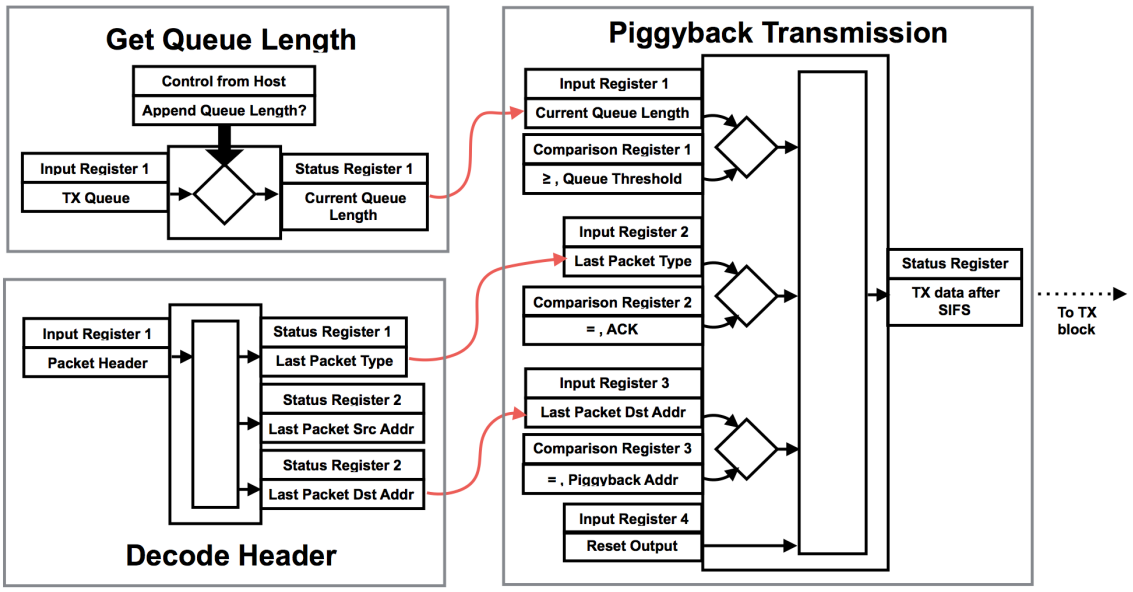

Figure 3.11: How different functional blocks interact with each other in the Meta Protocol.

For the experiment, each node was set to generate 1000 packets per second, with packet size set to 1500 Bytes. The queue length threshold is set to 2000 packets. In Figure 3.12, we show how the throughput changes as the protocol changes from CSMA to CHAIN and finally to the Meta Protocol. The system was run for around 54s under CSMA, and then for around 44s under CHAIN. After CHAIN, the protocol was switched to the proposed Meta Protocol. As we can see, at around 54s the throughput jumps, since CHAIN achieves a higher uplink throughput than CSMA. At around 98s, the throughput drops back to CSMA, since no node has queue length beyond the threshold. Hence, every node is on CSMA. After a short period of time, some of the nodes will have a higher queue length

than the threshold. Those nodes will start piggybacking over their precedent's ACKs. This pushes the system throughput slightly higher than CSMA but still lower than CHAIN.

This simple meta protocol is used to show how completely new protocols can be implemented by connecting different types of blocks, and how protocols can be changed on the fly. Implementation of this protocol also demonstrates how protocols can be designed completely heuristically on WiMAC if so desired.

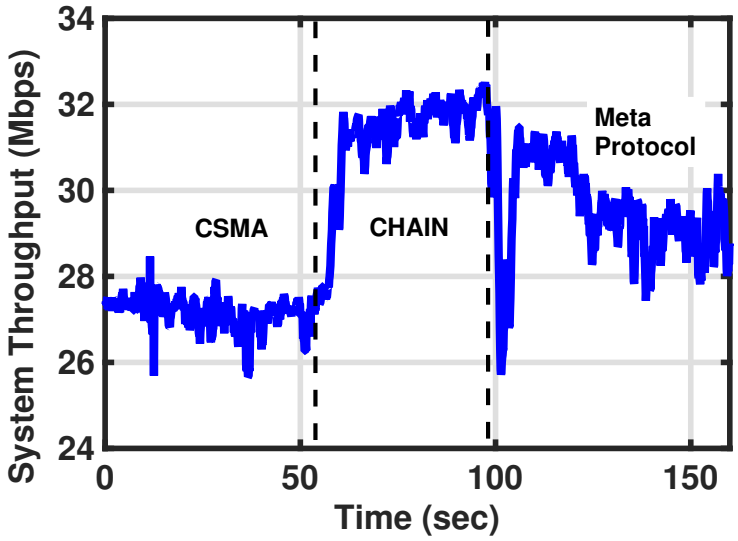

Figure 3.12: System throughput under different protocols. The system uses CSMA from 0 to 54s, then CHAIN until 98s, and finally the Meta Protocol until the end.

3.6 Extensibility of Platform

As the platform develops, we anticipate that new blocks will be added to the system to support different classes of MAC protocols providing extensibility. We demonstrate how new blocks can be easily added to the system, and how these blocks can be used in existing or future protocols.

As an example, let us suppose we want to add a feature that allows a node to sleep. Sleep is a required and widely used feature in energy saving protocols such as [68].

To incorporate sleeping as one of the functions that MAC designers can use when implementing their protocol, we first have to create a generic sleep function block. A sleep function block could look like the following: Control registers to decide whether or not to enable sleep mode. Input registers to specify the condition(s) to check before sleeping or waking, outputs of the blocks that trigger when certain events occur, the length of time to sleep, and force wake. The output registers could be the sleep status of the node, and the remaining time to sleep. A possible implementation of the sleep block is shown in Figure 3.13.

Now, suppose we want to use this block alongside the blocks that have been implemented. We can implement protocols that have any combination of the features that have been implemented. For example, suppose we have an AP that knows the number of nodes in the system. It could tell Client 1 to sleep after transmission and Client 2 to piggyback onto Client 1's transmissions, unless the number of packets in the queue is more than a certain threshold. Or if a client piggybacks when it has a high backoff count, it has to sleep for a period of time before it transmits again. That way, it saves its energy, and the system gets to maintain its fairness.

These are just a couple of examples on how implementing a new feature opens up a new class of protocols that can be implemented, without compromising on performance or the flexibility of the system. As such new features get added to the system, the number of protocols that the platform will be able to support will increase exponentially.

3.7 Conclusion

WiMAC seeks to provide a flexible platform to implement a wide range of protocols while meeting the strict timing requirements of a MAC. Such experimentation can reveal

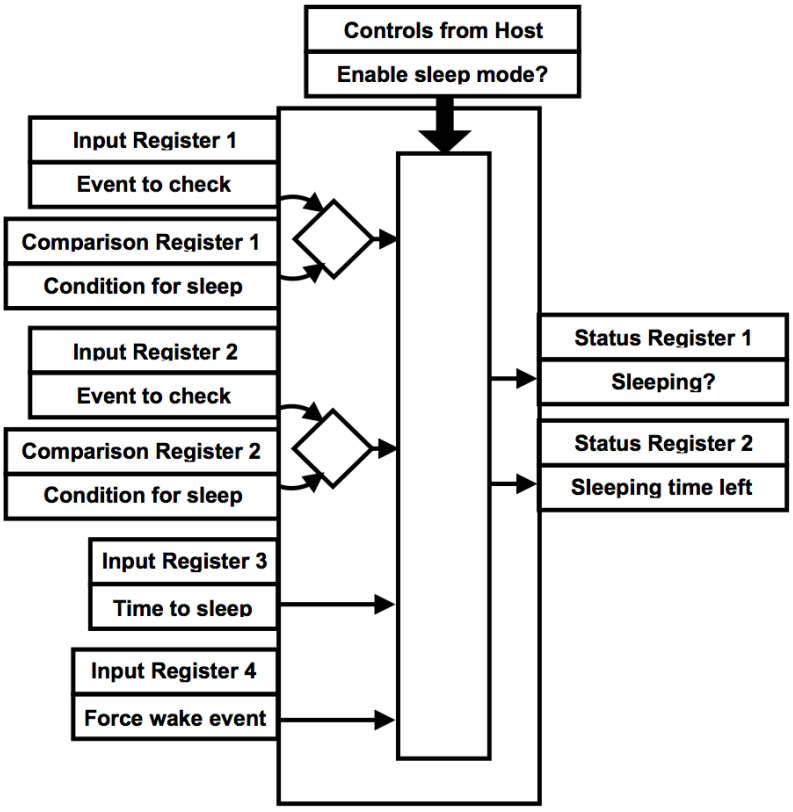

Figure 3.13: A possible implementation of a sleep block in hardware.

issues not considered in the design. An example is shown in the chapter. While Max-Weight has long been touted to require only queue state information from the nodes, we discovered that packet incoming rates, in conjunction with timeout length, impact the optimality of the protocol. This is just a glimpse of the potential issues concerning new MAC designs that can quickly be discovered from experimentation, and how potential design flaws can be resolved quickly. WiMAC has been designed to be easily extensible to support new classes of MAC protocols to explore such problems.

4. MEDIUM ACCESS CONTROL FOR DIRECTIONAL MILLIMETER WAVE WIRELESS NETWORKING

*

4.1 Introduction

As the usage and demand of wireless networks continues to expand, the Federal Communications Commission in the US has released new spectrum in the millimeter wave band (mmWave) for licensed (27.5-28.35 GHz, 37-38.6 GHz, and 38.6-40 GHz) and unlicensed (64-71 GHz) use. Indeed, one of the key enabling technologies of 5G wireless networks is to operate in bandwidths between 30 GHz and 300 GHz. The vast amount of spectrum available in the mmWave bands allows for significantly higher bandwidth that could potentially improve the network performance by orders of magnitude. However, developing wireless technology in this band is not without challenges. One key challenge is the high attenuation at mmWave frequencies. Specifically, it follows from the Friis formula

$$P_R = P_T G_R G_T (\frac{\lambda}{4\pi r})^2,$$

that in free space, the path loss of the signal scales as $\frac{1}{f^2}$ where f is the frequency of the transmitted signal. In the above, P_T and P_R denote the transmitted and received signal powers respectively, G_T and G_R denote the transmit and receive antenna gains respectively, r denotes the distance between the transmitter and the receiver, and $\lambda = \frac{c}{f}$ is the wavelength of the transmitted signal. This in turn implies that for a given transmit and receive antenna gains, the signal power attenuation at 60GHz, which is the frequency band

*

of interest in this chapter, is about 27dB higher than that at 2.5GHz.

In order to overcome this high attenuation, it is imperative to increase the antenna gains G_R and G_T. Fortunately, the antenna dimensions scale inversely with the operating frequency and consequently, embedding tens to even hundreds of antennae in a small form factor becomes feasible at 60GHz, allowing one to perform transmit and receive beamforming, thereby increasing the antenna gains in certain directions [85, 86, 87, 88, 89, 90, 91, 92, 93]. Indeed, the antenna gains G_T and G_R for a given antenna aperture scale as f^2, and consequently, one now obtains a power attenuation that is 27dB *lower* at 60GHz as compared to 2.5GHz [94]. The combination of high bandwidth and higher received power at 60GHz makes mmWave communications an ideal technology to achieve multi-gigabit-per-second wireless communications which has applications in several domains including wireless backhauling and real-time high-definition video streaming.

Employing highly directional beams for transmission and reception introduces certain novel challenges for MAC design. Specifically, highly directional beams introduce the problem of deafness described in [95], and traditional MAC protocols such as CSMA/CA (CSMA with collision avoidance), which rely on the omnidirectional nature of transmissions and receptions, may no longer be effective in orchestrating the medium access. Secondly, the fact that the nodes are directional necessitates the transmitter to keep track of where every station (STA) is, which could be a significant challenge when the STAs are mobile. Note that in such a scenario, it is not only the position of the nodes in the system that affects the network performance, but also their orientations. In other words, as the directional STAs physically move or rotate in a cell, the AP has to track these changes and adapt its transmissions and receptions accordingly. Likewise, each STA has to adapt its transmissions and receptions in response to its own displacement and rotation. Certain other challenges such as blockage also feature in millimeter-wave bands which the MAC protocol has to take into account.

In light of the above challenges unique to millimeter-wave networks, it becomes necessary to develop new directional MAC protocols for these networks which address the problems of deafness, mobility, and blockage. In this chapter, I present TrackMAC, a directional MAC protocol which has certain important beneficial features as described below.

First, TrackMAC allows for both scheduled service periods as well as contention-based channel access, and does so while taking into account the mobility of the nodes, both translational as well as rotational, based on a conservative estimate of how quickly a node can move and rotate. In this context, we introduce the notion of *Topological Coherence Time* T_{tc} of a directional wireless network. Roughly speaking, this is the maximum duration during which the topology of the network remains "constant." This quantity depends primarily on the mobility of the nodes in the network, but could also be influenced by the beamwidths of the antennae. We discuss this in more detail in the following sections.

One of the key bottlenecks affecting the performance of a mmWave network is the delay associated with discovering the relative position and orientation between the AP and a STA. To address this issue, TrackMAC is designed in such a manner that with a small overhead, the need for constant rediscovery of the network topology by the centralized scheduler or AP is eliminated.

Importantly, as we show in Section 4.4, TrackMAC has a significant architectural advantage: It can be implemented squarely within the framework of the IEEE 802.11ad standard. Specifically, TrackMAC can be realized by reprogramming *only the scheduling layer* of an IEEE 802.11ad network stack.

While the issue of blockage of millimeter waves by objects such as the human body, office furniture, etc., is not explicitly addressed in this chapter, the features of protocols that address it, such as multihop relaying [94], can well be implemented within the framework of TrackMAC in a relatively straightforward fashion.

In summary, the contributions of this chapter are three-fold:

1. The introduction of the notion of topological coherence time of a directional wireless network, which is potentially a key parameter for designing efficient directional MAC protocols.

2. TrackMAC, a directional MAC protocol which takes into account the topological coherence time, the directionality of the nodes, and which continually tracks every associated STA with a small overhead, thereby removing the need for constant rediscovery of mobile nodes.

3. Implementation of TrackMAC within the specifications of the IEEE 802.11ad standard.

The rest of the chapter is organized as follows. In Section 4.2, I present related work in this area, and the key distinctions between these efforts and our work is also outlined. Section 4.3 describes TrackMAC, the proposed MAC protocol. Section 4.4 presents certain key features of the IEEE 802.11ad standard, and also describes how TrackMAC can be implemented within the specifications of the standard by reprogramming only the scheduling layer of the network. Section 4.5 presents some simulation results, and Section 4.6 contains some concluding remarks.

4.2 Related Work

Before mmWave was widely viewed as a feasible technology for increasing usable spectrum for wireless networks, there had been several studies examining the use of directional MAC protocols to improve spatial reuse for sub-6GHz ad hoc wireless networks [96, 97, 98, 99, 100, 101, 102, 103, 104, 105]. However, these papers do not consider mobility of the nodes and consequently, may not perform well in an indoor scenario with mobile nodes. In [97, 106, 107, 100], the authors focus on directional MAC protocols for

ad hoc networks. However, protocols designed for ad hoc networks can incur significant overhead and cause significant degradation to the efficiency of an infrastructure wireless network. Furthermore, these protocols were not designed for mmWave bands, and they rely on assumptions that do not apply in these bands.

In [108], the authors briefly mention handling of mobility in a Wide Area Network (WAN) but do not consider how mobility affects the scheduling of nodes. Reference [109] proposes a MAC protocol that works in mobile scenarios and also details how the location of the stations are updated. However, this work is not based on the IEEE 802.11ad standard, and so, it may not be possible to implement it within the specifications of the standard. Similarly, in [110], the authors develop an algorithm for avoiding blockage and dealing with mobility. However, they require all associated STAs to inform the AP of the transmission rates between them and all other STAs every 10ms (for the scheduling decisions), which may not be compatible with IEEE 802.11ad. Full details of the IEEE 802.11ad specifications can be found in [111].

Reference [112] provides a directional MAC implementation on top of the current IEEE 802.11ad standard. However, like some of the papers cited above, it too does not consider mobility of the nodes.

4.3 Design Methodology of TrackMAC

Consider a mmWave infrastructure wireless network as shown in Figure 4.1. The role of the AP is to (i) communicate with the STAs already associated with it, and (ii) enable new nodes in its vicinity to join the network by associating them with it. The latter is commonly known as Initial Access (IA). In what follows, we describe the protocol for both the IA phase as well as for the data transmission phase.

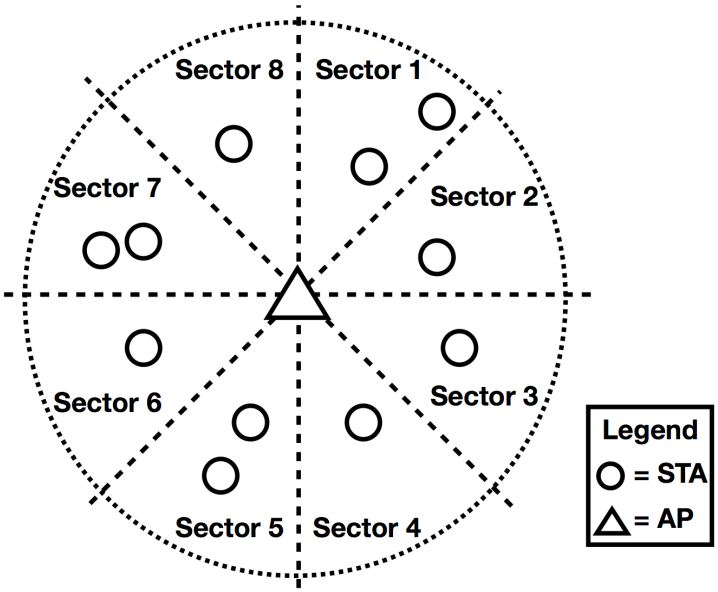

Figure 4.1: A mmWave network operating in infrastructure mode.

4.3.1 Topological Coherence Time

One of the key parameters that determines the exact design of the proposed MAC protocol is the topological coherence time, denoted by T_{tc}. Akin to the notion of the coherence time of a wireless channel, the topological coherence time is a measure of how fast the network topology changes. To illustrate this notion in a very simple context, consider a mmWave network with just one AP and one STA. Suppose also that each of these nodes is equipped with an antenna array with a 3dB beamwidth of, say, 10°. For simplicity, suppose that there is only one dominant path from the AP to the STA, and that it is the Line-of-Sight (LoS) path. Let the AP's beam and the STA's beam be perfectly aligned at time $t = 0$. As time progresses, the STA moves and rotates, so that the STA's beam becomes misaligned with respect to the AP's beam. Consequently, the link gain $G_T G_R$, where G_T and G_R are the transmit and receive antenna gains respectively, fluctuates from its peak value. The topological coherence time is defined as the time t until which any link's gain degrades

no more than 3dB from its peak value. Clearly, in this example, the topological coherence time is a function of the maximum rotational and translational speeds of the STAs. For instance, if the STAs are assumed to be stationary and the maximum rotational speed of the STAs is 360 degrees/s, a very conservative estimate, then, for a 3dB beamwidth of 10°, the topological coherence time is of the order of $\frac{5}{360} \approx 14$ms.

In a more general scenario, the topological coherence time may not be expressible explicitly in terms of just the mobility parameters of the STAs, but nevertheless, it is a quantity that, like channel coherence time, is measurable in an order of magnitude sense.

4.3.2 TrackMAC: A Novel Directional MAC Protocol

In what follows, we describe the TrackMAC protocol as well as the methodology by which the parameters of the protocol are designed. For the sake of exposition, we make use of a running example to illustrate the design methodology.

In the proposed MAC protocol, time is divided into a series of slots known as “macroslots." The duration of each macroslot, denoted by T_M, is set to be $T_M = \frac{T_{tc}}{p}$, where $p \geq 2$ is an arbitrary design parameter. In our running example, we take $p = 2$. Also for our example, we assume T_{tc} to be of the order of 10ms, so that the macroslot duration is $T_M = 5$ms.

Now, depending on the maximum number of STAs that are to be supported by an AP, each macroslot is further subdivided into several slots known as “microslots." Specifically, if N is the maximum number of STAs that are to be associated with an AP, then, each macroslot is divided into N microslots, so that each microslot extends for a duration of $T_m = \frac{T_M}{N}$. In our example, we suppose that the AP is not expected to serve more than $N = 25$ STAs at a time, so that each macroslot consists of 25 microslots. It follows that each microslot extends to a duration of $T_m = 200\mu$s. This structure of macroslots and microslots is illustrated in Figure 4.2.

Now, during the data transmission phase (as opposed to IA phase), each STA associated

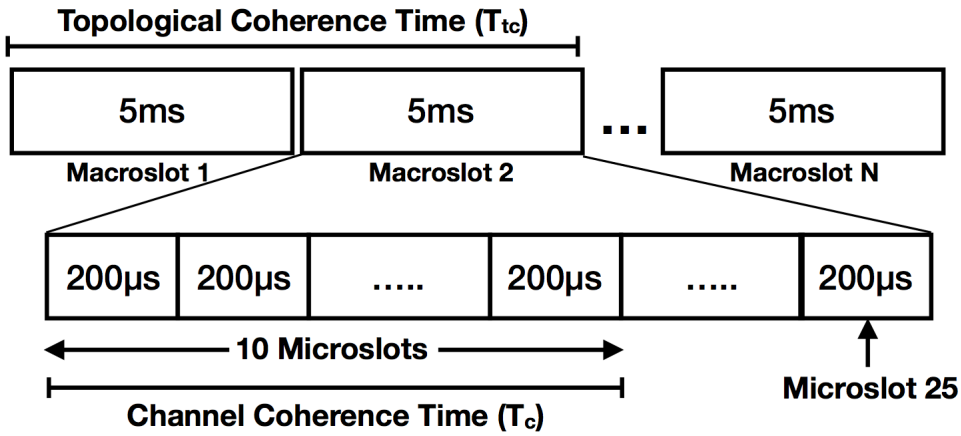

Figure 4.2: Division of time into macroslots, of macroslots into microslots.

with the AP is scheduled by the AP in certain microslots. The only constraint for the AP is that it schedules each STA at least once in each macroslot. Note that since by design the time difference between any two microslots belonging to adjacent macroslots is lesser than the topological coherence time, the AP knows the direction in which it has to point its beam in each microslot in order to communicate with the intended STA. Once the link is established during a microslot, the AP and the STA consume some training overhead time to refine their beam directions. This ensures that the STA never strays too far away from the stored direction in the AP's direction table, and can always be tracked as long as the AP schedules the STA at least once in each macroslot. Subject to just this one constraint, the AP can schedule the STAs in any manner that maximizes the network utility. In fact, as long as the AP satisfies this scheduling constraint, it can also allocate certain microslots in each macroslot for contention-based access (CBAP). Any contention protocol designed for directional nodes can be employed in this time period. Finally, the ACK frames are transmitted by the STAs at the end of each microslot using time division duplexing.

Next, we describe and evaluate the MAC overhead required for the functioning of TrackMAC. In addition to beam refinement, training overhead time should be and is allot-

ted in each microslot for the purposes of (i) symbol timing recovery, (ii) channel estimation, and (iii) frame synchronization. We now estimate the amount of overhead required for each of these.

Based on indoor channel measurement studies at mmWave frequencies, the delay spread of the channel in a typical indoor office-like environment is at most 500ns [113]. This mandates that the channel estimation (CE) pilots be at least 500ns in duration. While allocating a CE pilot duration that is an order of magnitude larger than the channel delay spread may be preferable for smoothing the channel estimate in the presence of noise, any allocation higher than the delay spread is sufficient for estimating the channel impulse response. While in our running example we allocate 5μs for CE pilots, which is an order of magnitude larger than the delay spread of the channel, an allocation of about 650ns would be in adherence with the 802.11ad standard as explained in the next section. Next, we examine how often within a microslot the channel has to be estimated. This depends on the channel coherence time. For a typical indoor speed of about $v = 1.5$m/s, the doppler spread D of the channel at 60GHz is about $D = \frac{f_c v}{c} = 300$Hz, corresponding to a channel coherence time that is of the order of $\frac{1}{D} = 3.33$ms. Since this an order of magnitude larger than the microslot duration, it is sufficient to allocate just 5μs per microslot out of the 200μs available for channel estimation.

Typically, standard sequences, such as Golay sequence or Zadoff-Chu sequence, of various possible lengths are chosen as training symbols for timing recovery and frame synchronization owing to certain autocorrelation properties that they exhibit. For example, as shown in [114] in the context of mmWave links, a sequence length of 2048 symbols provides robust symbol timing recovery. We use the same training length in our running example. Consequently, for a symbol duration of $T_s = \frac{1}{W} = 0.46$ns that is typical in the 60GHz band for a bandwidth $W = 2.16$GHz (in adherence with the IEEE 802.11ad standard), this translates to a pilot duration for timing recovery of about 1μs. Finally, while

training symbol duration for beam refinement can also be chosen freely, in our running example, we allocate between 10μs and 15μs for this, which again is in adherence with 802.11ad.

Adding all of the above pilot durations, we obtain a total overhead of approximately 20μs per microslot, which leaves about 180μs per microslot for transmission of the MAC payload. This in turn translates to a total MAC overhead of about 10%.

In addition to the above training overheads, some specified time $t_{control}$ may be allocated in each microslot for exchanging control information. Although we do not do this in this chapter in order to adhere with 802.11ad specifications, in general this could be done, and during this time, the AP and the STA could exchange information such as queue backlog, packet deadlines, channel quality index, the microslot in the next macroslot in which the STA is scheduled, etc. Note that this allows the AP to schedule the STAs adaptively every macroslot based on their queue backlogs, required deadlines, etc.

In order to allow for initial access, we designate one macroslot periodically as "IA macroslot." In our example, we suppose that one macroslot in every twenty macroslots is designated as an IA macroslot. In an IA macroslot, the AP transmits a beacon in one of the pre-defined sectors using a low-order modulation scheme such as BPSK and a low rate code. A STA that wishes to associate with the network configures its antenna in a quasi-omnidirectional mode similar to the IA procedure of IEEE 802.11ad described in the next section. While the quasi-omnidirectional reception leads to low received SNR, the fact that the beacon, which is only a few bits in length (not more than 1000 bits in 802.11 systems), is transmitted at a very low rate enables the STA to reliably decode the beacon. Once the beacon is decoded, the AP and the STA perform the beamforming procedure specified in the 802.11ad standard [111] to align their beams. Some specified duration of the IA macroslot is also allocated to communicate the schedule of every associated STA until the next IA macroslot.

We do not specify a duration for the IA phase, but the longer this period is, the higher the probability of an STA being associated with the AP. Consequently, more STAs can be associated with the AP during a single IA phase. On the other hand, if this period is set too short, an STA may require many IA phases before finally being associated with the AP (or reassociated with the AP if the AP fails to track the STA). Once the STA associates with the AP, the latter stores the location of the newly associated STA in its direction table, and also specifies its schedule till the next IA Macroslot.

4.4 Implementation of TrackMAC Within the IEEE 802.11ad Specifications

Before we describe how TrackMAC can be implemented within the specifications of the 802.11ad standard, we provide a brief overview of certain key specifications of the standard. Some basic familiarity with the standard is assumed of the reader in this section. A more comprehensive description of the standard can be found in [115, 116], and the standard itself can be found in [111].

Consider a Basic Service Set (BSS) operating in infrastructure mode – the AP is connected to all other STAs. Similar to legacy 802.11 standards, the 802.11ad divides the time axis into Beacon Intervals (BI). While the exact duration of a BI can be chosen by the system designer, it has to be chosen in the range of 100ms and 1000ms. In most practical deployments, it is chosen to be in the order of about 100ms.

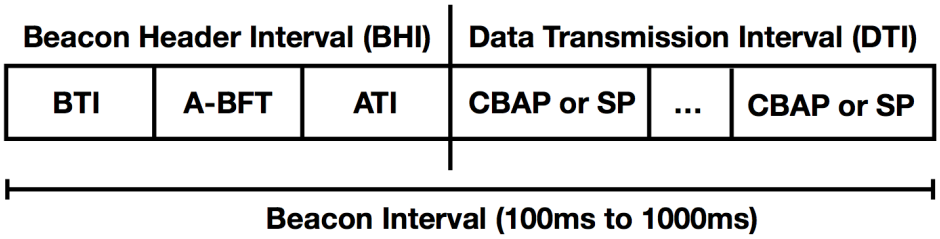

Figure 4.3: Structure of a Beacon Interval.

Figure 4.3 illustrates the structure of a BI. As shown, each BI is composed of two phases - the Beacon Header Interval (BHI) and the Data Transmission Interval (DTI). The BHI duration is not specified in the standard, and is free to be chosen by the system designer. Typically, the BHI is in the order of a few milliseconds, and the DTI consumes the bulk of the BI. In our example, we dedicate about 2ms for BHI, and about 98ms for DTI.

The BHI is further subdivided into three phases - the Beacon Transmission Interval (BTI), followed by an Association Beamforming Training (A-BFT) interval, and finally the Announcement Time Interval (ATI). During the BTI, the AP transmits a beacon directionally in certain predefined sectors as mentioned before using MCS 0. This is the lowest rate MCS scheme defined in the 802.11ad standard, and offers maximum error protection. This MCS is typically used prior to beamforming to exchange critical control information in a robust fashion [115], but never for data transmission since it is severely limited in terms of data rate.

The A-BFT period is used for associating and beamforming training of the new STAs. Specifically, during the A-BFT phase, the AP and the STA transmit and receive pilot sequences in different predefined sectors to converge on the best transmit and receive sectors. While these sectors, being predefined, are quantized and provide coarse beamforming, finer beam refinement could also be performed to converge on optimal beam directions.

Finally, during the ATI, the AP informs every associated STA the schedule during DTI. Specifically, during DTI, some time periods can be dedicated for servicing specific STAs, while other time periods could be designated for contention-based channel access. Whatever the schedule is, the AP informs each associated STA of the schedule.

The DTI begins after the ATI. Each DTI can support multiple data transmissions to different STAs, and they can be scheduled in any fashion by the scheduling layer. The standard supports a hybrid access method, i.e., both scheduled and contention-based trans-

missions are supported during the DTI.

Figure 4.4 illustrates how TrackMAC can be implemented within the 802.11ad framework. Specifically, with respect to the MAC layer, the IA macroslots in our protocol essentially are designed in such a way that microslots 1 thru 10 of the IA macroslots are designated for BHI. This translates to a BHI duration of 2ms (recall that each microslot is 200μs long). To attain a BI duration of 100ms, one IA macroslot is interleaved between every 19 macroslots (recall that a macroslot is 5ms long). The overlay of macroslots on an 802.11ad BI is shown in Figure 4.4.

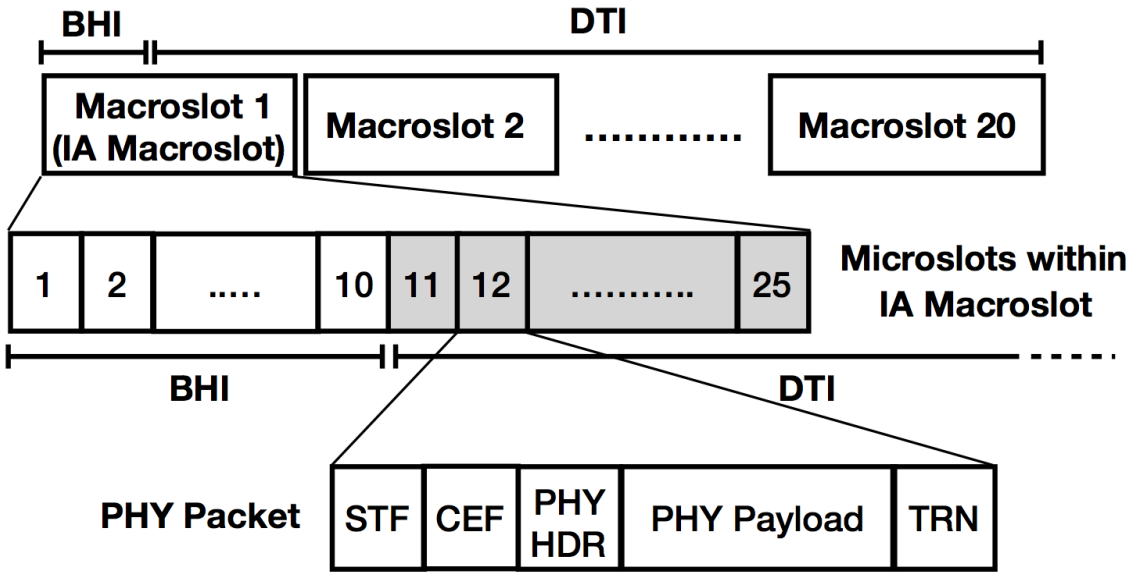

Figure 4.4: Overlay of 802.11ad PHY packets on microslots and macroslots on 802.11ad BI.

Finally, a physical layer packet of 802.11ad can be of three types, viz., control PHY, single-carrier PHY (SC-PHY), and Orthogonal Frequency Division Multiplexing (OFDM) PHY. All physical-layer packets of 802.11ad have the same set of fields - the Short Training Field (STF) for signal detection and symbol timing recovery, the Channel Estimation Field

(CEF), the PHY Header field which contains all necessary information to decode the MAC payload, the MAC payload, and finally the beamforming training field (TRN). It is only the duration of each field and the MCS used in the MAC payload that is different for different PHY packet types. The structure of a PHY packet with different fields marked is shown in Figure 4.5.

The beamforming training field of a PHY packet consists of pilot sequences that are transmitted by the AP in different sectors adjacent to the direction in which the MAC payload is transmitted. The number of sectors along which training sequences are to be transmitted can range from 0 to 64 in multiples of 4.

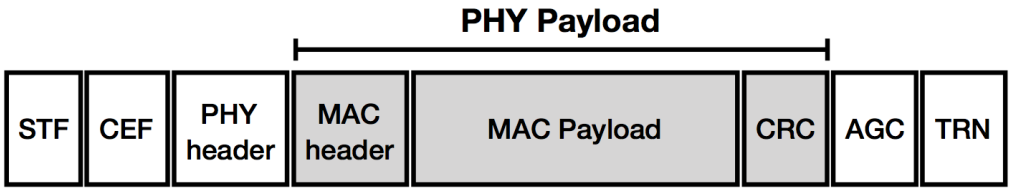

Figure 4.5: 802.11ad PHY packet structure.

Also shown in Figure 4.4 is the overlay of PHY packets used in 802.11ad on the microslots. Specifically, a PHY packet of 802.11ad has all the fields present in a microslot defined in TrackMAC. Furthermore, the duration of the different fields of the 802.11ad PHY packet is in conformity with those of a microslot. The STF of the 802.11ad SC-PHY packet, which serves the purpose of pilot symbols for symbol timing recovery, is about 1.2μs long, which is the about the same duration alloted in a microslot for timing recovery. Similarly, the channel estimation field of an 802.11ad SC-PHY packet is 645.12ns, which can be mapped to the channel estimation pilots in a microslot. Finally, the beamforming training fields in 802.11ad can range anywhere from a little more than 10μs to

about 180μs depending on the number of sectors that pilot symbols should be transmitted in. However, note that since the time difference between two consecutive channel accesses of a STA is never more than the topological coherence time in TrackMAC, it is enough for beamforming training sequences to be transmitted in just $S = 4$ sectors adjacent to the direction of payload transmission, for typical beamwidths used. For this choice of S, the duration of the training field extends to a little more than 10μs in 802.11ad, which conforms to the beamforming training field duration in a microslot defined in TrackMAC.

The aforementioned mapping ensures that the scheduling algorithm running on top of an 802.11ad MAC layer organizes medium access, as well as the physical layer transmissions, are in adherence with both TrackMAC and the 802.11ad specifications. The scheduling algorithm needs to only satisfy the constraint that every STA is scheduled at least once in every macroslot in order to track every mobile STA associated with it.

4.5 Simulation Results

As described in Section 4.3, an important feature of the proposed scheduling framework or the TrackMAC is its ability to track every mobile station that is associated with it. One of the primary purposes of our simulations is to evaluate the tracking efficacy of TrackMAC. Towards this, we evaluate in our simulations (i) the tracking performance of the AP when it employs the proposed scheduling algorithm, and (ii) the average received signal strength for different mobility parameters, a key physical layer performance metric.

In our simulations, each node (AP or STA) is equipped with a Uniform Linear Array (ULA) of 64 elements, which gives it the ability to beamform. The inter-element spacing in the ULA is set to $\lambda/2 = 25$mm, where $\lambda = \frac{c}{f_c}$ is the carrier wavelength. Figure 4.6 shows the azimuth pattern of such an array. The 3dB beamwidth of the array is about 1.6°.

We simulate the system in a “fully loaded" condition, i.e., the number of STAs associated with the AP equals the number of microslots. The number of microslots is set to

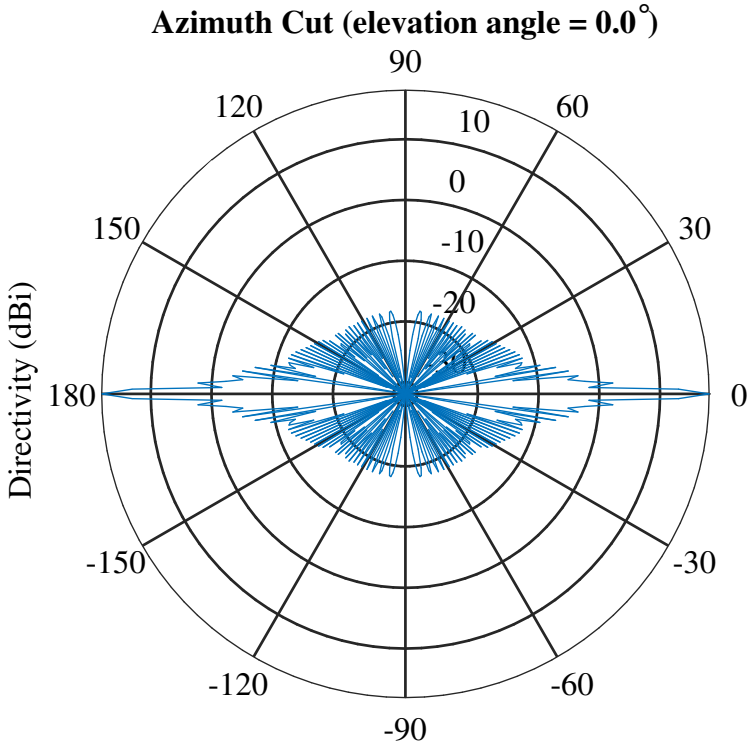

Figure 4.6: Azimuth pattern of a $\lambda/2$-spaced 64-element ULA at 60.48GHz.

25 in our simulations, resembling the design in our running example. Also, the network is simulated for a duration of 200ms which equals two beacon intervals.

In a LoS indoor environment and for typical translational and rotational velocities of users, the dominant factor that determines the topological coherence time is the rotational speed of the STAs. Motivated by this, we first present the simulation results of a scenario in which the STAs are stationary but rotate at a speed chosen uniformly at random with a given upper bound. In this case, the receiver uses the training fields (TRN) to steer its beam in the direction of best received signal strength. If the receiver, in addition to rotating about its position, also moves, then the AP has to track the STAs. In a second

scenario, we simulate this case with the STAs having their receive beams directed towards the transmitter, but are mobile, thereby requiring the AP to track them as they move in the azimuth. In this case, the STAs move at a speed chosen uniformly at random with a given upper bound.

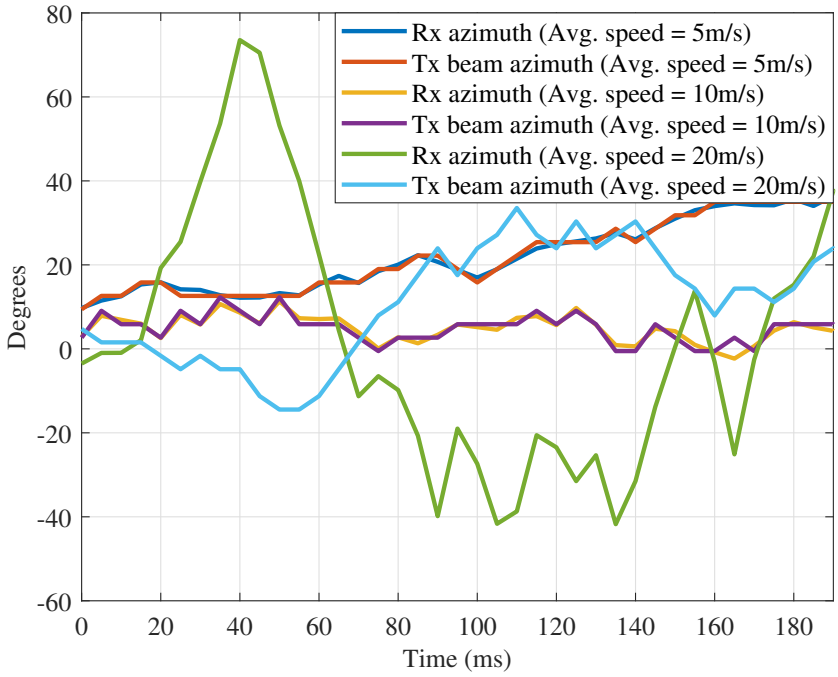

Figure 4.7: Tracking performance of the AP for three different STA translational speeds (5m/s, 10m/s, and 20m/s). Plotted against time are (i) the azimuth direction of a particular randomly chosen STA out of the 25 associated STAs, and (ii) the azimuth direction in which the AP points its transmit beam when it communicates with that particular STA. For STA translational speeds of 5m/s and 10m/s, the AP is able to track the movement of the STA, whereas for STA translational speed of 20m/s, the AP is unable to do so.

Figure 4.7 and Figure 4.8 illustrate the tracking performance of the AP as the receiver's azimuth varies. The receiver is assumed to be beamformed towards the AP, and the beamforming training field transmitted in the PHY packet of the AP consists of train-

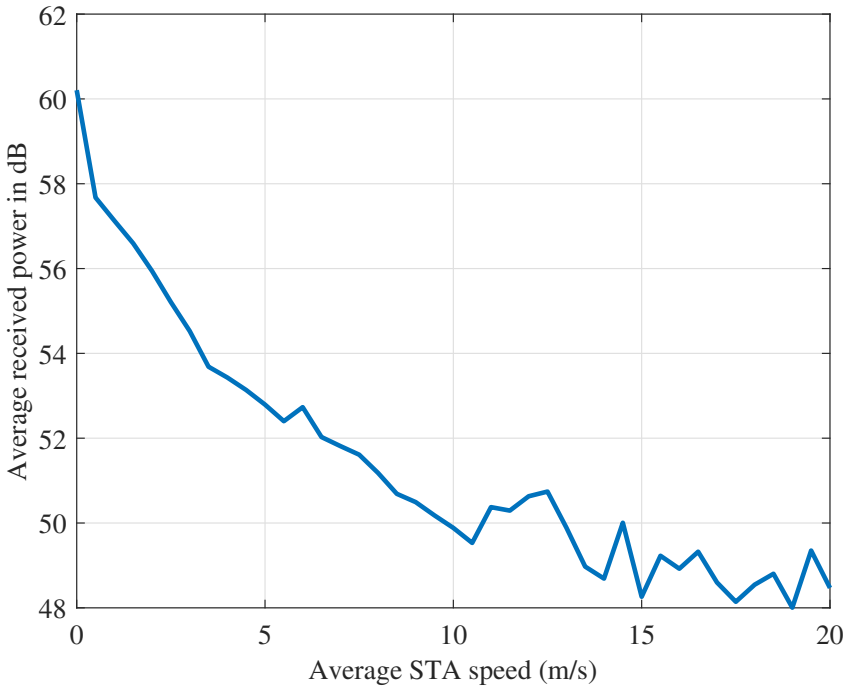

Figure 4.8: Average received signal power vs. translational speed of STAs. The abscissa denotes the avg. speed at which the STAs move, and the ordinate denotes the avg. power level at which the MAC payload is received. The averaging is performed over STAs.

ing symbols for four sectors. Since the 3dB beamwidth of the antenna is about $1.6°$, the training sectors of the PHY packet are chosen to be $\pm 1.6°$ and $\pm 3.2°$ relative to the direction in which the payload is transmitted. Figure 4.9 shows the array pattern when the antenna beam is steered using steering weights corresponding to $3.2°$ in the azimuth. It can be seen from Figure 4.7 that TrackMAC is able to track translational speeds that are fairly high for indoor scenarios (upwards of 10m/s), and deteriorates as the STA speed becomes larger and larger. Figure 4.8 plots the average power level at which the MAC payload is received a function of STA speed. Specifically, if $\mathbf{x}_{m,n}$ denotes the signal received by STA m, $m = 1, ..., 25$ during macroslot n, $n = 1, ..., 40$ (the network is simulated for 2 BIs which equals 200ms or equivalently, 40 macroslots), then the figure

plots $\frac{1}{25}\sum_{m=1}^{25}\frac{1}{40}\sum_{n=1}^{40}\frac{||\mathbf{x}_{m,n}||^2}{T}$. Here, T is the length of MAC payload during a microslot. In our simulations, $T = 41768$ octets. Since the focus is on simulating the MAC protocol, no PHY aspects such as small-scale fading or noise are simulated.

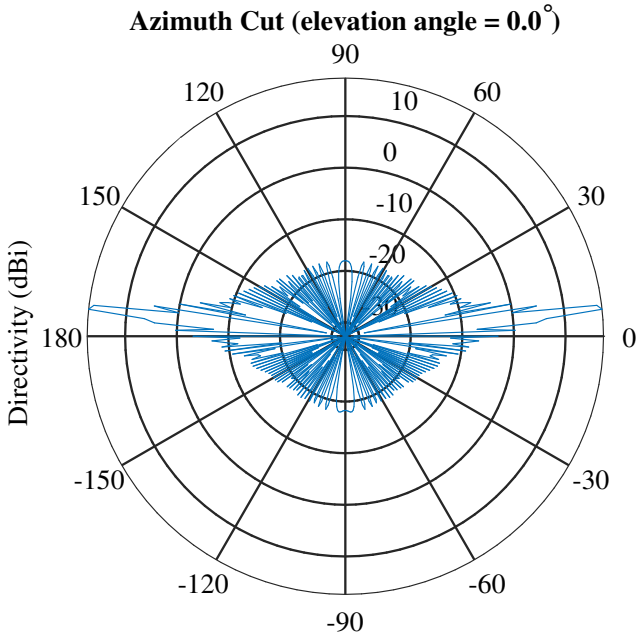

Figure 4.9: Azimuth pattern of the ULA with steering weights corresponding to $3.2°$ azimuth.

Figure 4.10 and Figure 4.11 illustrate the tracking performance of the protocol in the presence of STA rotation. The AP is now assumed to be beamformed in the direction of STAs. As in the previous set of simulations, here too the beamforming training field of the STAs consist of training symbols for sectors $\pm 1.6°$ and $\pm 3.2°$ relative to the direction in which the payload is transmitted. Figure 4.10 shows the average misalignment between the AP and the STA's beams as a function of the STA's rotational speed, and Figure 4.11 plots the average received payload signal power in a microslot. For nominal rotational

velocities expected of users, the misalignment between the beams, and also the average received signal power, are not severely degraded.

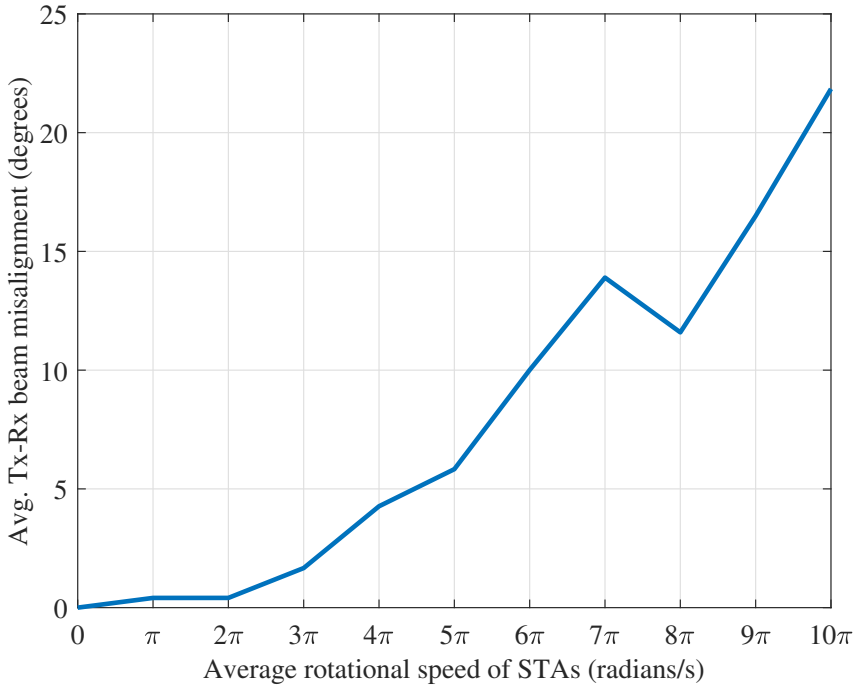

Figure 4.10: Tracking performance of AP and STAs for different rotational speeds. The abscissa denotes the avg. rotational speed of the STAs and the ordinate represents the angle between the directions of the transmitter's beam and the intended receiver's beam, averaged over both STAs and time. The network was simulated for one beacon interval.

4.6 Conclusions and Future Work

This chapter presented TrackMAC, a directional MAC protocol for indoor mmWave infrastructure wireless networks with mobile users. The high directionality of the nodes in such networks necessitates the AP and each station to track each other over time, preferably with small overheads, in order to obtain sufficient link budget in mmWave bands. A protocol that achieves this was presented, and the required MAC overheads were analyzed.

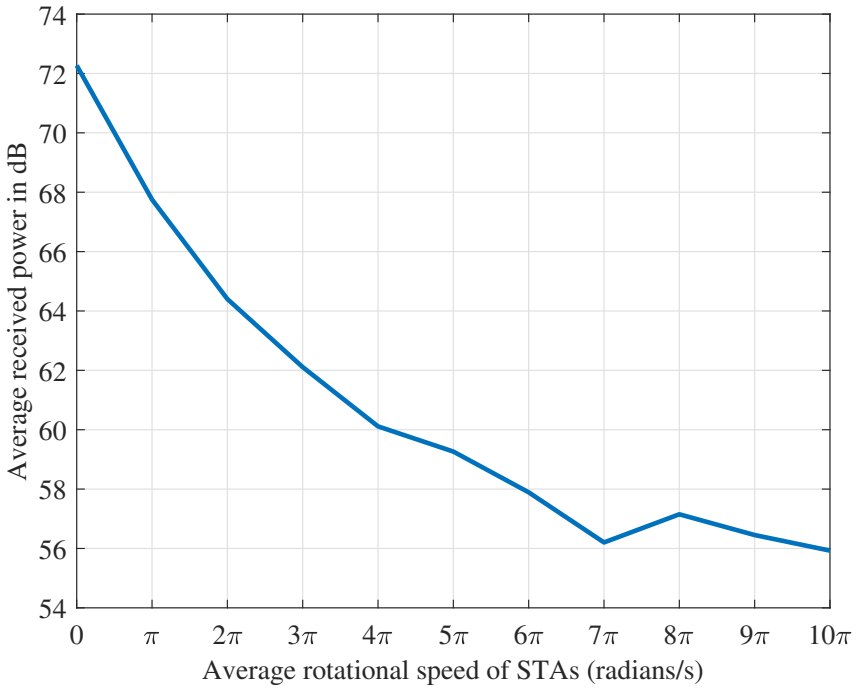

Figure 4.11: Average received signal power vs. rotational speed of STAs. The abscissa denotes the avg. speed at which the STAs rotate, and the ordinate denotes the avg. power level at which the MAC payload is received. The averaging is performed over STAs.

It was shown that for typical indoor speeds, tracking could be achieved with a MAC overhead of about 10%. The tracking performance was evaluated using computer simulations. Finally, it was shown that TrackMAC can be implemented squarely within the specifications of the IEEE 802.11ad standard developed for mmWave wireless local area networks. This is achieved by employing a certain class of scheduling algorithms in the scheduling layer of an 802.11ad network stack. However, as discussed in previous chapters, simulation alone is not sufficient to verify the performance of the protocol. Future work for TrackMAC includes prototyping the protocol on a mmWave platform and evaluating its performance.

5. CONCLUSIONS

The advancements in wireless communication have come a long way since its inception and they are showing no signs of slowing down. In fact, wireless networks will play a bigger role in our everyday lives in the future. As next-generation wireless technologies are developed, researchers are faced with new challenges to solve in order to make these technologies viable for next-generation wireless applications. Verifcation of the solutions to these challenges are performed differently, depending on which networking layer the challenges fall under. At the PHY layer, verification of developed technologies is performed by prototyping the hardware, and performing measurements and experiments on the hardware. At the NET layer, verification is performed by performing extensive network simulations. In some cases, researchers have even started using hardware switches and routers to evaluate the performance of new networking protocols when simulations are unable to sufficiently capture the behavior of the network. In both these cases, the verification methods correspond well to the hardware or software domains the layer primarily operates in. Since the PHY layer is in the hardware domain, verification should be (and is) performed in the hardware domain. Similarly, since the NET layer is performed primarily in the software domain, verification using simulations can capture the behavior of the network adequately. In cases where the protocol has hardware dependencies, networking researchers may bring use hardware routers and switches to verify their claims. At the MAC layer, however, even though it operates in both the hardware and software domain, verification of protocols is typically only done in simulation. This does not provide sufficient evidence that the protocol will work just as well, since the performance of the protocol with the hardware will remain largely untested. Furthermore, a simulation-only verification approach for MAC protocols may overlook some issues that are not found in

simulations as I have shown earlier. To have confidence on the claims of proposed MAC protocols, the protocols should be implemented on hardware to verify their performance in an emulated scenario.

In this dissertation, I have designed and developed platforms that can be used for experimentation of next-generation MAC protocols. In PULSE, we were able to experiment with scheduling real-time and non-real-time flows on a packet-by-packet basis with sub millisecond latency. However, since scheduling was performed in software, there was an overhead on the order of 100-200ms for transmission and reception of ACKs. One option to reduce this overhead is by using the WiMAC architecture presented in Chapter 3. By using the architecture, we can schedule packets on the FPGA without losing flexibility on the scheduling policies. In addition to that, the architecture is extensible, and new mechanisms can be added to WiMAC to rapidly prototype MAC protocols in the future. Finally, I have presented a MAC protocol for mmWave that takes into account the different propagation characteristics of transmissions in these bands, which include directionality of transmission and the need to track mobile users in Chapter 4. I have introduced the notion of *topological coherence time*, which is the period where the topology of the network can be assumed to be stationary. This value is highly dependent on the mobility of the users and the frequency of communication. Simulation results have been presented to give a sense of how this protocol might perform, but building a platform to implement the protocol is still necessary for evaluating the performance of the protocol.

REFERENCES

[1] GSMA, G. Intelligence, GSMA, and G. Intelligence, "Understanding 5G: Perspectives on future technological advancements in mobile," *GSMA Intelligence Understanding 5G*, no. December, pp. 3–15, 2014.

[2] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, "Millimeter wave mobile communications for 5G cellular: It will work!," *IEEE Access*, vol. 1, pp. 335–349, 2013.

[3] R. W. Heath, N. Gonzalez-Prelcic, S. Rangan, W. Roh, and A. M. Sayeed, "An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, pp. 436–453, 2016.

[4] E. Larsson, O. Edfors, F. Tufvesson, and T. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, 2014.

[5] F. Rusek, D. Persson, B. K. Lau, E. G. Larsson, T. L. Marzetta, O. Edfors, and F. Tufvesson, "Scaling up MIMO : Opportunities and challenges with very large arrays," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 40–60, 2013.

[6] Y. Saito, Y. Kishiyama, A. Benjebbour, T. Nakamura, A. Li, and K. Higuchi, "Non-orthogonal multiple access (NOMA) for cellular future radio access," in *IEEE Vehicular Technology Conference*, 2013.

[7] R. Razavi, M. Dianati, and M. A. Imran, "Non-Orthogonal Multiple Access (NOMA) for future radio access," in *5G Mobile Communications*, pp. 135–163, 2016.

[8] J. Duplicy, B. Badic, R. Balraj, R. Ghaffar, P. Horváth, F. Kaltenberger, R. Knopp, I. Z. Kovács, H. T. Nguyen, D. Tandur, and G. Vivier, "MU-MIMO in LTE systems," *Eurasip Journal on Wireless Communications and Networking*, vol. 2011, 2011.

[9] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. of IEEE INFOCOM*, vol. 3, pp. 1567–1576, 2002.

[10] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proc. of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 171–180, 2003.

[11] H.-S. So, J. Walrand, and J. Mo, "McMAC: A parallel rendezvous multi-channel MAC protocol," in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 334–339, March 2007.

[12] W. Hu, H. Yousefi'zadeh, and X. Li, "Load Adaptive MAC: A hybrid MAC protocol for MIMO SDR MANETs," *IEEE Transactions on Wireless Communications*, vol. 10, pp. 3924–3933, November 2011.

[13] L. Tang, Y. Sun, O. Gurewitz, and D. Johnson, "PW-MAC: An energy-efficient predictive-wakeup MAC protocol for wireless sensor networks," in *Proc. of IEEE INFOCOM*, pp. 1305–1313, April 2011.

[14] I.-H. Hou and P. Kumar, "Utility maximization for delay constrained qos in wireless," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[15] J. J. Jaramillo and R. Srikant, "Optimal Scheduling for Fair Resource Allocation in Ad Hoc Networks With Elastic and Inelastic Traffic," *IEEE/ACM Transactions on Networking*, vol. 4, no. 19, pp. 1125–1136, 2011.

[16] I.-H. Hou, "Broadcasting delay-constrained traffic over unreliable wireless links with network coding," *IEEE/ACM Transactions on Networking*, vol. 23, no. 3,

pp. 728–740, 2015.

[17] K. S. Kim, C.-p. Li, and E. Modiano, "Scheduling multicast traffic with deadlines in wireless networks," in *INFOCOM, 2014 Proceedings IEEE*, pp. 2193–2201, IEEE, 2014.

[18] Open Networking Foundation, "SDN Architecture Overview," *Onf*, pp. 1–5, 2013.

[19] A. Malik, J. Qadir, B. Ahmad, K. L. Alvin Yau, and U. Ullah, "QoS in IEEE 802.11-based wireless networks: A contemporary review," 2015.

[20] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, "Network Slicing for 5G: Challenges and Opportunities," *IEEE Internet Computing*, vol. 21, no. 5, pp. 20–27, 2017.

[21] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," 2017.

[22] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: Enabling enterprises' own software-defined cellular networks," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 146–153, 2016.

[23] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[24] F. Douglis and O. Krieger, "Virtualization," 2013.

[25] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[26] "NSF Workshop on Ultra-Low Latency Wireless Networks," November 2016.

[27] ITU-T, "The Tactile Internet," August 2014.

[28] H. Holma and A. Toskala, *LTE for UMTS: Evolution to LTE-advanced*. John Wiley & Sons, 2011.

[29] M. Laner, P. Svoboda, P. Romirer-Maierhofer, N. Nikaein, F. Ricciato, and M. Rupp, "A comparison between one-way delays in operating HSPA and LTE networks," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*, pp. 286–292, IEEE, 2012.

[30] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, and T. Moscibroda, "Characterizing and improving WiFi latency in large-scale operational networks," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 347–360, ACM, 2016.

[31] I. H. Hou, V. Borkar, and P. R. Kumar, "A Theory of QoS for Wireless," in *IEEE INFOCOM 2009*, pp. 486–494, April 2009.

[32] R. Singh and P. R. Kumar, "Decentralized throughput maximizing policies for deadline-constrained wireless networks," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 54rd IEEE Conference on Decision and Control,CDC 2015, pp. 3759–3766, 2015.

[33] I.-H. Hou and P. Kumar, *Packets with Deadlines: A Framework for Real-Time Wireless Networks*, vol. 6. 2013.

[34] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling mac protocol implementations on software-defined radios.," in *NSDI*, vol. 9, pp. 91–105, 2009.

[35] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova, and P. Mahonen, "Decomposable mac framework for highly flexible and adaptable mac realizations," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, pp. 1–2, IEEE, 2010.

[36] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: Practical differential backlog congestion control for wireless networks," in *INFOCOM 2009, IEEE*, pp. 262–270, IEEE, 2009.

[37] R. Laufer, T. Salonidis, H. Lundgren, and P. Le Guyadec, "XPRESS: A cross-layer backpressure architecture for wireless multi-hop networks," in *Proceedings of the 17th annual international conference on Mobile computing and networking*, pp. 49–60, ACM, 2011.

[38] J. Ryu, V. Bhargava, N. Paine, and S. Shakkottai, "Back-pressure routing and rate control for ICNs," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pp. 365–376, ACM, 2010.

[39] J. Lee, H. Lee, Y. Yi, S. Chong, E. W. Knightly, and M. Chiang, "Making 802.11 DCF near-optimal: Design, implementation, and evaluation," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1745–1758, 2016.

[40] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pp. 140–149, IEEE, 2013.

[41] O. N. Yilmaz, Y.-P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, and J. Sachs, "Analysis of ultra-reliable and low-latency 5G communication for a factory automation use case," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*, pp. 1190–1195, IEEE, 2015.

[42] S. Yoshioka, Y. Inoue, S. Suyama, Y. Kishiyama, Y. Okumura, J. Kepler, and M. Cudak, "Field experimental evaluation of beamtracking and latency performance for 5G mmWave radio access in outdoor mobile environment," in *Personal, Indoor, and*

Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on, pp. 1–6, IEEE, 2016.

[43] J. Pilz, M. Mehlhose, T. Wirth, D. Wieruch, B. Holfeld, and T. Haustein, "A Tactile Internet demonstration: 1ms ultra low delay for wireless communications towards 5G," in *Proc. of INFOCOM WKSHPS*, pp. 862–863, IEEE, 2016.

[44] IEEE, "IEEE Standard for Local and metropolitan area networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012*, 2012.

[45] S. Yau, L. Ge, P.-C. Hsieh, I. Hou, S. Cui, P. Kumar, A. Ekbal, N. Kundargi, *et al.*, "Wimac: Rapid implementation platform for user definable mac protocols through separation," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 109–110, ACM, 2015.

[46] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless mac processors: Programming mac protocols on commodity hardware," in *INFOCOM, 2012 Proceedings IEEE*, pp. 1269–1277, IEEE, 2012.

[47] "LabVIEW Communications 802.11 Application Framework 1.1," 2015.

[48] "USRP-2953," 2014.

[49] "Feasibility study on licensed-assisted access to unlicensed spectrum." `http://www.3gpp.org/dynareport/36889.htm`, June 2014.

[50] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar 2008.

[51] L. Kleinrock and F. Tobagi, "Packet switching in radio channels: Part i–Carrier Sense Multiple-Access modes and their throughput-delay characteristics," *IEEE Transactions on Communications*, vol. 23, pp. 1400–1416, Dec 1975.

[52] Z. Zeng, Y. Gao, K. Tan, and P. R. Kumar, "CHAIN: Introducing minimum controlled coordination into random access MAC," in *Proc. IEEE INFOCOM*, pp. 2669–2677, 2011.

[53] M. Markakis, E. Modiano, and J. Tsitsiklis, "Max-Weight scheduling in queueing networks with heavy-tailed traffic," *IEEE/ACM Transactions on Networking*, vol. 22, pp. 257–270, Feb 2014.

[54] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald, "SoftMAC–flexible wireless research platform," in *the 4th ACM Workshop on Hot Topics in Networks*, 2005.

[55] A. Sharma, M. Tiwari, and H. Zheng, "Madmac: Building a reconfiguration radio testbed using commodity 802.11 hardware," in *1st IEEE Workshop on Networking Technologies for Software Defined Radio Networks*, pp. 78–83, Sept 2006.

[56] M.-H. Lu, P. Steenkiste, and T. Chen, "FlexMAC: A wireless protocol development and evaluation platform based on commodity hardware," in *Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, pp. 105–106, 2008.

[57] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. Sicker, and D. Grunwald, "MultiMAC - an adaptive MAC framework for dynamic radio networking," in *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, pp. 548–555, Nov 2005.

[58] "GNU Radio." http://gnuradio.org/redmine/projects/gnuradio/wiki.

[59] T. Schmid, O. Sekkat, and M. B. Srivastava, "An experimental study of network performance impact of increased latency in software defined radios," in *Proc. of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WinTECH)*, pp. 59–66, 2007.

[60] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker, "Sora: High performance software radio using general purpose multi-core processors," in *Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 75–90, 2009.

[61] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, "WARP: A flexible platform for clean-slate wireless medium access protocol design," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, pp. 56–58, Jan 2008.

[62] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, "Airblue: A system for cross-layer wireless protocol development," in *Proc. of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 4:1–4:11, 2010.

[63] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC protocol implementations on software-defined radios," in *Proc. 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 91–105, 2009.

[64] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova, and P. Mahonen, "Decomposable MAC framework for highly flexible and adaptable MAC realizations," in *Proc. IEEE Symposium on New Frontiers in Dynamic Spectrum*, pp. 1–2, 2010.

[65] X. Zhang, J. Ansari, G. Yang, and P. Mahonen, "TRUMP: Efficient and flexible realization of medium access control protocols for wireless networks," *to appear in IEEE Transactions on Mobile Computing*, vol. PP, 2015.

[66] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity hardware," in *Proc. IEEE INFOCOM*, pp. 1269–1277, 2012.

[67] I.-H. Hou and P. R. Kumar, "Utility maximization for delay constrained QoS in wireless," in *Proc. IEEE INFOCOM*, pp. 1–9, 2010.

[68] Z. Zeng, Y. Gao, and P. R. Kumar, "SOFA: A sleep-optimal fair-attention scheduler for the power-saving mode of WLANs," in *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pp. 87–98, 2011.

[69] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an international standard for wireless access in vehicular environments," in *IEEE Vehicular Technology Conference (VTC)*, pp. 2036–2040, May 2008.

[70] I.-H. Hou, V. Borkar, and P. Kumar, "A theory of QoS for wireless," in *Proc. of IEEE INFOCOM*, pp. 486–494, April 2009.

[71] S. Shakkottai, T. Rappaport, and P. Karlsson, "Cross-layer design for wireless networks," *IEEE Communications Magazine*, vol. 41, pp. 74–80, Oct 2003.

[72] V. Kawadia and P. Kumar, "A cautionary perspective on cross-layer design," *IEEE Wireless Communications*, vol. 12, pp. 3–11, Feb 2005.

[73] T. ElBatt and A. Ephremides, "Joint scheduling and power control for wireless ad hoc networks," *IEEE Transactions on Wireless Communications*, vol. 3, pp. 74–85, Jan 2004.

[74] S. Kumar, D. Cifuentes, S. Gollakota, and D. Katabi, "Bringing cross-layer MIMO to today's wireless LANs," in *Proc. of the ACM SIGCOMM*, pp. 387–398, 2013.

[75] 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*, vol. 37, pp. 1936–1948, Dec 1992.

[76] M. Neely, E. Modiano, and C. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 89–103, Jan 2005.

[77] X. Lin, N. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 1452–1463, Aug 2006.

[78] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and MAC for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 1514–1524, Aug 2006.

[79] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, pp. 1–144, Apr 2006.

[80] M. Neely, "Order optimal delay for opportunistic scheduling in multi-user wireless uplinks and downlinks," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 1188–1199, Oct 2008.

[81] A. Ganti, E. Modiano, and J. Tsitsiklis, "Optimal transmission scheduling in symmetric communication models with intermittent connectivity," *IEEE Transactions on Information Theory*, vol. 53, pp. 998–1008, March 2007.

[82] K.-H. Chou and W. Lin, "Performance analysis of packet aggregation for IEEE 802.11 PCF MAC-based wireless networks," *IEEE Transactions on Wireless Com-*

munications, vol. 12, pp. 1441–1447, April 2013.

[83] A. Eryilmaz, R. Srikant, and J. Perkins, "Stable scheduling policies for fading wireless channels," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 411–424, April 2005.

[84] Y. S. Satchidanandan B., E. A. Kumar P. R., Aziz A., and K. N., "Trackmac: An ieee 802.11ad-compatible beam tracking-based mac protocol for 5g millimeter-wave local area networks," in *10th International Conference on Communication Systems and Networks (COMSNETS)*, 2018.

[85] J. M. Gilbert, C. H. Doan, S. Emami, and C. B. Shung, "A 4-Gbps uncompressed wireless HD A/V transceiver chipset," in *IEEE Micro*, vol. 28, pp. 56–64, 2008.

[86] S. Piersanti, L. A. Annoni, and D. Cassioli, "Millimeter waves channel measurements and path loss models," in *2012 IEEE International Conference on Communications (ICC)*, pp. 4552–4556, 2012.

[87] Z. Pi and F. Khan, "An introduction to millimeter-wave mobile broadband systems," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 101–107, 2011.

[88] E. Ben-Dor, T. S. Rappaport, Y. Qiao, and S. J. Lauffenburger, "Millimeter-wave 60 GHz outdoor and vehicle AOA propagation measurements using a broadband channel sounder," in *GLOBECOM - IEEE Global Telecommunications Conference*, 2011.

[89] T. S. Rappaport, E. Ben-Dor, J. N. Murdock, and Y. Qiao, "38 GHz and 60 GHz angle-dependent propagation for cellular & peer-to-peer wireless communications," in *IEEE International Conference on Communications*, pp. 4568–4573, 2012.

[90] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, "Millimeter wave mobile communications for

5G cellular: It will work!," *IEEE Access*, vol. 1, pp. 335–349, 2013.

[91] T. E. Bogale and L. B. Le, "Massive MIMO and mmWave for 5G Wireless HetNet: Potential Benefits and Challenges," *IEEE Vehicular Technology Magazine*, vol. 11, no. 1, pp. 64–75, 2016.

[92] G. R. Maccartney and T. S. Rappaport, "73 GHz millimeter wave propagation measurements for outdoor urban mobile and backhaul communications in New York City," in *2014 IEEE International Conference on Communications, ICC 2014*, pp. 4862–4867, 2014.

[93] T. S. Rappaport, G. R. MacCartney, M. K. Samimi, and S. Sun, "Wideband millimeter-wave propagation measurements and channel models for future wireless communication system design," *IEEE Transactions on Communications*, vol. 63, no. 9, pp. 3029–3056, 2015.

[94] S. Singh, F. Ziliotto, U. Madhow, E. M. Belding, and M. Rodwell, "Blockage and directivity in 60 GHz wireless personal area networks: From cross-layer model to multihop MAC design," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 8, pp. 1400–1413, 2009.

[95] S. Singh, R. Mudumbai, and U. Madhow, "Distributed coordination with deaf neighbors: Efficient medium access for 60 GHz mesh networks," *Proceedings - IEEE INFOCOM*, 2010.

[96] M. Takai, J. Martin, R. Bagrodia, and A. Ren, "Directional carrier sensing for directional antennas in mobile ad hoc networks," *International symposium on Mobile ad hoc networking & computing*, vol. 3, pp. 183–193, 2002.

[97] V. Shankarkumar and N. Vaidya, "Medium access control protocols using directional antennas in ad hoc networks," in *Proceedings IEEE INFOCOM 2000. Con-*

ference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), pp. 13–21, 2000.

[98] Y.-B. K. Y.-B. Ko, V. Shankarkumar, and N. Vaidya, "Medium access control protocols using directional antennas in ad hoc networks," *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 1, no. c, pp. 13–21, 2000.

[99] A. Nasipuri, S. Ye, J. You, and R. E. Hiromoto, "A MAC Protocol for Mobile Ad Hoc Networks Using Directional Antennas," *Proc. of IEEE INFOCOM*, pp. 1214–1219, 2000.

[100] R. R. Choudhury, X. Yang, N. H. Vaidya, and R. Ramanathan, "Using directional antennas for medium access control in ad hoc networks," *Proceedings of the 8th annual international conference on Mobile computing and networking - MobiCom '02*, p. 59, 2002.

[101] M. Gong and R. Stacey, "A directional CSMA/CA protocol for mmWave wireless PANs," *Wireless . . .* , pp. 1–6, 2010.

[102] M. X. Gong, D. Akhmetov, R. Want, and S. Mao, "Directional CSMA/CA protocol with spatial reuse for mmWave wireless networks," *GLOBECOM - IEEE Global Telecommunications Conference*, pp. 1–5, 2010.

[103] Q. Chen, X. Peng, J. Yang, and F. Chin, "Spatial reuse strategy in mmWave WPANs with directional antennas," *GLOBECOM - IEEE Global Telecommunications Conference*, pp. 5392–5397, 2012.

[104] C. S. Sum, Z. Lan, R. Funada, J. Wang, T. Baykas, M. A. Rahman, and H. Harada, "Virtual time-slot allocation scheme for throughput enhancement in a millimeter-wave multi-Gbps WPAN system," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 8, pp. 1379–1389, 2009.

[105] I. K. Son, S. Mao, M. X. Gong, and Y. Li, "On frame-based scheduling for directional mmWave WPANs," in *Proceedings - IEEE INFOCOM*, pp. 2149–2157, 2012.

[106] T. Korakis, G. Jakllari, and L. Tassiulas, "A {MAC} protocol for full exploitation of directional antennas in ad-hoc wireless networks," *ACM Int. Symposium on Mobile Ad-Hoc Networking & Computing*, pp. 98–107, 2003.

[107] E. Shihab, S. Member, L. Cai, J. Pan, and S. Member, "A Distributed Asnchronous Directional-to-Directional MAC Protocol for Wireless Ad Hoc Networks," *Ieee Tvt*, vol. 58, no. 9, pp. 5124–5134, 2009.

[108] H. Shokri-Ghadikolaei, C. Fischione, G. Fodor, P. Popovski, and M. Zorzi, "Millimeter wave cellular networks: A MAC layer perspective," *IEEE Transactions on Communications*, vol. 63, no. 10, pp. 3437–3458, 2015.

[109] T. Korakis, G. Jakllari, and L. Tassiulas, "CDR-MAC: A protocol for full exploitation of directional antennas in ad hoc wireless networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 2, pp. 145–155, 2008.

[110] Y. Niu, Y. Li, D. Jin, L. Su, and D. Wu, "Blockage robust and efficient scheduling for directional mmwave wpans," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 2, pp. 728–742, 2015.

[111] "ISO/IEC/IEEE International Standard for Information technology–Telecommunications and information exchange between systems–Local and

metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (," pp. 1–634, 2014.

[112] Q. Chen, J. Tang, D. Wong, X. Peng, and Y. Zhang, "Directional cooperative MAC protocol design and performance analysis for IEEE 802.11ad WLANs," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 6, pp. 2667–2677, 2013.

[113] P. F. Smulders and A. G. Wagemans, "Frequency-domain measurement of the millimeter wave indoor radio channel," *IEEE Transactions on Instrumentation and Measurement*, vol. 44, no. 6, pp. 1017–1022, 1995.

[114] R. Kimura, R. Funada, Y. Nishiguchi, M. Lei, T. Baykas, C.-S. Sum, J. Wang, A. Rahman, Y. Shoji, H. Harada, *et al.*, "Golay sequence aided channel estimation for millimeter-wave wpan systems," in *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, pp. 1–5, IEEE, 2008.

[115] T. Nitsche, C. Cordeiro, A. B. Flores, E. W. Knightly, E. Perahia, and J. C. Widmer, "IEEE 802.11ad: Directional 60 GHz communication for multi-Gigabit-per-second Wi-Fi [Invited Paper]," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 132–141, 2014.

[116] T. Nitsche, C. Cordeiro, A. B. Flores, E. W. Knightly, E. Perahia, and I. N. P. Aper, "Radio Communications IEEE 802 . 11ad : Directional 60 GHz Communication for," no. December, pp. 132–141, 2014.