Energy-Aware Computation in Mobile Devices

BY

AJITA SINGH
B.E., Birla Institute of Technology, Mesra, India, 2013

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Hulya Seferoglu, Chair and Advisor
Daniela Tuninetti
Rashid Ansari

To my family.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof Hulya Seferoglu for her unwavering support and invaluable guidance. This thesis would not have been possible without her encouragement and counsel. I would also like to thank my committee members, Prof Daniela Tuninetti and Prof Rashid Ansari for their support and insightful suggestions.

I would also like to thank members of the Networking Research Lab namely, Yuxuan Xing for his help and suggestions while I was working on my thesis. I would also like to thank my parents and my sister for always being there for me, I appreciate your constant care and support.

AS

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

ACK	Acknowledgement
AP	Access Point
ARQ	Automatic Repeat Request
AVC	Advanced Video Coding
BSS	Basic Service Set
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
D2D	Device to Device
DCF	Distributed Coordination Function
DHCP	Dynamic Host Control Protocol
DIFS	Distributed Inter-frame Space
FIN	Finish
GO	Group Owner
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MSS	Maximum Segment Size
NoA	Notice of Absence
P2P	Peer to Peer
QoS	Quality of Service
RTS	Request to Send
RTT	Round Trip Time

LIST OF ABBREVIATIONS (Continued)

SIFS	Short Inter-frame Space
SSID	Service Set Identifier
SYN	Synchronize
TCP	Transmission Control Protocol
WLAN	Wireless Local Area Network
WPS	Wi-Fi Protected Setup

SUMMARY

The booming of mobile devices and applications has a significant social and economic impact. However, new data intensive applications, which are continuously emerging in daily routines of mobile users, continuously increase the demand for wireless resources. Although bandwidth is traditionally considered as the primary scarce resource in wireless networks, the developments in communication theory shifts the focus from bandwidth to other scarce resources including processing power and energy. Thus, it is crucial to develop new networking mechanisms by taking into account the processing power and energy as bottlenecks.

In this thesis, our primary goal is to overcome the bottlenecks created by processing power and energy. To achieve this goal, we first analyze and evaluate the effects of processing power and energy on data rates in a real testbed. Motivated by our initial observations on the impact of processing power and energy limitations on data transmission rates, we develop an energy-aware computation (EaC) framework. The crucial components of EaC are decoder, energy filter, and cooperation among mobile devices. The decoder part deals with the processing power limitation of mobile devices. The energy filter helps us incorporate the energy consumption requirements of mobile devices. Cooperation among mobile devices eliminate and distribute the processing power and energy bottlenecks over multiple mobile devices.

The integral part of this thesis is to evaluate the EaC framework in a practical setup. Thus, we created a testbed consisting of Android operating system based mobile devices, where these mobile devices communicate via WiFi Direct based device-to-device links. We implemented the proposed EaC framework in this testbed. Finally, we evaluated the EaC framework as compared the baselines. The experimental results demonstrate that EaC significantly improves data transmission rates by overcoming processing power and energy bottlenecks.

1 INTRODUCTION

1.1 Motivation

The rapidly increasing number of mobile device users and its applications have made a significant impact on the current wireless networks. The number of worldwide mobile users, including both business and consumers have already reached over 5.6 billion (in 2014) and is expected to rise to 6.2 billion by the end of 2018 [1]. This poses a huge challenge for current wireless networks, so new methodologies should be developed to overcome this challenge. Furthermore, the need for data intensive applications has increased significantly. This, along with the increased number of mobile users has put stress on how significantly the demand for wireless resources has risen in the past few years [2], [3].

In this setup, although bandwidth is traditionally considered the primary scarce resource in today's wireless networks, other wireless resources such as processing power, energy, and memory of mobile devices could become bottlenecks in mobile networks. Thus, in this thesis, we specifically focus on processing power and energy as scarce resources and we aim to develop efficient mechanisms to effectively utilize these scarce resources.

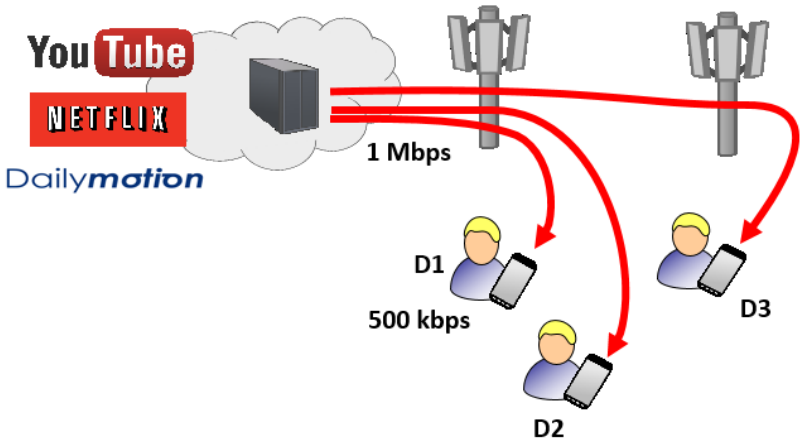


Figure 1: Example setup, where D1 receives data with rate 1Mbps, but decodes with rate 500kbps

Let us consider the impact of processing power via an example. Consider a mobile device D_1 in Figure 1 with a Wi-Fi or cellular link of rate 1 Mbps, and assume that D_1 is receiving video and decodes it

with rate of 500kbps. This would limit the streaming rate of to 500 kbps. Apart from decoding, there are various other computationally intensive tasks that is couple with data transmission such as error correction, packet randomization and network coding of data at the source introduce computational complexities up to $O(n^3)$, [5], [6]. Moreover, H.264/AVC decoders increase the computational complexity when higher quality guarantees are required by the user [7], [8].

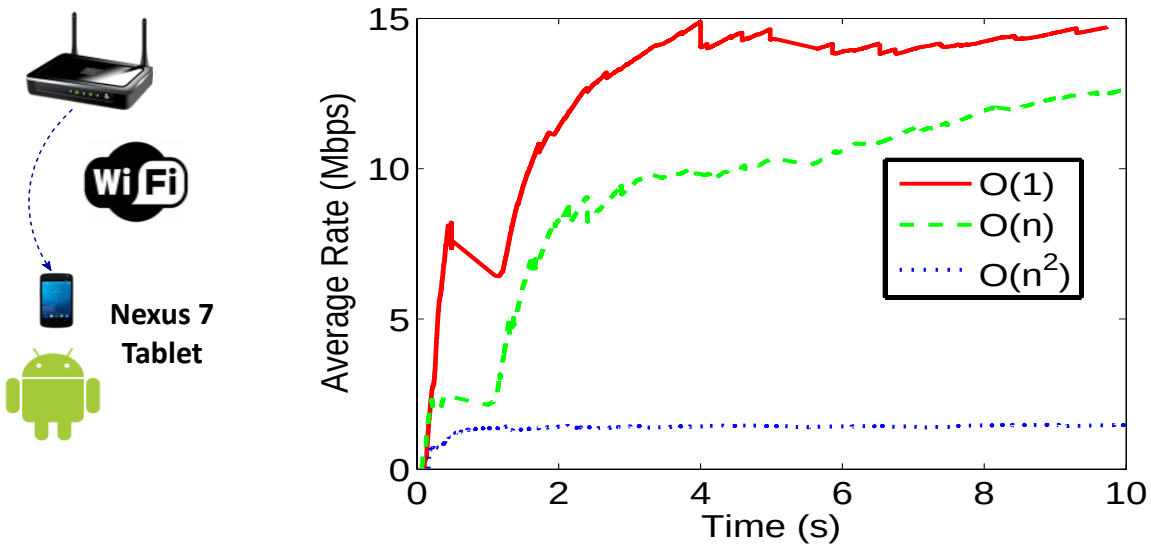


Figure 2: Experiment setup is on the left side, and the data rate versus time graph for varying computational complexities such as *O(1), O(n) and* $O(n^2)$ are shown on the right hand side.

In order to understand the impact of computational complexity on today's mobile devices, we create a prototype shown in the left figure of Figure 2. In this setup, a mobile device receives data from an access point, where the mobile device is an Android operating system (OS) based Nexus 7 tablets. The specific version of the Android OS is Android Lollipop 5.1.1. The device has 16GB storage, 2GB RAM, Qualcomm Snapdragon S4 Pro, 1.5.GHZ COU, and Adreno 320, 400MHz GPU. Packet size is 500B. In this experiment, after receiving the packets, the mobile device performs operations with complexities of *O(1)*, *O(n)*, and $O(n^2)$ above the transport layer (TCP), where *n* is the packet size, and the operations we perform are counting the bytes in the packets. In particular, *O(1)*, *O(n)*, and $O(n^2)$ correspond to (i) no counting, (ii) counting every byte in a packet once, and (iii) counting every byte in a packet *n* times, respectively. We

demonstrate in the right hand side of Figure 2 the received rate at the mobile device (note that this is the rate we measure at the mobile device after performing computations) versus time. As seen, when the computational complexity increases, the average rate measured at the mobile device (after computation) decreases. Figure 2 clearly indicates the extent up to which decoding complexities can affect the transmission rates in mobile devices. Our goal is in this thesis is to develop mechanisms to overcome the bottleneck of processing power.

The second challenge is the energy limitation of mobile devices transmission/reception of data. As similar to the processing power bottleneck discussed above, energy could be bottleneck during data transmission. In this context, our goal is to develop mechanisms that ensure that mobile devices have enough energy to transmit, receive and process the data.

1.2 Overview

The goal of this thesis is to understand the impact of processing power and energy consumption bottlenecks on mobile wireless networks. Towards this goal, we have developed a networking mechanism designed specifically by taking into account processing power and energy bottlenecks, and implemented this mechanism in a real testbed.

The crucial components of the developed framework are decoder, energy filter, and cooperation among mobile devices. The decoder helps us understand how the processing power can vary, depending on the type of application that is being used. The energy filter helps us in incorporating the needs of the user along with the decoder to analyze the battery that is consumed for different computational complexities. Cooperation among mobile devices helps us to eliminate and distribute the processing power and energy bottlenecks over multiple mobile devices.

Our testbed is developed using Android operating system based mobile devices. We implemented Wi-Fi Direct [11] to create device-to-device (D2D) connections which are used to connect multiple mobile devices in close proximity.

1.3 Organization of the Thesis

The thesis is organized as follows: Chapter 2 discusses in detail, the related work and previous publications that have focused on maximization of resources like bandwidth. Chapter 3 contains background information of the current wireless technologies: Wi-Fi and Wi-Fi Direct. It also discusses transport layer protocols (TCP) and socket programming. Chapter 4 explains the Energy-Aware Computation (EaC) system model that we have composed, consisting of a source, wireless channel and the mobile device. We have also formulated the problem statement in this chapter. The implementation details of our system model has been outlined in Chapter 5. The simulation results of the proposed system model is presented in Chapter 6, and finally Chapter 7 summarizes our conclusion to the thesis.

2 RELATED WORK

The work in this thesis is based mainly upon Wi-Fi Direct connections, along with maximization of resources used in processing power and battery consumption. In particular, D2D group formations are used for several data streaming applications for various purposes including, but not restricted to, cooperative video streaming over mobile devices, offloading cellular networks and distribution of content among mobile devices, as discussed below.

A peer-to-peer cooperation scheme was introduced in [16] with focus on power reduction. In this paper, mobile devices were connected to the Internet through a wireless AP, all of them requesting the same data from a server. The server would then distribute the data amongst all devices, who would further exchange data with each other through Bluetooth technology. On similar lines, [17] introduced a scheme where mobile devices collaboratively helped each other to recover lost packets by broadcasting to its neighbors. This scheme, also known as BOPPER (broadcasting with peer-to-peer error recovery) achieved high scalability and low recovery delay.

An optimal and scalable distribution of dynamic content was first introduced in [18]. It utilized and allocated the bandwidth optimally, making sure that the content received by every user was 'fresh'. This paper ensured that such an optimal system would work even if the total bandwidth of the service provider remains fixed, ensuring both optimality and scalability. A case study on information delivery in Mobile Social Networks (MoSoNets) was performed in [19], where opportunistic communications were exploited to facilitate the information dissemination and reduce the amount of cellular traffic. A particular target set was provided with information by the content service providers, after which the information was further propagated amongst all the subscribed users using opportunistic communications, thereby minimizing cellular data traffic.

Content dissemination amongst users led to a novel social-based forwarding algorithm called BUBBLE [20] which utilized real human mobility traces to enhance content delivery performance. This

paper focused on the social structure and interaction of users for its algorithm, unlike the previous methods that relied on building and updating routing tables to cope with dynamic network conditions. Opportunistic networks is another technology which allow content sharing between mobile users without requirement of any pre-existing Internet infrastructure. In [21], a protocol named HiBOp exploits the framework and represents the user's behavior and social relations through context information. The user then uses this information to drive the forwarding process. HiBOp is compared with other solutions to show that a context-aware approach is much more efficient for forwarding in opportunistic networks.

We have attempted to examine in particular the bottleneck created by processing power and battery consumption of mobile devices in this thesis, and remove it through cooperation amongst devices in [14]. An increased interest in computing using mobile devices by exploiting connectivity among mobile devices is seen in [22]. Such an approach suggests that devices can be used together to process certain tasks, which turns out to be a cheaper alternative to remote clouds. This has led to interesting work specifically towards collaborative cloud computing.

'Transient clouds', as introduced in [23] allows nearby devices to form ad hoc networks and provide various functionalities of the devices and their social awareness, which cannot be provided as efficiently by the traditional clouds. This paper showed the efficiency of their algorithm by implementation on Android devices, using the Wi-Fi Direct framework. In the attempt to converge cloud computing with mobile computing, [24] uses virtual machine technology to reconcile the tradeoffs between centralization of cloud computing and decentralization of mobile computing. The paper yielded a transient PC computing model that preserved centralization benefits without sacrificing mobility or usability. Furthermore, [25] discusses a future wherein mobile devices are capable of forming mobile clouds, or 'mClouds' to accomplish tasks without relying on the backend communication. In this thesis, our focus is on processing power and energy of mobile devices. Our resource allocation mechanism shares similar flavors with [26], [27], and [28], but fundamentally different than these works as we explicitly design resource allocation mechanism by taking into account processing power, energy and cooperation among real mobile devices.

3 BACKGROUND

3.1 Wi-Fi: 802.11 Wireless LANs

Wireless LANs [9] have become increasingly popular in our everyday lives: our homes, offices and public places are being equipped with them to connect computers, smart phones, tablets and other wireless devices. The main wireless LAN standard which is in great use today is 802.11, also called Wi-Fi. There are various 802.11 standards for wireless LAN technology, including 802.11b, 802.11a and 802.11g. A summary of these standards, with their main characteristics is given in Table 1 [15].

All three 802.11 standards have major differences in the physical layer. The 802.11b wireless LAN competes for frequency spectrum with 2.4 GHz phones and microwave ovens. 802.11a wireless LANs run at higher rates at higher frequencies. This means that 802.11a has a shorter transmission distance for a given power level and suffer greatly from multipath propagation. On the other hand, 802.11g can not only transmit at data rates of 802.11a, but do so at lower frequencies and are backward compatible with 802.11b.

Standard	Frequency Range	Data Rate
802.11b	2.4-2.485 GHz	Up to 11 Mbps
802.11a	5.1-5.8 GHz	Up to 54 Mbps
802.11g	2.4-2.485 GHz	Up to 54 Mbps

Table 1: SUMMARY OF IEEE 802.11 STANDARDS [15]

In the following sections, we will discuss more in detail about the common characteristics of the various standards of 802.11. Along with the basic architecture, these standards use the same medium access protocol (CSMA/CA) and frame structure for their link-layer frames. Other features such as the 'ad-hoc mode' of wireless LANs will be discussed, and later compared to Wi-Fi Direct.

3.1.1 Architecture

802.11 networks are used in two modes [9]: infrastructure and ad hoc. Figure 3(a) illustrates the infrastructure mode, where the fundamental building block if a basic service set (BSS). A BSS contains

multiple clients, each associated with a central base station known as an Access Point (AP). These APs can connect to other networks through routers and switches.

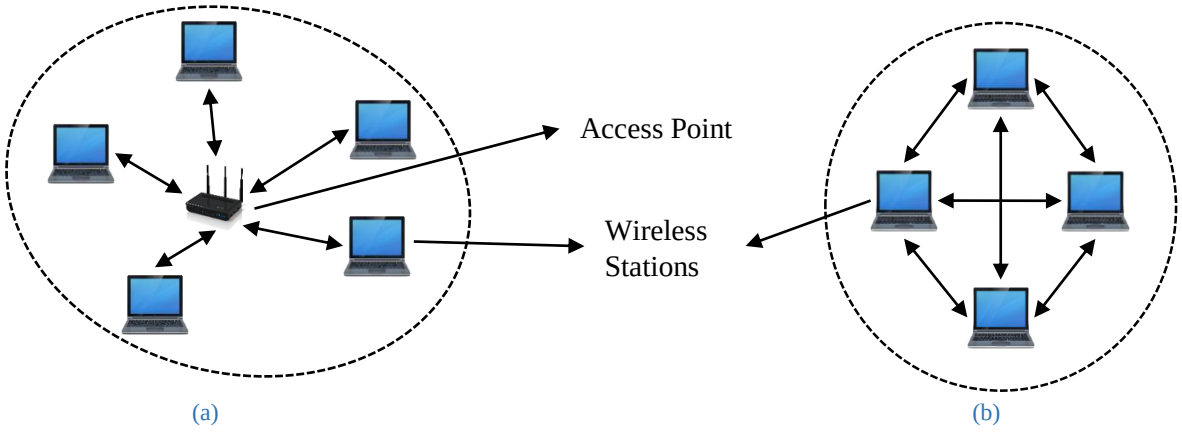


Figure 3: 802.11 Architecture (a) Infrastructure mode (b) Ad-hoc mode

Figure 3(b) shows the ad hoc network formed by various IEEE 802.11 devices. Networks are formed 'on-the-fly' by mobile devices that are in close proximity of each other. Each client can connect to others directly, without an intermediate access point. We will discuss more about this mode later in this chapter and compare it against the Wi-Fi Direct technology.

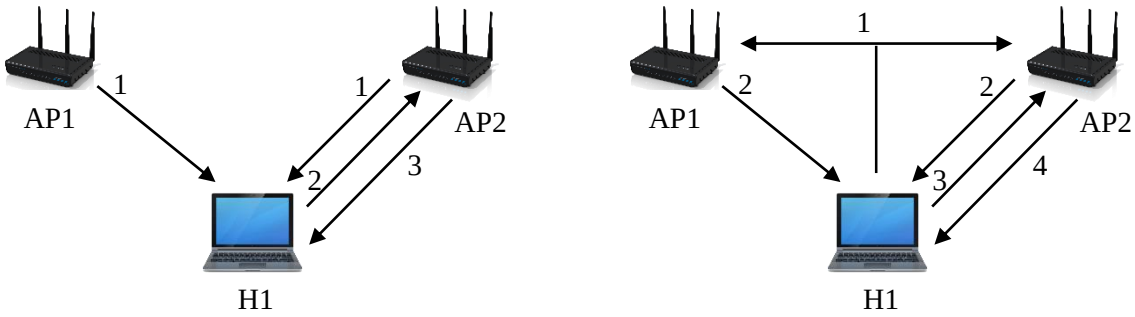
Channels and Association

Every wireless device has to be configured to a wireless AP before it can send or receive any data. When an AP in installed, a one or two word Service Set Identifier (SSID) and channel number is assigned to it. Since 802.11b/g operates in 2.4-2.485 GHz range, it leaves an 85 MHz band which defines 11 partially overlapping channels. Any two channels are non-overlapping if and only if they are separated by four or more channels. Thus, the set of 1, 6 and 11 is the only set of three non-overlapping channels, which can give a maximum speed of 33 Mbps. This is done by installing 3 APs at the same physical location, with channels 1, 6 and 11 respectively, and connecting them with a switch [15].

In order to gain Internet access, a wireless device needs to associate itself with an AP. The 802.11 standard dictates that each AP must periodically send out beacon frames containing its SSID and MAC address. The wireless device will periodically scan all 11 channels, looking out for any beacon frames from

any APs. Even though there isn't a specific algorithm to select which AP to associate with, typically the AP with the highest signal strength is selected.

Passive scanning is defined as the process of scanning channels and listening for beacon frames. A wireless device can also perform active scanning by sending a broadcast probe frame to all APs within the device's range. These two scenarios are shown in Figure 4 [15]. Once an AP is selected, an association request is sent by the wireless device, to which the AP responds with an association frame. It is possible that a wireless device be asked to authenticate itself to the AP. The most common method to implement this is by employing usernames and passwords. An authentication server is used to relay information to the AP, which allows the server to be not only used for multiple APs, but also keeps costs and complexity to the minimum.



(a) Passive Scanning
1. Beacon frames sent from APs
2. Association Request frame sent: H1 to AP2
3. Association Response frame sent: AP2 to H1

(b) Active Scanning
1. Probe Request frame broadcast from H1
2. Probes Response frame sent from APs
3. Association Request frame sent: H1 to AP2
4. Association Response frame sent: AP2 to H1

Figure 4: Active and passive scanning for access points [15]

3.1.2 MAC Protocol

Certain factors that are fundamental to wireless communication, make the 802.11 MAC layer protocol very different from Ethernet. The MAC protocol must be independent of the underlying physical layer and must also be efficient for periodic as well as burst traffic. Since multiple devices or APs may want to transmit data frames at the same time, and possibly over the same channel, an appropriate multiple access protocol is required to coordinate such transmissions. Unlike Ethernet, 802.11 uses a collision-avoidance

technique instead of a collision detection technique, a Distributed Coordinate Function (DCF) known as Carrier Sense multiple access with collision avoidance (CSMA/CA) [9]. Also, due to higher bit error rates of wireless channels, 802.11 uses a link-layer acknowledgment and retransmission (ARQ) scheme.

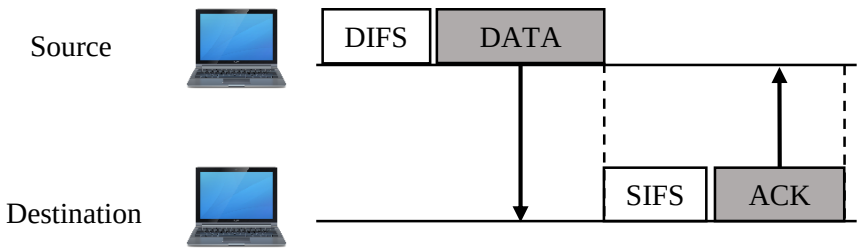


Figure 5: 802.11 using link-layer acknowledgments

As 802.11 does not use a collision detection scheme, a station (wireless device or an AP) will transmit a frame in its entirety every time. This can significantly degrade the performance of the MAC protocol if the likelihood of collisions is high. To avoid such scenarios, 802.11 uses a link-layer acknowledgment scheme consisting of two very short durations of time: SIFS (Short Inter-frame Spacing) and DIFS (Distributed Inter-frame Space) [9].

DIFS is the minimum delay at the source station before starting transmission of a packet, after the channel is expected to be idle and SIFS is the delay at the destination station between end of transmission of a packet and sending an acknowledgement frame to the source station. The concept of CSMA/CA along with the link-layer acknowledgments is shown through the Figure 5 and the following [9]:

1. At the source station:
    a. Channel is busy:
        i. The stations that wish to transmit data will back off for a random time and counts down this value while the channel is sensed idle. While the channel is busy, the counter is frozen.
    b. Channel is idle:
        i. If the value of the counter is zero, the station that wishes to transmit data waits for a period of time equal to DIFS before transmission.

ii. The station waits for an acknowledgement once the frame is transmitted in its entirety.

iii. Once an acknowledgement is received from the destination station, it understands that its frame has been correctly received at the destination station.

iv. If an acknowledgement isn't received within a certain timeout interval, the station enters the back-off phase, assuming that the packet has been lost.

2. At the destination station:

    a. When a frame is correctly received (checked through CRC), it waits for a short period of time equal to SIFS, and then sends back an acknowledgment frame.

In CSMA/CA, the station refrains from transmitting during the count down, even if the channel senses to be idle. This is done as the basic approach to CSMA/CA is collision avoidance. If two stations try to transmit data at the same time on a busy channel, they both will enter into random back-off times. This way, if one transmits before the other, the second station will detect a busy channel and back-off again, thus avoiding collision.

DCF with RTS/CTS

Consider the scenario shown in Figure 6 [15]. We have two wireless stations A and B, such that both are associated with the AP between them, but each station is hidden from the other. Suppose that station A is transmitting a frame and halfway through A's transmission, station B wants to send a frame to the AP. As B will not hear the transmission from A, it will wait a DIFS interval and then transmit the frame, resulting in a collision. This shows the wastage of the channel during the entire period of A's and B's transmission.

The 802.11 protocol avoids this problem by using a short Request to Send (RTS) control frame and a short Clear to Send (CTS) control frame to reserve access to the channel [9]. When a sender wishes

to send a data frame, it will first send an RTS frame to the AP, indicating the total time required to transmit the data frame and the acknowledgment (ACK) frame. When the AP receives the RTS frame, it responds by sending a broadcast CTS frame. Two purposes are solved with this frame: it gives the sender explicit permission to send and also instructs all other stations to not send any data for the reserved duration.

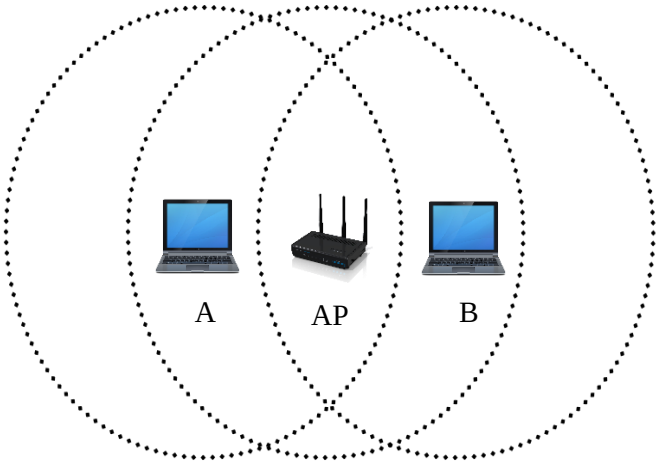


Figure 6: Hidden terminal example: A is hidden from B and vice versa [15]

Thus, in our example, A will first broadcast an RTS frame, which is heard by all the stations in its circle, including the AP. The AP will then respond with the broadcast of the CTS frame, which is heard by all stations within its range.

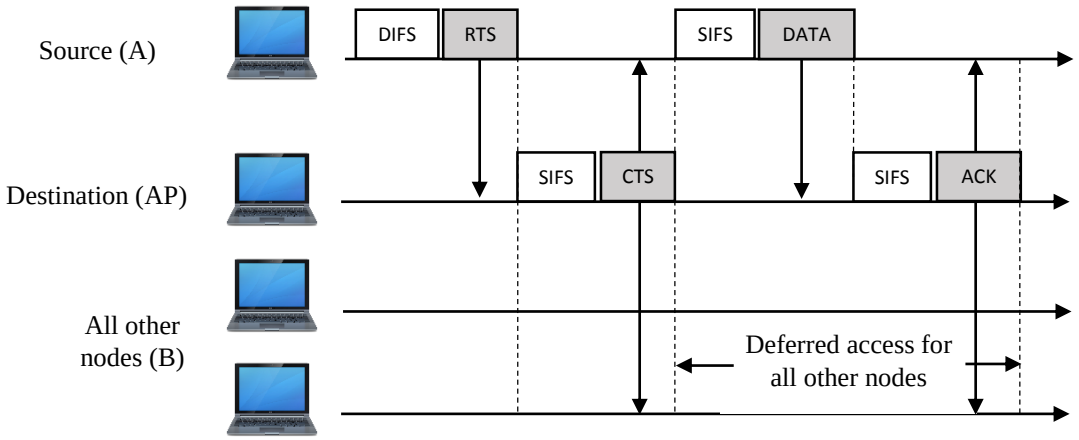


Figure 7: Collision Avoidance using RTS and CTS frames

As station B will have heard the CTS, it will refrain from transmitting for the time specified by the CTS frame. This is portrayed in Figure 7. It is obvious that the delay in increased with the use of RTS and CTS, hence, in practice it is used only when the data frame is longer than a specified RTS threshold.

3.1.3 Frame Structure

The frame structure for 802.11 is shown in Figure 8 [9]. The numbers above each field represents the lengths of the fields in bytes, and the numbers above each of the subfields in the frame control field represents the lengths of the subfields in bits. The entire frame can be broadly classified into four parts, as discussed below.

Payload and CRC fields

The Payload typically consists of an IP datagram or an ARP (Address Resolution Protocol) packet of size fewer than 1,500 bytes (although up to 2,312 bytes are allowed). The 802.11 frame also consists of a 32-bit cyclic redundancy check (CRC) to detect bit errors in the received frame.

Address fields

There are four different address fields in an 802.11 frame, out of which three address fields are required for internetworking purposes – for moving the network-layer datagram from a wireless station through an AP to a router interface. The fourth address field is used when APs forward frames to each other in ad hoc mode. For now, we will only concentrate on the first three address fields. Address 2 is the MAC address of the station that transmits the frame. Address 1 is the MAC address of the wireless station that is to receive the frame. Address 3 contains the MAC address of a router interface which connects to other subnets.

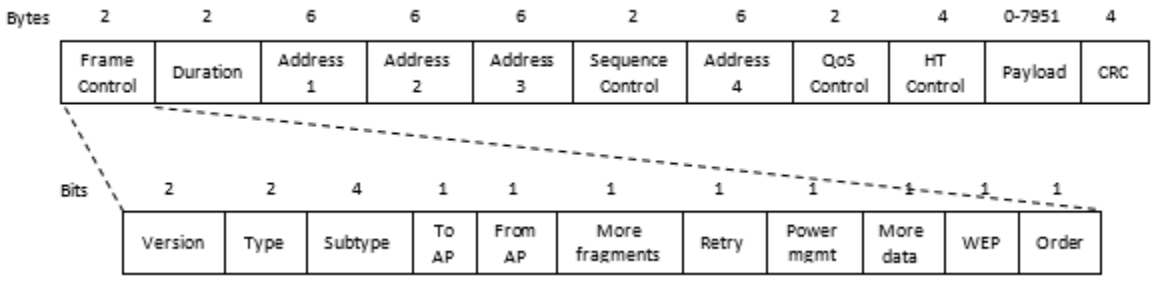


Figure 8: 802.11 Frame Structure [9]

QoS Control and HT Control Fields

The QoS Control field identifies the traffic category or the traffic stream to which the frame belongs to. This 16 bit field contains of five or eight subfields, used to determine QoS and mesh-related information about the frame, which we will not discuss here. HT Control Field is present in QoS Data and management frames. This 32 bit field is further divided into 9 sub fields, which are mainly used for Link Adaptation Control [9].

Sequence Number, Duration and Frame Control fields

The use of sequence numbers in 802.11 allows a receiver to distinguish between a newly received frame and a retransmitted previous frame. Whenever a station correctly receives a frame from another station, it sends back an acknowledgment according to the sequence number. The duration value in the frame gives the time duration for which the channel needs to be reserved, which includes the time to transmit the data frame and the time to transmit an acknowledgment. This value is included for both data frames and for the RTS and CTS frames.

The Frame control field consists of 11 subfields [9] [15]. *Protocol version* allows future versions of 802.11 to operate at the same time in the same cell. The *Type* (data, control or management) and *Subtype* (RTS or CTS) fields for regular data frame without quality of service, are set to 10 and 0000 in binary. The *To AP* and *From AP* frames indicate whether the frame is going to or coming from the network connected to the APs. *More fragments* bit means that more fragments will follow. *Retry* marks a retransmission of a frame sent earlier. The *Power Management* bit indicates that the sender is going into power-save mode. The *More data* bit indicates that the sender has additional frames for the receiver. The *WEP* (Wired Equivalent Privacy) bit indicates that the frame body has been encrypted for security. Finally, the *Order* but tells the receiver that the higher layer expects the sequence of frames to arrive strictly in order.

3.1.4 Wireless Ad Hoc Networks

As mentioned briefly at the beginning of this chapter, ad hoc networks [10] can be formed with 802.11 devices. These decentralized networks do not rely on any pre-existing infrastructure, such as access points in wireless networks. All stations form an Independent Basic Service Set (IBSS). Any other station

that is within the transmission range of another can start communicating. Access Point (APs) are not required unless a station has an ad hoc and a wired network connection.

Every station participates in routing and forwarding data to other stations. These stations are the network and they co-operatively provide the functionality which is normally provided by the infrastructure e.g. routers and switches. Apart from classic routing, ad hoc networks also use flooding for sending data. IEEE 802.11 is normally associated with a single-hop ad hoc network, with the range of stations being up to 100-200 meters. This can be overcome by adding routing mechanisms to forward packets towards the intended destination for multi-hop ad hoc networking. This extends the range of any ad hoc network beyond the transmission radius of the source station.

Common Problems in Wireless Ad Hoc Networks

As stated in [9] [10], there are many problems that can arise in wireless networks, specifically in the ad hoc mode:

1. The channels are unprotected from outside signals
2. The wireless medium has neither absolute nor readily observable boundaries outside of which stations are known to be unable to receive network frames
3. The channels have time-varying and asymmetric propagation properties
4. The wireless medium is significantly less reliable than wired media

Apart from the basic drawbacks of wireless networks, the 802.11 ad hoc mode provides minimal security against any unwanted incoming connections. Ad hoc devices cannot disable their SSID broadcast like infrastructure mode devices can. If any attacker is within the wireless device's range, they can easily connect to it. Furthermore, wireless networking standards such as 802.11g supports only speeds up to 11 Mbps in ad hoc mode. This is a huge step down from the speeds provided in 802.11g infrastructure mode (54 Mbps). Finally, as the number of devices increases in an ad hoc mode, the performance of the networks suffers. Devices may randomly disconnect from time to time and it may get difficult to manage such a network.

Taking into consideration the drawbacks of ad hoc mode of wireless networks, we delve further into the Wi-Fi Direct technology, and how it improves on the shortcomings of the wireless ad hoc mode.

3.2 Wi-Fi Direct

Apart from the ad hoc mode in wireless networks, 802.11z or Tunneled Direct Link Setup (TDLS) [12] also provides direct device to device (D2D) communication, on the condition that all wireless stations be associated with the same AP. Unlike these technologies, Wi-Fi Direct [11], defined by the Wi-Fi Alliance [4], aims at enhancing D2D communications in Wi-Fi by building upon the successful IEEE 802.11 infrastructure mode and allows negotiation amongst devices to decide who will take over the AP-like functionalities. In this way, Wi-Fi Direct inherits all the improved features such as QoS, power saving, and security mechanisms developed for the Wi-Fi infrastructure mode in the past years.

In the following sections, we will study an overview of the Wi-Fi Direct specification [11]. We will focus on its novel functionalities and group formation procedures along with architecture, security and power of Wi-Fi Direct.

3.2.1 Architecture

In Wi-Fi Direct, the roles of an AP and a device (or client) are dynamic in nature, which means that at least one device in the network must behave as both a client and an AP (also referred to as Soft-AP). Before establishing a peer-to-peer (P2P) connection, each device will have to agree on the role that they will assume. Each network consists of P2P devices which communicate by establishing P2P Groups, similar to traditional 802.11 infrastructure networks.

One device acts like an AP in the P2P Group and is referred to as the P2P Group Owner (P2P GO), with the other devices acting as P2P Clients. When P2P devices discover one another, they negotiate their roles as P2P GO and P2P Client, and establish a P2P Group. Other P2P Clients can now join the P2P Group as they do in traditional Wi-Fi network. Legacy clients also communicate with the P2P GO as long as they are not 802.11b-only devices and support the required security mechanisms. These legacy devices do not formally belong to the P2P Group, but they simply ‘see’ the P2P GO as a traditional AP.

Looking at Figure 9 [11], we can see two simple Wi-Fi Direct architectures. Figure 9(a) represents a scenario with two P2P Groups. The first group is formed by a mobile phone which shares its 4G connection with two other laptops, where the phone acts as the P2P GO and the laptops act as the P2P Clients. Expanding the network further, one of the laptops create a separate P2P Group with a printer, for which the laptop if the P2P GO.

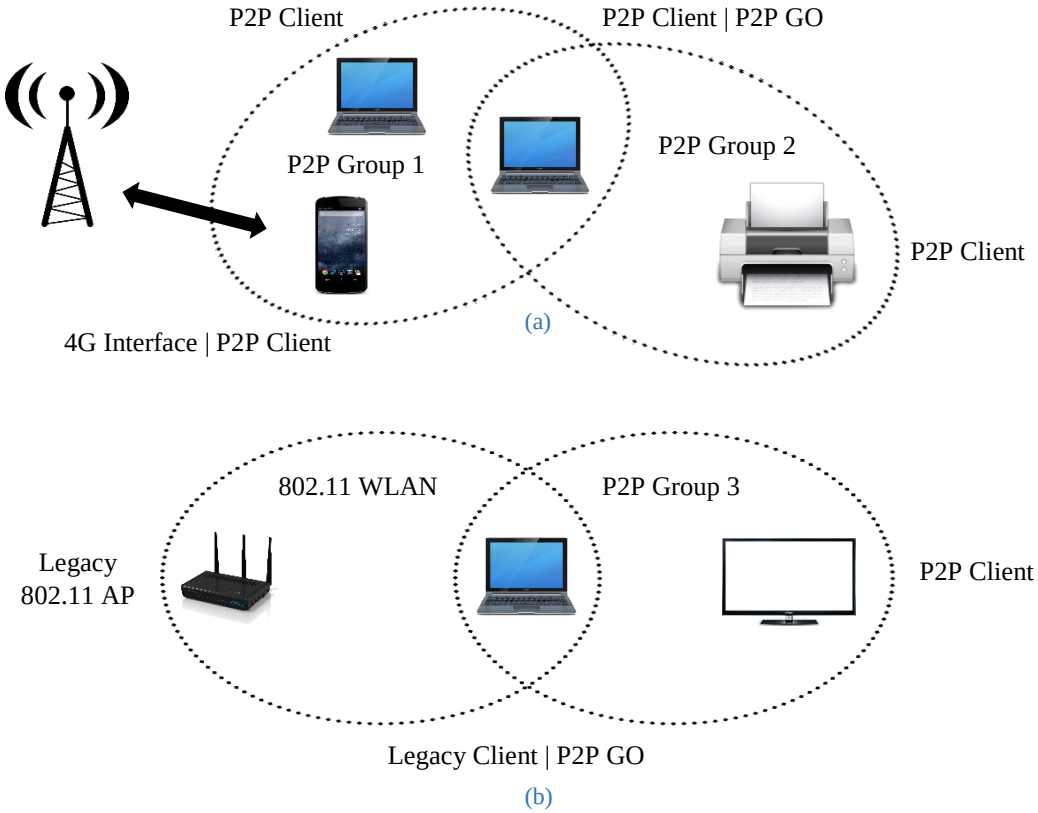


Figure 9: Wi-Fi Direct supported technologies and use cases [11]

For the laptop to work as both a P2P GO and P2P Client, it will alternate between the roles by using the Wi-Fi interface on a time-sharing basis. Figure 9(b) depicts a case of a laptop accessing the Internet via a legacy infrastructure AP, while simultaneously streaming data to a TV, by establishing a separate P2P Group, in which the laptop is the P2P GO. The P2P GO needs to announce itself through beacons, like a traditional AP. It runs Dynamic Host Configuration Protocol (DHCP) server to provide P2P Clients with IP addresses. Transfer of the role of a P2P GO is not permitted once a P2P Group has been

formed. If the P2P GO leaves the P2P Group, it will tear down the connections as well, and must be re-established.

3.2.2 Group Formation

Three main ways have been defined for two devices to establish a P2P Group – Standard, Autonomous and Persistent – depending on if they have to negotiate which device should take the role of the P2P GO, or if they have previously shared some security information.

Standard

This is the most basic case for formation of a P2P Group. A traditional Wi-Fi scan, either active or passive, is performed to discover any pre-existing P2P Groups and Wi-Fi networks. Following this, a *Discovery* algorithm [11] is executed as follows:

1. A P2P Device selects one of the channel: 1, 6 or 11 in the 2.4GHz band, as a *Listen* channel.
2. The device now alternates between two states: a *search* state, wherein it actively scans by sending Probe Requests in each of the channels; and a *listen* state, wherein it listens for the channel to respond with Probe Responses.

The amount of time spent in each state is decided randomly, but it usually falls between 100-300ms. Once both devices have found each other, the GO Negotiation phase begins. A three-way handshake occurs between the devices: *GO Negotiation Request, GO Negotiation Response* and *GO Negotiation Confirmation*. Here, the two devices decide who the P2P GO will be, and on which channel the group will operate (2.4 GHz or 5GHz band). A numerical parameter for each device, called the *GO Intent* value, is shared during the three-way handshake. The device that declares a higher value becomes the P2P GO. In rare cases when the *GO Intent* value is the same for both devices, a *tie-breaker* bit is included in the *GO Negotiation Request* phase, which is randomly set for each request. The next step is establishment of secure communication, using *Wi-Fi Protected Setup* or *WPS Provisioning* phase (discussed later), following which DHCP sets up the IP addresses for both devices.

Autonomous

Compared to the previous case, the *Discovery* algorithm is simplified the Autonomous case. Any P2P Device can autonomously create a P2P Group, where it immediately assigns itself to be the P2P GO by sitting on a channel and send beacon frames. Other P2P Devices discover and establish connections using the traditional scanning mechanism and proceed to *WPS Provisioning* followed by DHCP for IP address configuration. Clearly, no GO negotiation phase is required, nor is there any need to alternate between states.

Persistent

This case is commonly used when P2P Devices need to repeatedly connect to each other over a period of time. Here, the P2P Devices declare a group as *persistent*, by setting a flag in its beacon frames, Probe Responses and GO Negotiation frames. The devices store the group's networks credentials and assign the same P2P GO and Clients for any subsequent re-connections of the P2P Group.

After the *Discovery* phase, if a P2P Device recognizes to have previously formed a persistent group with the corresponding peer in the past, it uses the *Invitation Procedure* (a two-way handshake) to quickly re-instantiate the group. The *WPS Provisioning* phase is significantly reduced as the network credentials that were stored can be reused.

3.2.3 Security

Once the roles of each device in the P2P Group has been negotiated, the Wi-Fi Direct devices are required to implement Wi-Fi Protected Setup (WPS) [13] to support secure connections. WPS requires the P2P GO to implement an internal *Registrar*, and the P2P Client is required to implement an *Enrollee*. WPS comprises mainly of two phases. In the first phase, the *Registrar* is in charge of generating and issuing the network credentials (a security key) to the *Enrollee*. WPS is based on WPA-2 security and the Advanced Encryption Standard (AES)-CCMP is used as a cypher, randomly generating a Pre-Shared Key (PSK) for mutual authentication. In phase two, the *Enrollee* disassociates itself and reconnects using its new

authentication credentials. This way, for a persistent connection, only the second authentication phase is required as it would already have the required network credentials.

3.2.4 Power

Energy efficiency is of vital importance for battery constrained devices that may act as a P2P GO or a soft-AP. In current Wi-Fi networks, power saving mechanisms like legacy power save mode, have been defined only for clients, and P2P Clients in Wi-Fi Direct can also benefit from this. On the other hand, two new mechanisms [11] have been defined for the P2P GO in Wi-Fi Direct: Opportunistic Power Save protocol and the Notice of Absence (NoA) protocol.

Opportunistic Power Save

This mechanism assumes that P2P Clients use the legacy power saving protocol and use it as a leverage for P2P GO. A time window defined by *CTWindow* is advertised by the P2P GO within each Beacon and Probe Response frames. This window specifies the amount of time the P2P GO will stay awake after a Beacon is received. During this time, P2P Clients which are in their power saving mode can send their frames. All connected clients will be in doze state due to two reasons: either they announced a switch to the doze state by sending a frame with the Power Management (PM) bit set to 1, or they were already in the doze state during the previous beacon interval; and P2P GO can enter sleep mode until the next Beacon is to be sent. If a P2P Client leaves the power saving mode (by setting its PM bit to 0), the P2P GO is obligated to stay awake until all P2P Clients return to their power saving modes. The power saving mechanism for the P2P GO is clearly dependent on the activity of any associated P2P Clients. On the other hand, the Notice of Absence protocol allows the P2P GO to control its own energy consumption, as explained below.

Notice of Absence

Unlike the Opportunistic Power Save protocol, Notice of Absence (NoA) protocol allows a P2P GO to define and announce a time interval referred to as *absence periods*. P2P Clients are not allowed access to the channel irrespective of whether they are in power saving mode or not. The *absence periods* are also defined in Beacons and Probe Requests using four parameters:

1. Duration: Length of each *absence period.*
2. Interval: Time between consecutive *absence periods.*
3. Start Time: Start time of the first *absence period* after the current Beacon frame.
4. Count: Number of *absence periods* scheduled during the current NoA schedule.

If the P2P GO decides to change or cancel the current NoA schedule, it can do so by modifying or omitting the signaling element. P2P Clients will always obey the most recently received schedule. For both, Opportunistic Power Save and NoA protocols, Wi-Fi Direct specification does not define any mechanism to calculate the *CTWindow* or the *absence periods*.

3.3 Transmission Control Protocol

The Transmission Control Protocol (TCP) is a highly reliable, connection oriented transport layer protocol. The basics of TCP has been defined in RFC 793, and we will discuss some important underlying principles such as connection, reliability, flow control, congestion control in the following sections.

3.3.1 Segment Structure

TCP segments are transmitted as internet datagrams. While the Internet Protocol header carries several information fields, including the source and destination host address, the TCP header also allows information specific to TCP. The TCP header format [RFC 793] is shown in Figure 10 below.

1. Source Port and Destination Port (16 bits each): This allows the source process to pass application data to the correct source host and the destination host to pass application data to the correct process running on the destination end system.
2. Sequence Number (32 bits): This field contains a value that is used for numbering the packets of data flowing from the sender to the receiver. Any gaps in the sequence numbers of packets received allows the receiver to detect a lost packet, and duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.

3. Acknowledgment Number (32 bits): This field contains the value of the next sequence number that the receiver is expecting to receive. The ACK flag bit is also set to 1. TCP acknowledges bytes up to the first missing byte in the stream of packets, and thus TCP is said to provide cumulative acknowledgments.

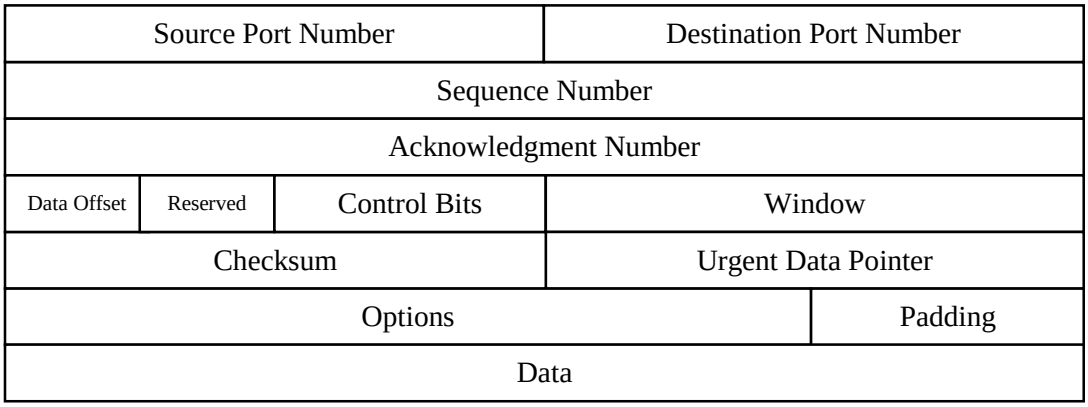


Figure 10: TCP Segment Structure [RFC 793]

4. Data Offset (4 bits): This field specifies the length of the TCP header in 32-bit words. The length varies due to the TCP options field. If the options field is empty, the header is 20 bytes.
5. Reserved (6 bits): Reserved for future use and value must always be zero.
6. Control Bits (6 bits): This field has 6 flags. The URG bit is used to indicate any 'urgent' upper-layer data in the given segment. ACK bit indicates that the value carried in the Acknowledgment Number is valid. PSH bit tells the receiver to pass the data to the upper layer immediately. RST, SYN and FIN bits are used for initiating and tearing down a TCP connection as discussed in the TCP's Connection Management in the following section.
7. Window (16 bits): This field is mainly used for flow control and specifies the amount of data the sender is willing to accept, after the most recently acknowledged segment.
8. Checksum (16 bits): This field is used to detect errors in a transmitted packet. The sender performs a 1's complement of the sum of all 16-bit words in the header and the result is put

in the checksum field of the TCP header. The receiver performs the same method on the received segment, and can detect any errors or alterations that have been made to the segment.

9. Urgent Data Pointer (16 bits): This field points to the sequence number of the octet following the urgent data, and is to be interpreted only when the URG flag is set.
10. Options (variable): This is mostly used when the sender wants to negotiate the maximum segment size (MSS) with the receiver. RFC 854 and RFC 1323 can be seen for further details.
11. Padding (variable): This field is used to ensure that the TCP header ends and data begins on a 32 bit boundary. It consists of zeros only.

Now that we have a fair idea to what goes in a TCP header, we will see how this header is used to make TCP a reliable, and connection oriented protocol.

3.3.2 Connection

The three-way handshake is used to establish a TCP connection between two devices, and hence is called a connection oriented protocol. TCP runs only on the end devices, and not in the intermediate network elements such as routers, switches etc. These elements are oblivious to TCP connections and simply forward any datagrams that they receive. TCP provides a full-duplex service, which means that application layer data can flow simultaneously from two devices, to each other. TCP also provides a point-to-point connection between a single sender and a single receiver, which implies that multicast (transfer of data from one sender to multiple receivers) is not possible with TCP.

Connection Management

A TCP connection is established by using the SYN and ACK flags, and it is torn down using the FIN and ACK flags of the TCP header. The client application process first informs the client TCP that it wants to establish a connection with the process at the server. Connection establishment is then proceeded through the following steps [15]:

1. A TCP SYN segment with no data, the SYN bit set to 1 and a randomly generated initial sequence number (SN_C) is encapsulated in an IP datagram and sent to the server.
2. Assuming that the IP datagram arrives at the server, it extracts the TCP SYN segment and allocates a TCP buffer and variables to the connection, and sends a connection-granted segment to the client. This segment does not have any data, the SYN bit is set to 1, the Acknowledgment Number field is set to (SN_C +1) and a randomly generated server sequence number (SN_S) is added to the Sequence Number field of the server TCP header. This implies that the server has granted and agreed to a connection with the client.
3. Once this segment is received at the client, buffers and variables are allocated to the connection. The client sends another segment with the SYN bit to 0, Acknowledgment Number field set to (SN_S+1) and Sequence Number field as (SN_C+1). This segment may or may not carry data in the segment payload.

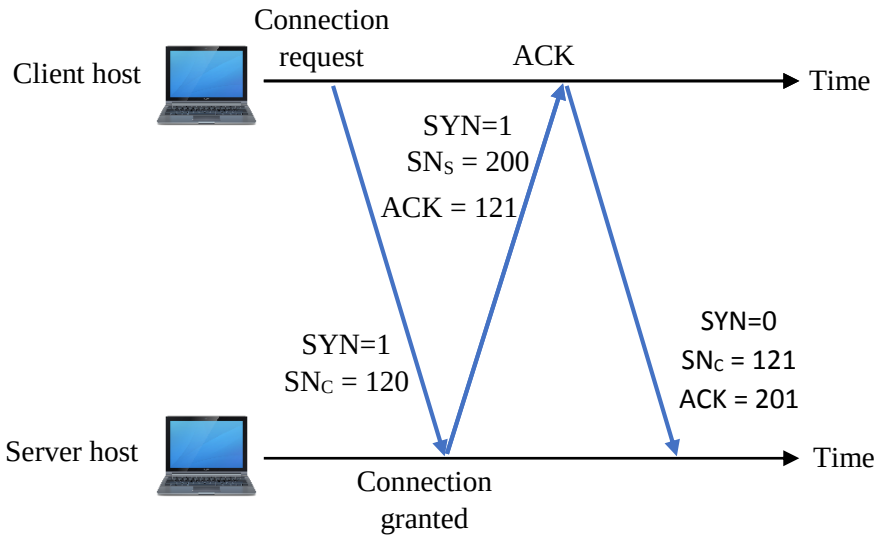


Figure 11: TCP three-way handshake

In all future segments, the SYN bit is set to 0. Since three packets are sent to establish a TCP connection, it is known as a three-way handshake. A basic example for establishing a TCP connection is shown in the Figure 11 above.

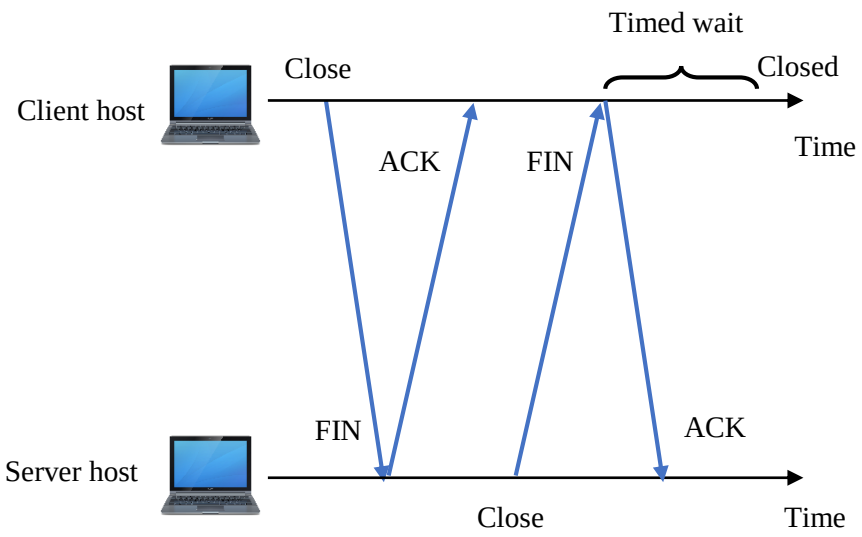


Figure 12: Terminating a TCP connection

A TCP connection is terminated in a similar manner [RFC 793] [15], once the client decides to close the connection. This is done to de-allocate the buffers and variables of the hosts. The TCP client send a segment to the TCP server process with the FIN bit set to 1. Once the server receives this segment, it sends the client an acknowledgment segment in return and sends another shut down segment with its FIN bit set to 1. To complete the termination process, the client acknowledges the server's final shut down segment and enters a wait period before it can shut down itself. All resources (buffers and variables) have been de-allocated. Figure 12 illustrates this process.

3.3.3 Reliable Data Transfer

It is essential for TCP to create a reliable data transfer service [15] on top of Internet's network layer service (IP service), as IP does not guarantee datagram delivery, in-order delivery or integrity of the data. Three major events are associated with data transmission and retransmission at the sender: data received from the application above, timeout and ACK reception.

When TCP receives data from the application, it encapsulates the data in a segment and passes the segment to IP. Each of these segments consist of a sequence number which is the byte-stream number of the first data byte in the segment. TCP starts the timer if no other timer is running, and its expiration time is calculated through an estimate of RTT (Round Trip Time). If timeout occurs, TCP retransmits the

segment that caused the timeout and restarts the timer as well. If an ACK arrives from the receiver, TCP compares the ACK value to the last unacknowledged byte or *SendBase* [15]. Thus, *SendBase-1* is sequence number of the last byte that is known to have been received correctly at the receiver. If the ACK value is greater than *SendBase*, the ACK is acknowledging one or more previously unacknowledged segments. The *SendBase* variable is then updated and the timer is restarted if there are currently any unacknowledged segments.

TCP implementations employs some modification of the events described above. Whenever a timeout occurs and a TCP sender has to retransmit an unacknowledged segment, it sets the next timeout interval to twice the previous value, rather than estimating it from the RTT. Thus, the intervals grow exponentially after each retransmission. However, when data is received from the application above, or when an ACK in received, the timeout interval is derived from RTT. This provides a limited amount of congestion control in TCP, which is discussed in detail in the later sections.

One of the major problems with the retransmission is that the timeout periods can be extremely long. When a segment is lost, a large timeout period forces a delay in retransmitting the lost segment, increasing end-to-end delay. Instead, TCP identifies a lost segment through reception of duplicate ACKs at the sender. This means that the receiver re-acknowledges a segment for which the sender has already received an earlier ACK. When a segment is lost, the receiver will send back-to-back acknowledgments for that segment, and once the sender receives 3 duplicate ACKs, it will assume that the segment is lost and will retransmit. This helps in reducing the end-to-end delay in the network. This mechanism is also known as Fast Retransmit.

3.3.4 Flow Control

As discussed in connection management of TCP, buffers are allocated for every connection at the client and server processes. The receiver TCP buffer receives bytes in the correct sequence and the associated application reads data from this buffer. It is possible that the sender overflows the receiver's TCP buffer, and hence TCP provides a flow-control service. It can be considered as a speed-matching

service – synchronizing the rate at which the sender is sending data against the rate at which the receiver application reads the data.

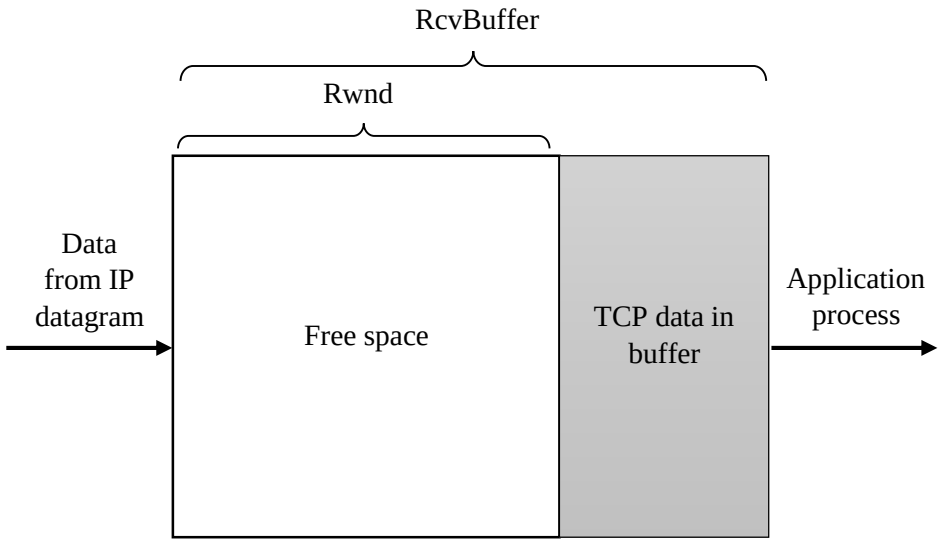


Figure 13: Buffer Allocation in TCP [15]

As explained in [15], in flow control, the sender maintains a variable called the *Receive Window*, which gives an idea of how much free buffer space is available at the receiver. Consider a scenario where Host A is sending a file to Host B over a TCP connection. Host B will allocate a receive buffer to the connection denoted by *RcvBuffer*. Host B will read data from this buffer from time to time.

LastByteRead gives the last byte in the data stream read from the buffer by Host B. *LastByteRcvd* gives the last byte in the data stream that has arrived from the network and has been placed in the receiver buffer at B. In order to avoid overflow at the allocated buffer, the following condition must be satisfied;

$$LastByteRead - LastByteRead \leq RcvBuffer$$

The receive window, given by *rwnd* is set to the amount of free space in the buffer [RFC 813];

$$rwnd = RcvBuffer - [LastByteRecvd - LastByteRead]$$

The value of *rwnd* is dynamic as the amount of free space changes with time. An illustration of flow control is shown in Figure 13 [15]. Host B puts the value of *rwnd* in the Window field of its TCP segment and sends it to Host A. In turn, Host A keeps a track of the *LastByteSent* and *LastByteAcked*, the meaning of which is evident. The difference between these two variables is the amount of unacknowledged

data that A has sent to B. This unacknowledged data should not exceed the value of *rwnd* received from Host B, thus assuring that A does not overflow Host B's buffer.

$$LastByteSent - LastByteAcked \leq Rwnd$$

Now consider the scenario where Host B's receiver buffer becomes full such that *rwnd* = 0. After telling Host A that *rwnd* is zero, suppose that Host B has nothing to send to A. This means that once Host B's buffer starts to empty, it won't be able to inform A by sending a new segment and Host A will be blocked. To avoid this problem, TCP specification requires Host A to continue to send segments with one data byte when B's receiver window is zero. These segments will have to be acknowledged by Host B, and hence will be able to inform A about any change in the free space, given by *rwnd*.

3.3.5 Congestion Control

It is essential that TCP provides an end-to-end congestion control mechanism since the IP layer provides no feedback to the end systems regarding network congestion. This can be done by using an additional variable, the *Congestion Window* It is denoted by *cwnd* and constraints the rate at which a TCP sender can transmit traffic in to the network. Thus, considering Flow Control and Congestion Control in TCP, we have the following condition [15], [RFC 5681];

$$LastByteSent - LastByteAcked \leq \min \{cwnd, rwnd\}$$

For this section, let us assume that the *rwnd* is always large, and the amount of unacknowledged data at the sender is primarily limited by *cwnd*. Also assume that the sender always has some data to send. This way, the sender's rate is roughly *cwnd/RTT* bytes/sec; where RTT is the average round trip time for a packet to be sent and acknowledged in a network. Thus, the rate at the sender can be adjusted by changing the value of *rwnd*. Congestion control in TCP follows three basic principles:

1. A lost segment implies congestion and thus TCP sender's rate should be decreased. This lost segment is identified either by a timeout or duplicate ACKs.
2. When an ACK arrives for a previously unacknowledged segment, it implies that the network is delivering the data to the receiver and thus the sender rate can be increased.

3. TCP's strategy for adjusting its transmission rate is very simple. It keeps probing the network and increases its rate until a loss occurs (indicating congestion in the network), where it backs off from that rate and begins probing again to see when congestion occurs.

The above principles are seen implemented in three ways in TCP's Congestion Control algorithm: Slow Start, Congestion Avoidance and Fast Recovery [15], [RFC 5681]. While Slow Start and Congestion Avoidance is mandatory, Fast Recovery is recommended, but not required for TCP senders. Figure 14 gives an example of how TCP enters all three states.

Slow Start

When a TCP connection starts, the value of Congestion Window or *cwnd* is initialized to a small value of 1 maximum segment size (MSS) resulting in an initial sending rate of roughly MSS/RTT. MSS is defined as the maximum amount of data that can be placed in a segment (along with the TCP/IP header), and is determined by the maximum transmission unit (MTU) or the largest link-layer frame that can be sent by the sending host. The available bandwidth may be much larger than MSS/RTT, thus the TCP sender would want to calculate the bandwidth as fast as possible.

The Slow Start state begins with the value of *cwnd* beginning at 1 MSS and increasing by 1 MSS every time a transmitted segment is first acknowledged. So once the sender receives the first acknowledgement, it will increase its *cwnd* by 1 and transmit two segments. When these two segments are acknowledged, it will increase its window by 1 for each segment, thus incrementing *cwnd* to 4 MSS. Hence, the sender rate starts slowly, but grows exponentially during the slow start state. This exponential growth ends when congestion (a loss event) occurs, which is indicated by a timeout. The sender resets the value of *cwnd* to 1 and begins the slow start process afresh. Another variable is added, *ssthresh* which is *cwnd/2* – half the value of the congestion window when congestion was detected.

Once the *cwnd* reaches or surpasses *ssthresh*, slow start state ends and TCP transitions into congestion avoidance mode, where the value of *cwnd* increases more cautiously. Congestion can also be

detected by three duplicate ACKs, in which TCP performs fast retransmit and enters the fast recovery state as discussed below.

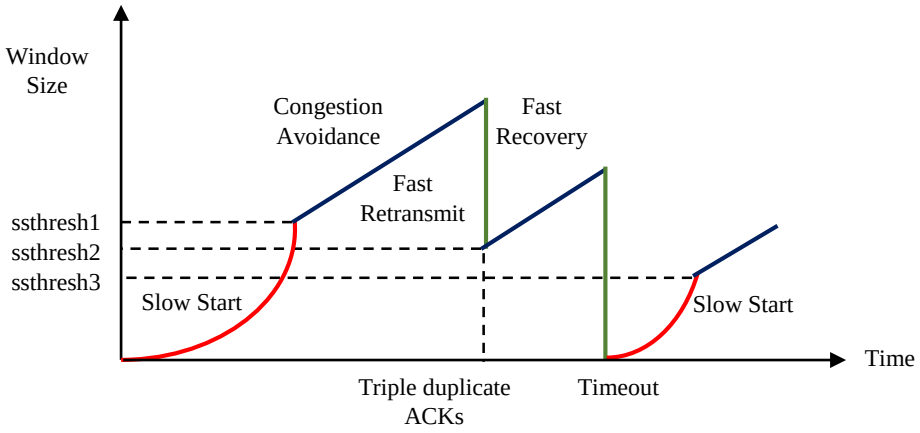


Figure 14: TCP Congestion Control Mechanism

Congestion Avoidance

When TCP enters the congestion avoidance state, *cwnd* is approximately half the value when congestion was encountered. The value of *cwnd* must therefore be increased in a conservative manner, by incrementing *cwnd* by a single MSS every RTT. This means that if there are 10 segments being acknowledged every RTT, each acknowledgment would imply an increase in *cwnd* by 1/10. If and when timeout occurs, the value of *cwnd* is reset to 1MSS and *ssthresh* is updated to half the value of *cwnd* when congestion occurred. If the loss event is triggered by triple duplicate ACKs, the network continues to deliver segments to the receiver and a less drastic approach is adopted: TCP halves the value of *cwnd* and *ssthresh* is half the value of *cwnd* when the triple duplicate ACKs were receiver. Fast Recovery state is then entered.

Fast Recovery

For every duplicate ACK that is received for a missing segment, *cwnd* is increased by 1 MSS. Eventually, when an ACK does arrive, acknowledging the missing segment, TCP enters congestion avoidance state after decreasing *cwnd*. If a timeout event occurs, TCP enters slow-start state after decreasing

the *cwnd* to 1 MSS and *ssthresh* to half the value of *cwnd* when timeout occurred. TCP Tahoe (RFC 5681), which is the earlier version of TCP does not incorporate Fast Recovery. Whereas, the newer version TCP Reno does.

3.4 Creating Network Applications

Now that we have a basic understanding of how network applications works, we can explore more on how the programs are actually created. Each network application consists of two programs in general: a client and a server program. Both the programs reside in two different devices. When they are executed, their respective processes are created which in turn communicate by writing to and reading from sockets. Since we have previously discussed how TCP works, we will see how the client and server programs are created for TCP itself.

3.4.1 Socket Programming with TCP

As we have seen in the previous section, TCP in a connection oriented protocol. This implies that the client and server need to perform a three-way handshake before any data can be sent across the connection. Two sockets are associated with the TCP connection – a client and server socket – each linked with an IP address and a port number. The data can be dropped via the socket once a TCP connection has been established [15].

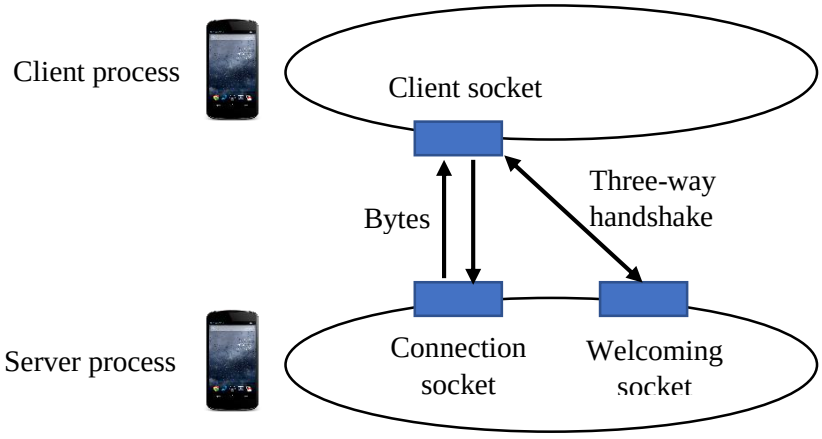


Figure 15: TCP sockets

It is the client's job to contact the server, which implies two things. First, the server has to be up and running for an incoming TCP connection. Secondly, a special socket must be defined at the server that welcomes initial contact from the client process. This socket is referred to as the *Welcoming socket* [15]. Once the server process is running, the client can initiate a TCP connection by creating a TCP socket. The client will need to specify the address of the *Welcoming socket* at the server, which is the IP address of the server host along with the port number of the socket. A three-way handshake can now be initiated at the transport layer, which is completely invisible to the client and server processes. During the three-way handshake, the client process communicates with the server through the *Welcoming socket*, which initiates creation of a new socket called the *Connection socket* is dedicated to that particular client (as shown in the Figure 15).

Eventually, the *Connection socket* and the *Client socket* is directly connected to send bytes of data. TCP guarantees that the server process will receive each and every byte, in the order that it was sent. The same socket is also used at the client process to receive bytes from the server; and at the server process to send bytes into its *Connection socket*.

4 SYSTEM MODEL AND FORMULATION

Now that we have a good understanding of how wireless networks work, we will focus on the Energy-Aware Computation (EaC) system model for this thesis. Our model shown in Figure 16(a) helps in focusing on the bottlenecks of the system: processing power, energy consumption of the mobile devices and bandwidth. The model consists of *N* mobile devices, where the mobile device communicates with each other via a D2D connections and the source communicates with mobile devices via cellular or Wi-Fi links. Also, our analysis assumes a time slotted system where *t* refers to the beginning of slot *t*.

4.1 System Model

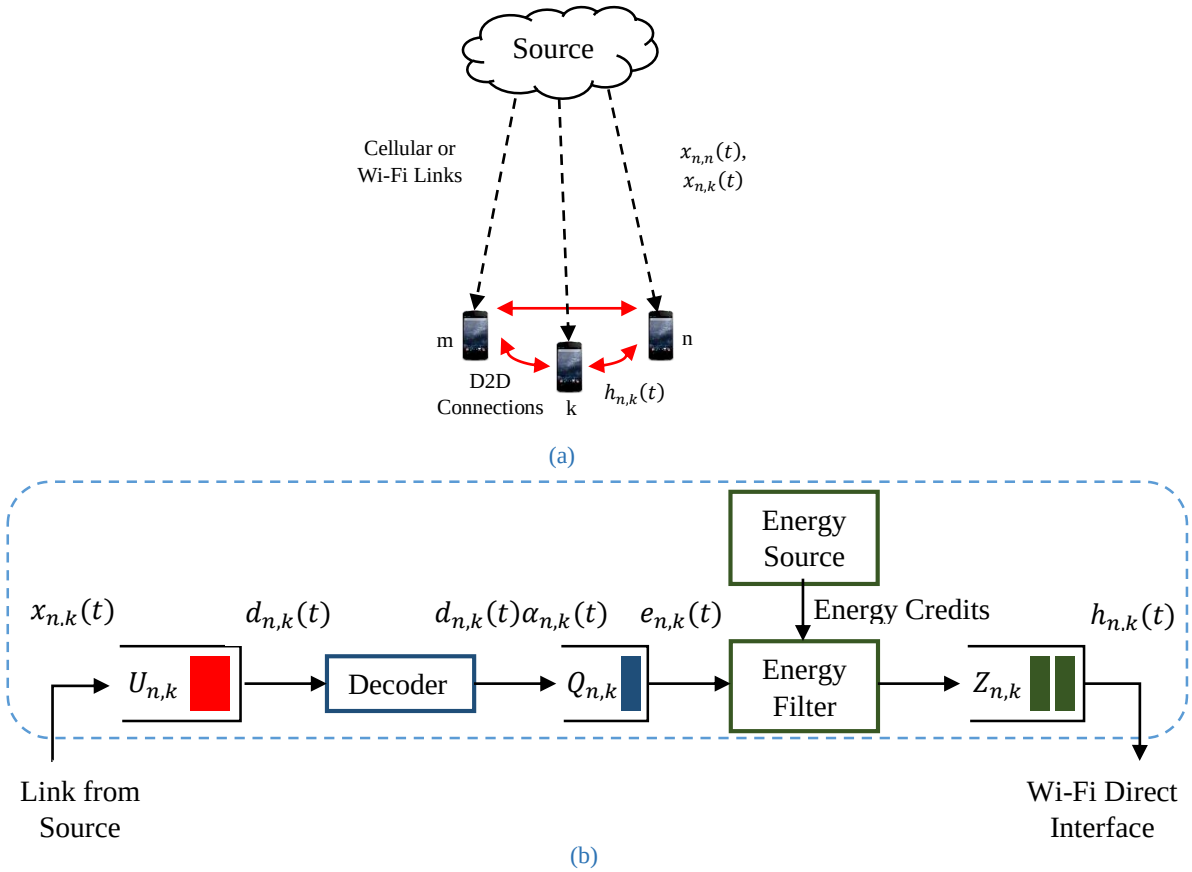


Figure 16: Building Block of mobile device n

For *N* devices, the flow rate towards any device *n* in Figure 16(a) is $\sum_{k\in N}\{x_{n,k}(t)\}$, where $x_{n,k}(t)$ is the transmission rate of the packets from the source to the device *n* and these packets will be used by device *n*. For $x_{n,k}(t)$, packets will be processed by device *n*, to be further forwarded to device *k*. Multiple

queues have been implemented in a mobile device, to represent different levels of progress in packet processing and energy awareness for a mobile device. Figure 16(b) shows these building blocks of a mobile device *n*, where $U_{n,k}$, $Q_{n,k}$ and $Z_{n,k}$ represent three levels of queues constructed at the mobile device *n* to process packets for mobile device *k*. All the incoming packets from the source are stored in $U_{n,k}$, and is then forwarded to the *decoder* at rate $d_{n,k}(t)$. These packets are decoded by the decoder block and passed to queue $Q_{n,k}$ at rate $d_{n,k}(t).\alpha_{n,k}(t)$, where $\alpha_{n,k}(t)$ is a positive real value which captures any changes in the rate at the decoder. It is thus also identified as a rate shaper. A common example is to consider the decoder to be a H.264/AVC decoder such that the output of the decoder is higher than the input, as H.264/AVC will decompress the incoming packets. This change in rate will be captured by $\alpha_{n,k}(t)$.

The decoded and rate changed packets are then passed to queue $Q_{n,k}(t)$, which further passes it to the *energy filter*. This filter is associated with the energy source and determines the amount of energy that can be consumed on any given task, at each time slot. *Energy credits* are allotted to the device depending upon the battery level as well as an estimate of the expected battery consumption in the near future. These credits are used to calculate the number of packets that can be supported by the device, and that value enters the energy filter in terms of energy credits. Therefore, at each time slot, packets will be transmitted from $Q_{n,k}(t)$, to $Z_{n,k}(t)$ if there are energy credits in the filter. Finally, the packets are transmitted to the original destination *k* via a local interface.

Since the focus of our thesis is the processing power and energy consumption of a mobile device, we will see concentrate on how the above model helps in both scenarios. Consider a case where the number of packets in $U_{n,k}$ increases too much. This implies that the decoder, hence the processing power is the bottleneck, and node *n* should stop receiving packets from the source. Likewise, increase in $Q_{n,k}$ implies that the energy filter in the bottleneck and hence node *n* should stop receiving packets. Since the packets are intended for destination *k*, there is also a possibility of buildup in $Z_{n,k}$ if the link between node *n* and *k*

is the bottleneck of the system. Based on these observations, a resource allocation algorithm was developed in [14], from which we will formulate a solution in the next section.

4.2 Problem Formulation and Solution

According to the stability region and NUM formulation in [14], in order to maximize the efficiency of our system, we need to take into consideration the number of packets in each of the queues $U_{n,k}$, $Q_{n,k}$ and $Z_{n,k}$. The main focus of this thesis will be on the system model of an individual mobile device shown in Figure 16(b), which is a part of the system represented in Figure 16(a). The D2D connections between various devices from Figure 16(a) and its efficiency maximization has been discussed in [14] as well.

The problem of processing power and energy consumption in a mobile device as discussed above, have the following two proposed solutions:

1. Decoder control: At every time slot *t*, the rate $d_{n,k}(t)$, who's value is greater than the decoding rate of the device, is determined by calculating the difference in the number of packets processed by $U_{n,k}$ and $Q_{n,k}$ and by setting a maximum threshold (Thr_U) for the number of buffered packets in $U_{n,k}$. Thus, we have the conditions: $U_{n,k}(t) - Q_{n,k}(t) \geq 0$ and $Buffered(U_{n,k}(t)) < Thr_Q$. If these conditions are not satisfied, no packets are forwarded to the decoder, and further to queue $Q_{n,k}$.
2. Energy Control: At every time slot *t*, the rate $e_{n,k}(t)$ is determined by calculating the difference in the number of packets processed by $Q_{n,k}$ and $Z_{n,k}$ and by setting a maximum threshold Thr_U for the number of buffered packets in $Q_{n,k}$. Thus, we have the conditions: $Q_{n,k}(t) - Z_{n,k}(t) \geq 0$ and $Buffered(Q_{n,k}(t)) < Thr_U$. If these conditions are satisfied, the packets are forwarded to the energy filter. Here, the energy credits are evaluated, and if the device has enough credits the packets are forwarded to $Z_{n,k}$. If the condition is not satisfied, packets aren't forwarded to the filter, and further to $Z_{n,k}$.

Note that all transmissions over the links are unicast transmissions in our work, where unicast is dominantly used in practice over cellular, Wi-Fi, and Wi-Fi Direct links. It is straightforward to extend our framework for broadcast transmissions. As it can be seen, our algorithm takes into account the resources of processing power and energy in addition to bandwidth in an optimal manner. In the next section, we will see the implementation details and the performance of our algorithm in a practical setup.

5 IMPLEMENTATION DETAILS

The scenario shown in Figure 16(b) was implemented on mobile devices, specifically Nexus 5 smartphones and Nexus 7 tablets. The source in Figure 16(a) is also represented using a Nexus 7 tablet. In a simple source-to-receiver transmission, the building blocks shown in Figure 16(b) are implemented on top of the transport layer at the receiver. The TCP socket is connected to the output of queue $Z_{n,k}$, and over this TCP connection Wi-Fi Direct operates.

We created the Energy-Aware Computation (EaC) application using Eclipse, along with the Android SDK. The IP Addresses associated with the WiFi-Direct Group was acquired after initiating WiFi-Direct connections using the Android 5.1.1 operating system. These IP Addresses were used to create and maintain the sockets on both the source and receiver phones, as discussed in Section 3.4. Once the sockets were created, packets could be transmitted from the source to the receiver.

Source Device

Figure 17: Source Device

At the source, each packet is of size 500 bytes. These packets are read using the public java class *BufferedInputStream* from the file and stored in a buffer, and then pushed to the *OutputStream* of the source socket. This is done continuously till the end of the file has been reached. At this point, the *BufferedInputStream* and source socket is closed.

Receiver Device

At the receiver, the socket uses its *BufferedInputStream* to read the incoming data, one packet (500 bytes) at a time. This data will later be stored using an *OutputStream* in a location which has been pre-defined by the user, if it satisfies the Decoder and Energy conditions as shown in Figure 16(b). All queues are created using the *LinkedList* class which implements the *Queue* interface in Java. Each of these three

queues is implemented on separate threads, so that they can be executed simultaneously and independently. All queues work on a slotted system where each slot is of 20ms, and no more than 100 packets can be sent in a slot.

In the first thread, in every slot, packet that are received at the socket are added to the first queue $U_{n,k}(t)$. In the second thread, the decoder decodes the data (this can vary in terms of complexity) and forwards it to the queue $Q_{n,k}(t)$ according to the following rules:

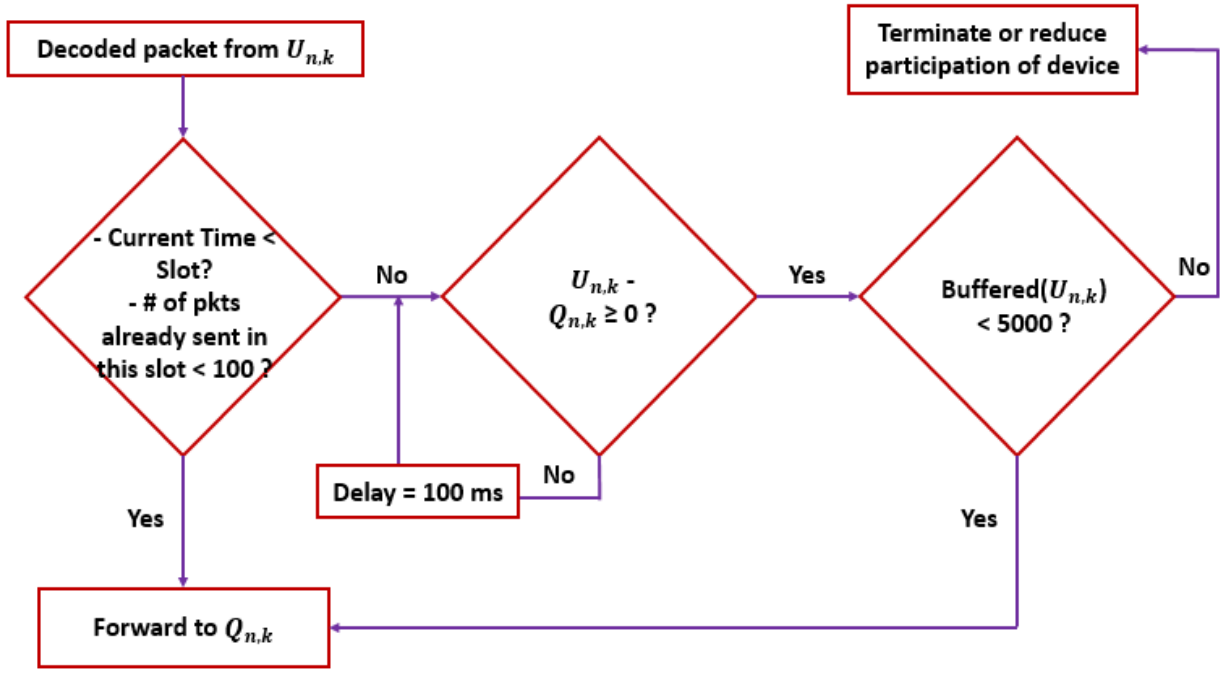


Figure 18: Receiver - Flowchart for thread 2

1. In the duration of a slot, every packet that is received is transmitted directly to the next queue $Q_{n,k}(t)$.
2. If the packet is received at a time that exceeds the current time slot of 20ms, it is transmitted to the next queue $Q_{n,k}(t)$ if the following conditions are satisfied:
    a. The difference between the number of packets processed by queue $U_{n,k}(t)$ and $Q_{n,k}(t)$, i.e., $U_{n,k}(t) - Q_{n,k}(t) \geq 0$

b. The number of buffered packets in queue $U_{n,k}(t)$, i.e. $Buffered\left(U_{n,k}(t)\right) < 5000$ which implies that there should not be more than 5,000 packets waiting to be processed. If this condition is not satisfied, it implies that the decoder is a bottleneck. The participation of this device is either completely terminated, or reduced in comparison to others.

3. If the number of packets in the slot have already exceeded 100, the slot is terminated at that instance and the end-of-slot condition (Step 2) is executed.

In the third thread, the Energy Filter is used to first determine whether the device has enough energy resources to process the incoming file. The Energy Filter forwards the packets to the final queue $Z_{n,k}(t)$ based on the following rules:

1. A threshold is first set by the user, which indicates the minimum battery percentage that the user requires after the file has been processed.
2. If the device is plugged in/charging while processing the packet, it simply forwards the packets to the next queue ($Z_{n,k}(t)$), in each slot.

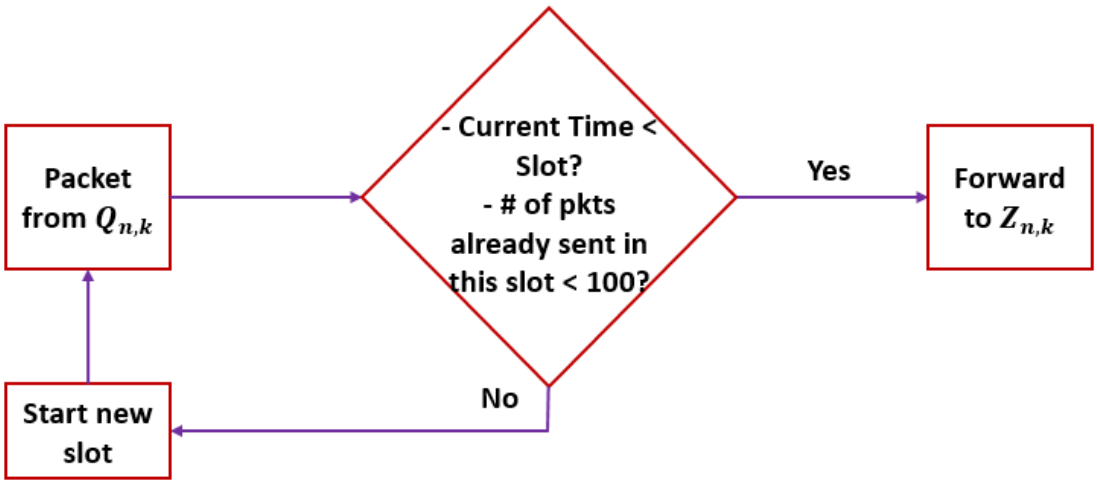


Figure 19: Receiver - Flowchart for thread 3, device is charging

3. If the device is not plugged in and the packet is received in a slot duration, it is directly forwarded to the next queue ($Z_{n,k}(t)$).

4. If the device is not plugged in and the packet is received at a time that exceeds the current time slot of 20ms, it is transmitted to the next queue $Z_{n,k}(t)$ if the following conditions are satisfied:
    a. The difference between the number of packets processed by queue $Q_{n,k}(t)$ and $Z_{n,k}(t)$, i.e., $Q_{n,k}(t) - Z_{n,k}(t) \geq 0$
    b. The number of buffered packets in queue $Q_{n,k}(t)$, i.e. $Buffered\left(Q_{n,k}(t)\right) < 5000$ which implies that there should not be more than 5,000 packets waiting to be processed.
    c. Energy Credits: The battery level of the device must be greater than the threshold set by the user.
5. If the number of packets in the slot have already exceeded 100, the slot is terminated at that instance and Step 4 is executed.

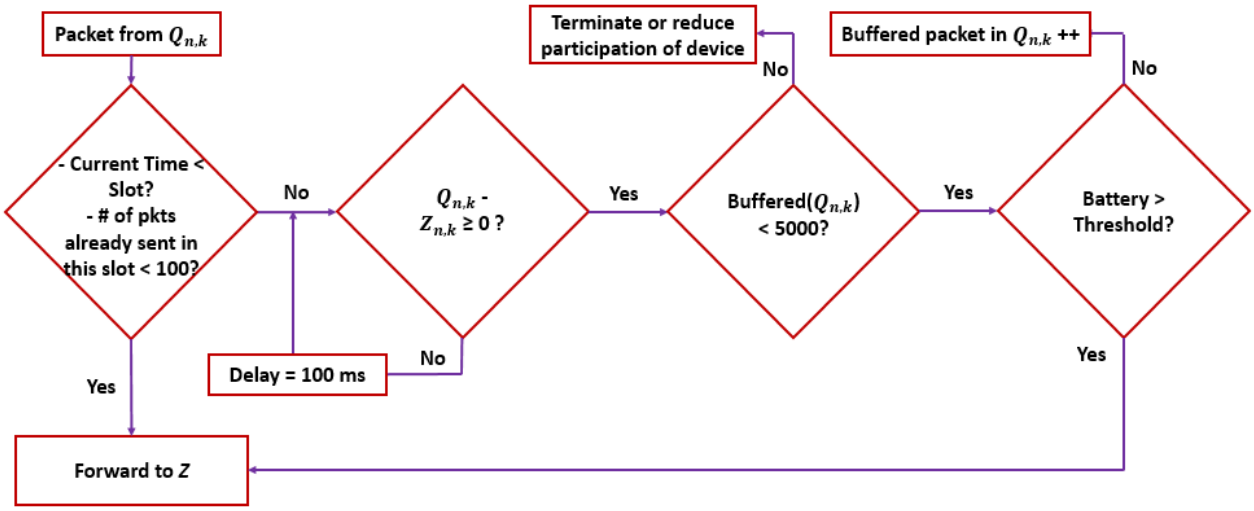


Figure 20: Receiver - Flowchart for thread 3, device is not charging

Once the packet has been forwarded to queue Z according to the above rules, it is written to a file using the *OutputStream.* Once all packets have been received, the *OutputStream* and socket at the receiver is closed. The file transfer is complete.

In our experiments, we used two files of size 22.49MB and 308.80MB. Both the files are video files in MP4 format. Since each packet is 500 bytes each, there are roughly 45,000 packets for the smaller file and 645,000 packets for the larger file. Each of the queues mentioned above have a maximum size of 150,000 packets. With these implementation details, we present our results in the next section.

6 PERFORMANCE EVALUATION

This section discusses the results in different levels of computational complexity. The results show a significant improvement in throughput in terms of data transfer rate and battery consumption as well. Our results will focus on the two main goals of this thesis: processing power and energy consumption.

6.1 Processing Power

We will first show the results we achieved while analyzing the processing power in data intensive applications. Like we had discussed in the beginning of this thesis, processing power limits the rate of a device by introducing computational complexities up to $O(n^3)$. These results are shown for transfer of the smaller file, size 22.49MB.

Two-way Transmission

Our first setup is focused on exchange of data over two mobile devices (Phone A and Phone B), under the 3 sets of computational complexities discussed above. This analysis helps us focus on the data rates when a cooperative setup is introduced, and mobile devices help each other by exchanging data. As expected, the data rates reduce during the full-duplex transmission, and are further reduced when the complexities are introduced. Figures 21, 22 and 23 show the data rates for two-way transmissions for complexities $O(1)$, $O(n)$ and $O(n^2)$ respectively.

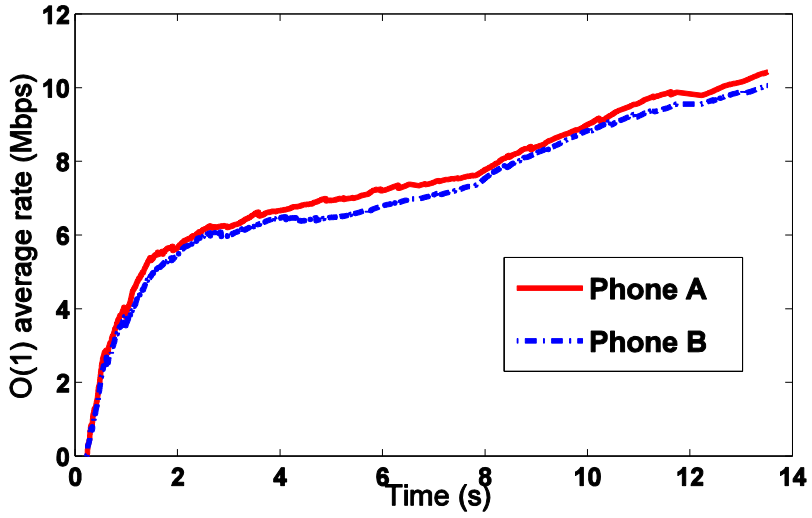


Figure 21: Two-way Transmission: Data rate for O(1) complexity

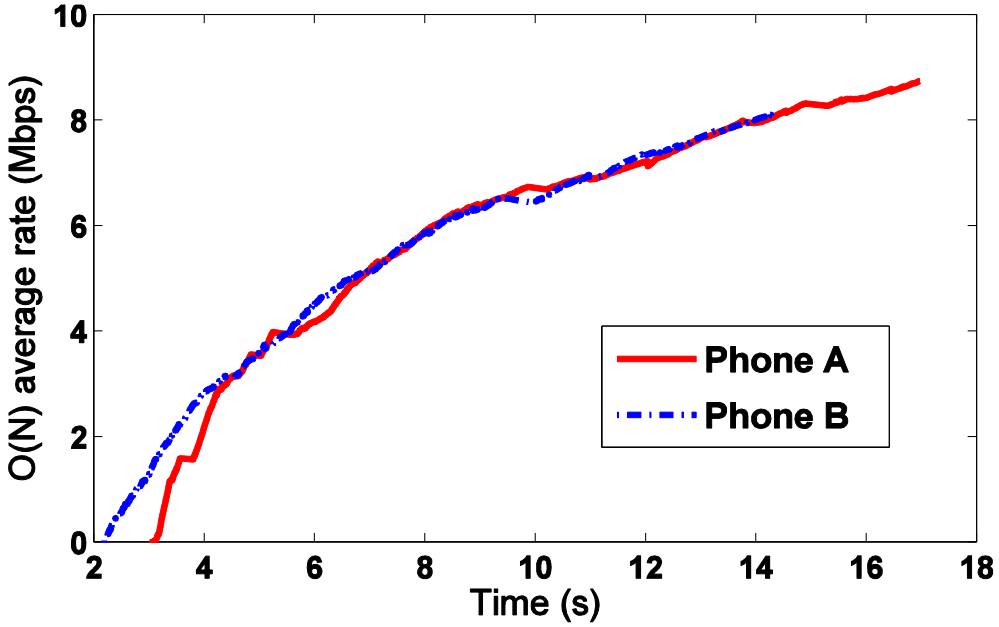


Figure 22: Two-way Transmission: Data rate for O(n) complexity

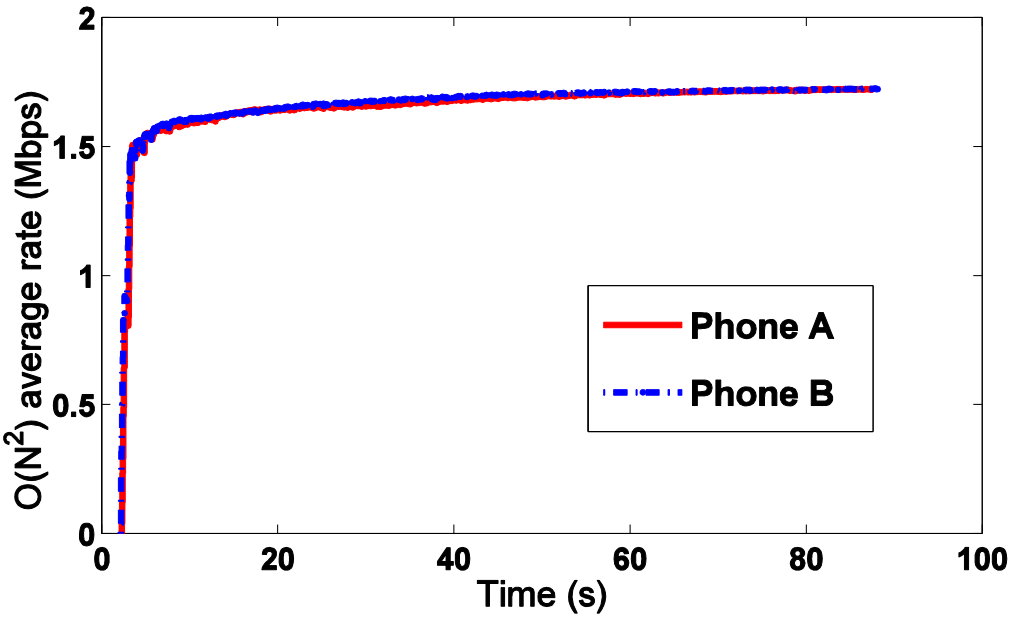


Figure 23: Two-way Transmission: Data rate for $O(n^2)$ complexity

6.2 Energy Consumption

Our second set of results concentrate on the energy consumption of mobile devices. This setup involves two scenarios; the first one is the 'No Energy Constraint' scenario in which the battery threshold set by the user is below the current battery level, and the second one is the 'Energy Constraint' scenario in which the battery threshold set by the user is above the current battery level.

In the first scenario, packets will check the end-of-slot condition every 20ms and will forward the packet, whereas in the second scenario the packets will be processed at a lower rate (1Mbps), as there is not sufficient battery in the device. This setup is shown for complexities *O(1)*, *O(n)* and $O(n^2)$; and as expected, the battery consumption is lower under the influence of the Energy Constraint applied to the battery level using the threshold. These results are shown in Figures 24, 25 and 26; and are shown for transfer of the larger file, size 308.80MB with the battery threshold set at 40%.

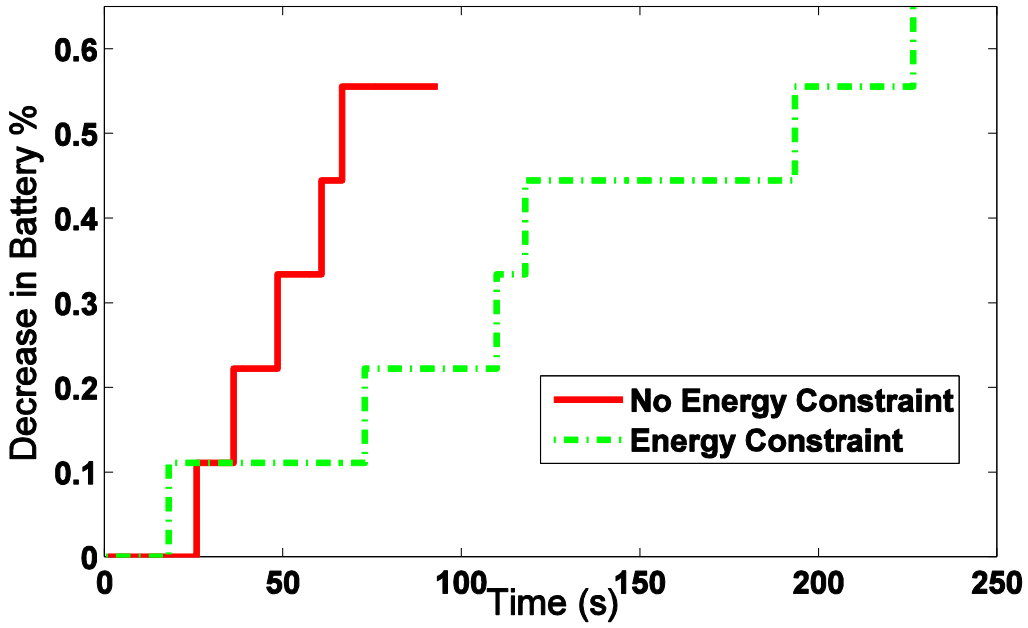


Figure 24: Decrease in Battery % over Time for O(1) complexity

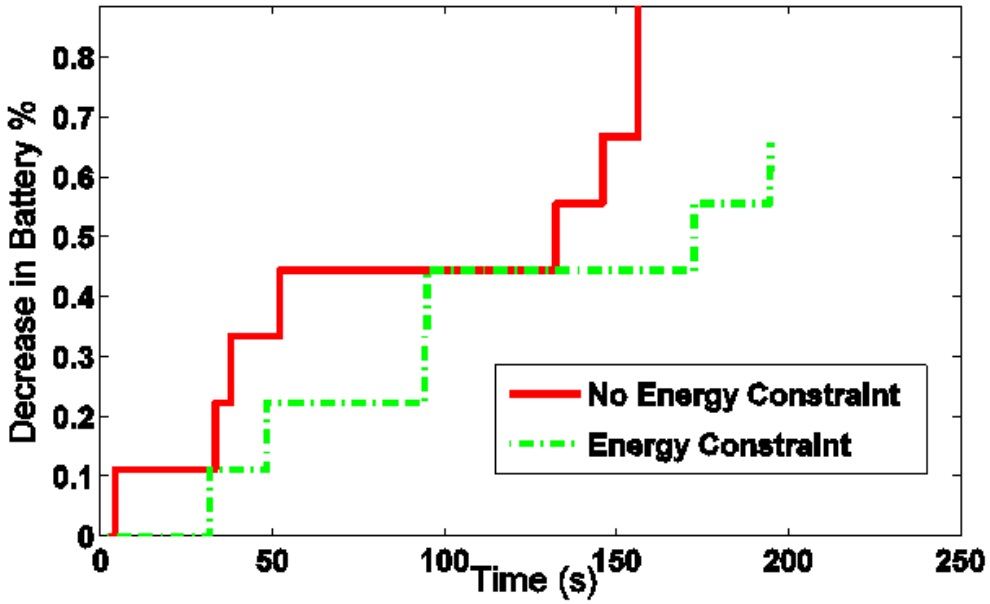


Figure 25: Decrease in Battery % over Time for O(n) complexity

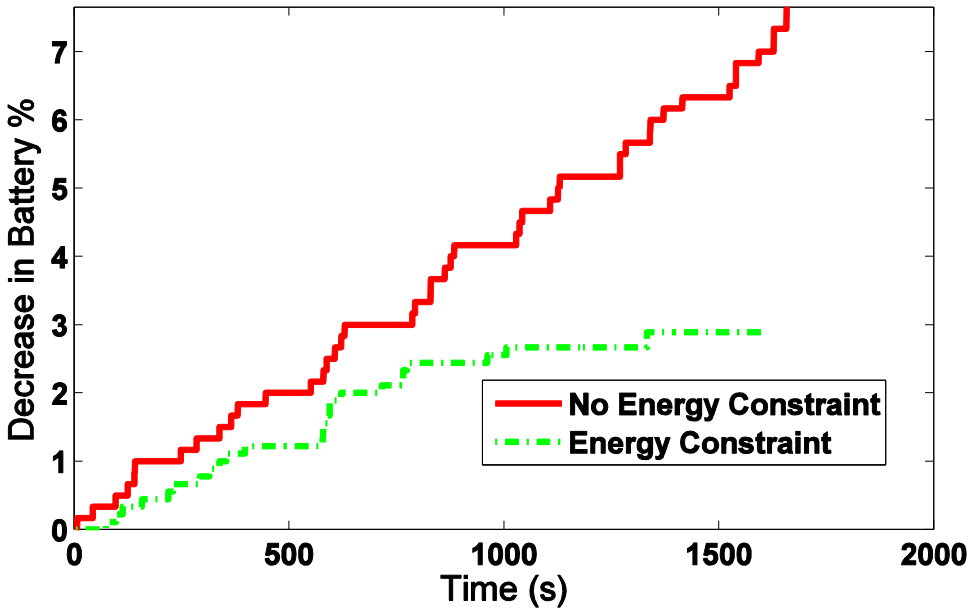


Figure 26: Decrease in Battery % over Time for $O(n^2)$ complexity

7 CONCLUSION

In this thesis, we considered a device-to-device, Wi-Fi Direct connection between two mobile devices within proximity of each other. We successfully showed how data rates are affected due to changes in decoder complexities, for both one way and two way transmissions. We also showed that the battery consumption is significantly less in our algorithm with the Energy Filter, which is also implemented in the cooperative setup shown in [14]. Our framework provided a set of algorithms for decoder and energy control. These algorithms were implemented in a test bed which consists of real mobile devices and showed significant performance improvements.

REFERENCES

[1] "Mobile Statistics Report, 2014-2018", The Radicati Group, Inc. February, 2014.

[2] "Cisco visual networking index: Global mobile data traffic forecast update", 2014-2019.

[3] "Ericsson Mobility report", February 2015.

[4] Wi-Fi Alliance, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.0 [Online]. Available: http://www.wi-fi.org

[5] P. Vingelmann, P. Zanaty, F. Fitzek, and H. Charaf, "Implementation of random linear network coding on opengl-enabled graphics cards," in Wireless Conference, 2009. EW 2009. European, May 2009, pp. 118–123.

[6] H. Shojania, B. Li, and X. Wang, "Nuclei: Gpu-accelerated many-core network coding," in INFOCOM 2009, IEEE, April 2009, pp. 459–467.

[7] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with h.264/avc: tools, performance, and complexity," Circuits and Systems Magazine, IEEE, vol. 4, no. 1, pp. 7–28, First 2004.

[8] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 13, no. 7, pp. 704–716, July 2003.

[9] IEEE standard 802.11, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", March 2012.

[10] G. Anastasi, E. Borgia, M. Conti, E. Gregori, IEEE 802.11 ad hoc networks: performance measurements, in: Proceedings of the Workshop on Mobile and Wireless Networks (MWN 2003) in conjunction with ICDCS 2003, May 19, 2003.

[11] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device to device communications with WiFi Direct: Overview and experimentation," IEEE Wireless Communication, vol. 20, no. 3, pp. 96-104, June 2013.

[12] IEEE 802.11z-2010 — Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 7: Extensions to Direct-Link Setup (DLS).

[13] Wi-Fi Alliance, Wi-Fi Protected Setup Specification v1.0h, Dec. 2006.

REFERENCES (Continued)

[14] A. Singh, Y. Xing, H. Seferoglu, "Energy-Aware Cooperative Computation in Mobile Devices," under preparation for submission, Nov. 2015.

[15] Kurose, J.F. and Ross K.W.: Computer Networking: A top-down approach, 2013.

[16] M. Ramadan, L. E. Zein, and Z. Dawy, "Implementation and evaluation of cooperative video streaming for mobile devices," in Proc. of IEEE PIMRC, Cannes, France, September 2008.

[17] S. Li and S. Chan, "Bopper: wireless video broadcasting with peer-to-peer error recovery," in Proc. of IEEE ICME, Beijing, China, July 2007.

[18] S. Ioannidis, A. Chaintreau, and L. Massoulie, "Optimal and scalable distribution of content updates over a mobile social network," in Proc. IEEE INFOCOM, Rio de Janeiro, Brazil, April 2009.

[19] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, G. Pei, and A. Srinivasan, "Cellular traffic offloading through opportunistic communications: a case study," in Proc. of ACM Workshop on Challenged Networks (CHANTS), Chicago, IL, September 2010.

[20] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: social-based forwarding in delay tolerant networks," in Proc. of ACM MobiHoc, Hong Kong, May 2008.

[21] C. Boldrini, M. Conti, and A. Passarella, "Exploiting users' social relations to forward data in opportunistic networks: The hibop solution," in Proc. of Pervasive and Mobile Computing, October 2008.

[22] R. K. Lomotey and R. Deters, "Architectural designs from mobile cloud computing to ubiquitous cloud computing - survey," in Proc. IEEE Services, Anchorage, Alaska, June 2014.

[23] T. Penner, A. Johnson, B. V. Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in Proc. IEEE GLOBECOM, Austin, TX, December 2014.

[24] M. Satyanarayanan, S. Smaldone, B. Gilbert, J. Harkes, and L. Iftode, "Bringing the cloud down to earth: Transient pcs everywhere," in MobiCASE'10, 2010, pp. 315–322.

[25] E. Miluzzo, R. Caceres, and Y. Chen, "Vision: mclouds - computing on clouds of mobile devices," in ACM workshop on Mobile cloud computing and services, Low Wodd Bay, Lake District, UK, June 2012.

REFERENCES (Continued)

[26] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in mul- tihop radio networks," IEEE Trans. On Automatic Control, vol. 37, no. 12, December 1992.

[27] "Dynamic server allocation to parallel queues with randomly varying connectivity," IEEE Trans. on Information Theory, vol. 39, no. 2, March 1993.

[28] M. J. Neely, E. Modiano, and C. Li, "Fairness and optimal stochastic control for heterogeneous networks," IEEE Trans. on Networking, vol. 16, no. 2, April 2008.

VITA

NAME: Ajita Singh

EDUCATION: B.E., Electronics and Communication Engineering, Birla Institute of Technology, Mesra, India, 2013

M.S., Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, 2016

TEACHING: Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois:

Graduate Teaching Assistant – ECE 436 Computer Communication Networks II, Spring 2015

HONORS: Tuition and fee waiver for Graduate Students, Fall 2014