An Implementation of Utility-Based Traffic Shaping on Android Devices

Andrew M. Pham

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

T. Charles Clancy, Chair
Robert W. McGwier
Joseph G. Tront

July 1, 2014
Blacksburg, Virginia

Keywords: traffic shaping, LTE, mobile applications, quality of service, Android
Copyright 2014, Andrew M. Pham

An Implementation of Utility-Based Traffic Shaping on Android Devices

Andrew M. Pham

(ABSTRACT)

Long Term Evolution (LTE) was designed to provide fast data rates to replace 3G service for mobile devices. As LTE networks and the user base for those networks grow, it becomes necessary for the resources used for those networks to be used as efficiently as possible. This thesis presents an implementation which utilizes an algorithm extended upon the Frank Kelly algorithm to determine resource allocation for UEs and shapes traffic for each UE to meet those allocation limits. The implementation's network represents what an LTE network would do to manage data rates for a UE through a distributed algorithm for rate allocation. The main focus of the implementation is on the UE, where traffic shaping limits application rates by an elastic or inelastic classification through the use of Hierarchical Token Bucket (HTB) queuing disciplines.

Acknowledgments

I would like to thank my committee chair, Dr. Clancy, for assisting and advising me during my graduate program. I'd also like to thank Dr. McGwier and Dr. Tront for being on my committee and for the advising provided in the traffic shaping project.

I'd like to acknowledge the other members who worked on the AMFI project. In particular, acknowledgments go to Ahmed Abdelhadi, Ravi Tandon, and Michael Fowler for the technical assistance they provided.

Special thanks go to my fellow peers: Seth Hitefield and Sergio Bernales for their contributions in the original work accomplished in the first design of the project.

And of course, I'd like to thank my family and friends for their support.

See You Space Cowboy...

Contents

List of Figures vi

List of Tables viii

List of Abbreviations ix

1 Introduction 1

1.1 Background 1

1.2 Motivation 2

1.3 Thesis Organization 5

2 Related Work 6

2.1 Traffic Engineering 6

2.2 Traffic Engineering Tools and Implementations 10

2.3 Distributed Algorithm 12

2.4 Utility Functions 14

2.5 Thesis Contributions 18

3 System and Architecture 20

3.1 User Equipment 21

3.1.1 Background Service 22

3.1.2 Queuing Disciplines 24

3.2 Base Station 27

3.2.1 Rate Allocation Daemon 28
3.3 Resource Broker 28
3.3.1 Implementation States 29
3.4 Android System 30
3.4.1 Android System Architecture 31
3.4.2 User Application 32
3.4.3 System Service 33
3.4.4 Native Root Level Daemon 34

4 Methodology and Results **36**
4.1 Equipment 36
4.2 Testing and Verification 37
4.3 Analysis 43
4.4 Evaluation 49

5 Conclusion **52**
5.1 Summary 52
5.2 Future Work 53

Bibliography **54**

List of Figures

1.1 Cisco's projected mobile traffic growth by category . . . 3

1.2 NTT Docomo's LTE subscription growth as of 2012 . . . 3

2.1 Inelastic utility function with varying parameters . . . 16

2.2 Elastic utility function with varying parameters . . . 17

3.1 Basic overview of system components . . . 21

3.2 Standard packet routing flow . . . 22

3.3 BidService `onHandleIntent()` actions . . . 23

3.4 HTB hierarchy . . . 25

3.5 Base station layout . . . 27

3.6 Resource broker states . . . 29

3.7 Layers of the Android software stack . . . 30

3.8 Traffic shaping communication . . . 32

4.1 System layout . . . 37

4.2 Bids by each UE with 1 inelastic/elastic UE and 2 only-elastic UEs . . . 39

4.3 UE bitrates with all even allocations . . . 41

4.4 UE bitrates with an inelastic UE receiving a higher rate allocation . . . 41

4.5 Sample tc qdisc listing from a UE . . . 42

4.6 Sample tc qdisc class listing from a UE . . . 42

4.7 UE bitrate during streaming audio measured locally . . . 43

4.8 UE bitrate during streaming audio as measured by base station . . . 43

4.9 Partial derivative of inelastic utility function with varying parameters 44
4.10 Screenshot of the Speedtest application on the UE 46
4.11 Packet transmit and receive rates during speed test without traffic shaping . 46
4.12 Packet transmit and receive rates during speed test with traffic shaping . . . 46
4.13 UE maximum bitrates with no traffic shaping 47
4.14 UE maximum bitrates with traffic shaping 47
4.15 UE bitrate with applications using both inelastic and elastic HTB classes . . 48
4.16 Bitrate for three UEs using the base station's 10 Mbps rate allocation 50

List of Tables

4.1 Network rate allocations for all elastic UEs 41
4.2 Network rate allocations for 1 inelastic UE and 2 elastic UEs 41
4.3 Network rate allocations with 1 strict elastic UE and 2 flexible inelastic UEs 44
4.4 Network rate allocations with 2 inelastic UEs and 1 elastic UE 48
4.5 Network rate allocations for 2 inelastic UEs and 1 elastic UE 50

List of Abbreviations

ADB Android Debug Bridge
AIDL Android Interface Definition Language
AOSP Android Open Source Project
CBQ Class-based Queuing
DiffServ Differentiated Services
eNodeB enhanced node B
FIFO First In, First Out
GCM Google Cloud Messaging
HD High Definition
HTB Hierarchical Token Bucket
IFB Intermediate Function Block
LTE Long Term Evolution
netd Network Daemon
OFDMA Orthogonal Frequency Division Multiple Access
PID Process ID
qdisc Queuing Discipline
SFQ Stochastic Fair Queuing
TBF Token Bucket Filter
UE User Equipment
UID User ID
VOIP Voice Over IP
WFQ Weighted Fair Queuing

Chapter 1

Introduction

Traffic shaping is a method of regulating how resources in a network are used to ensure that the network is not overwhelmed any one user or group of users. Traffic shaping is generally used in larger networks such as those managed by internet service providers or larger private networks. In recent years, the growth of both internet capable mobile devices and high speed mobile networks means that resources for those mobile networks must be managed properly. Maintaining an adequate quality of service would benefit both users of those devices and the maintainers of the mobile networks.

Most traffic shaping generally occurs further within a network core. On the edges of the network where the users sit, traffic generally flows with less restriction which can cause congestion if an edge node has too many users. Expanding and improving the usage of traffic shaping to more steps along the network would mitigate the effects of congestion and better utilize bandwidth and other network resources.

1.1 Background

Long Term Evolution (LTE) is a wireless standard that has seen very quick growth in use for cellular networks in the past several years. LTE has a component called an enhanced node B (eNodeB) that serves as a base station and sits at the edge of the mobile network. Devices that communicate with when using the mobile network are called user equipment (UE). UEs can be phones, tablets, or any other device which uses the mobile network. Each eNodeB has a certain amount of bandwidth it can manage for data going to and from every UE connected to it.

Traffic engineering, also known as teletraffic engineering, is a topic that has been studied for many years and is a topic that contains on two ideas called traffic shaping and traffic policing. In particular, traffic shaping will be the focus in this implementation since traffic shaping

works by delaying traffic while policing discards traffic that does not meet its criteria. It is generally paired with queuing theory to determine what types of structures are best to shape packets and manage traffic flow in network pipes. A queuing discipline (qdisc) is a scheduler that is used for flow management on a device's network interface. Since traffic shaping has been used for many years, there are several types of qdiscs which can be used to manage network traffic.

The most commonly used queuing discipline is First In, First Out (FIFO) which queues and serves packets in the same order they arrive. They generally do not shape or modify traffic in any way which makes them simple for processing network packets. There are also several queuing disciplines which work in a more sophisticated manner such as those based on the Token Bucket Filter (TBF), which can more efficiently and fairly schedule the processing of packets going through a network link.

In the growing smartphone market, the operating system that has gained the majority of the market share is the Android mobile operating system. The Android Open Source Project (AOSP) allows any user or group to make their own modifications and build their own versions of the Android operating system. Being an open-source and highly customizable system, most manufacturers use it as the basis for the operating system they ship with their devices.

Part of Android's open source roots also stem from the fact that the Android kernel is heavily based on the Linux kernel, which also contains support for various queuing disciplines for traffic shaping. Most of these queuing disciplines and supporting kernel configuration information are detailed in [1]. Having an open source operating system and kernel, which supports most of the options the Linux kernel supports, means that adding traffic shaping capabilities to phones running Android is possible.

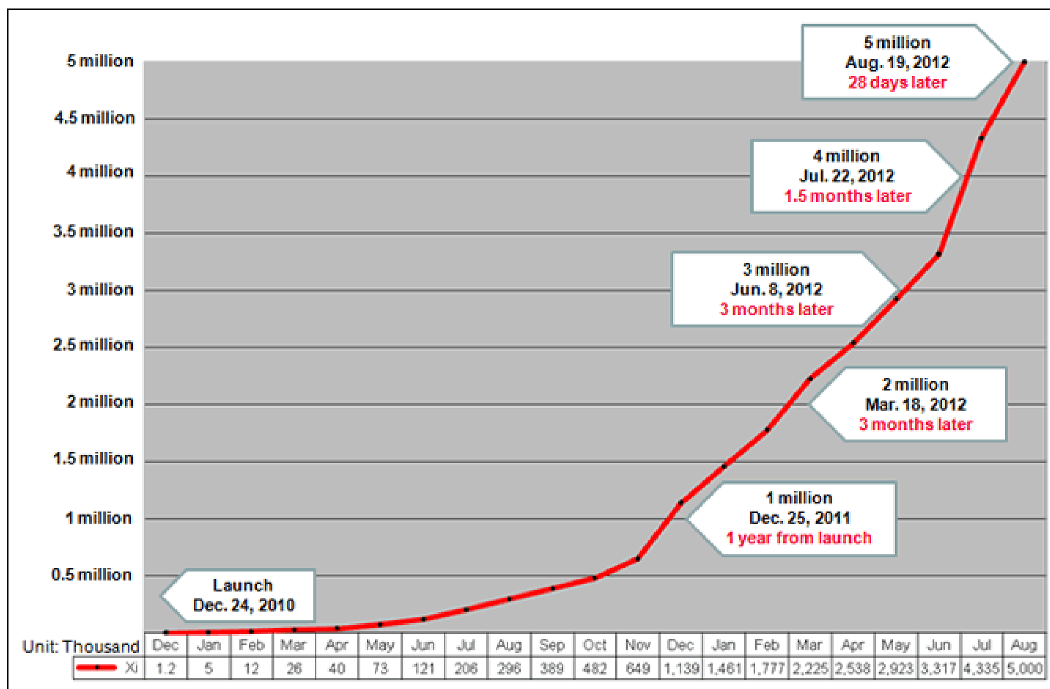
1.2 Motivation

The quick growth of both LTE and smartphone technology is the main motivator behind the ideas that will be presented in this thesis document. As of 2012, Gartner's reports show over 1.7 million smartphone units sold worldwide in 2011, an increase over the 1.5 million units from the previous year [2]. With the rapid growth of smartphones, especially those using LTE, resources on mobile networks will be used more heavily for data since LTE is all-IP.

Mobile data as a whole is growing rapidly with the growth of LTE networks and smartphone technology. Cisco's mobile traffic report has shown an increase in just video traffic alone from less than 50% of all mobile traffic globally in 2012 to 53% in 2013; their traffic analysis also projects data for 2018, predicting that traffic volume will grow by about 11 times that of 2013's traffic [5].

Figure 1.1 presents a projection of the growth of IP traffic on mobile networks based on traffic

Figure 1.1: Cisco's projected mobile traffic growth by category used with permission of [3]



Unit: Thousand	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Xi	1.2	5	12	26	40	73	121	206	296	389	482	649	1,139	1,461	1,777	2,225	2,538	2,923	3,317	4,335	5,000

Figure 1.2: NTT Docomo's LTE subscription growth as of 2012 used with permission of [4]

data in 2013. As shown in the data collected by Cisco, 2013 traffic data and projections for the following years show that video traffic is responsible for a majority of all traffic on mobile networks [3]. The rise in traffic for heavier media is allowed by the advances in mobile technology and networks.

The rise of LTE in particular has facilitated the growth of both aspects mentioned since faster speeds allow users to use the internet for more bandwidth heavy applications such as audio and video. A sample of this growth can be seen in Figure 1.2. This figure shows the growth of subscribers for NTT Docomo's LTE service from its launch in 2010 until 2012 when the number of subscribers hit 5 million [4]. With LTE's growth moving at such a fast pace, the infrastructure for mobile networks must be adequately maintained to provide acceptable service for users. Thus, an implementation is proposed to manage resource allocation to mobile users and optimize bandwidth usage.

The implementation takes into account classes of applications to manage rate allocation between heavy and light resource users. Heavy users need more bandwidth to use video applications but not too much that they use more than they need. Likewise, light users which may only browse the web would need less bandwidth but too little that they cannot browse anything. These ideas would help to improve management of LTE resources which might also benefit the upcoming LTE-Advanced standard. Being able to do so will ensure that adequate quality of service is available to users of an LTE network.

Within the smartphone and smart device ecosystem, the operating system that has grown the fastest is the Android mobile operating system. With its growth, it took over as the majority share of common operating systems in 2010 [6]. Being an open-source platform has allowed many vendors to pick it up as a system to distribute on their devices.

The traffic shaping implementation uses Android devices since the open nature of the Android mobile operating system and its kernel makes the necessary modifications to allow the use of traffic shaping utilities is possible. This is made easier by the fact that kernel options to allow and use various shaping and queuing schemes already exist in the base Linux and Android kernels. By hitting all of the clear advantages of the aforementioned technologies, the end goal of improving quality of service via traffic shaping from the end user's perspective is realistically achievable.

Studies such as those in [7] and [8], show that most users frequently use applications for web browsing and text communication. However, applications for streaming media are also a main consideration despite their lower usage frequency as they use up more of the user's available bandwidth accounting for over half of all mobile traffic as mentioned previously. As LTE usage grows, typical usage of mobile networks will begin to resemble those of traditional wireline networks. More users will be accessing internet services and the amount of data transferred to mobile clients will increase heavily over time such as seen in [5].

Applications can be classified into two general categories for how they handle network traffic: elastic and inelastic. Application elasticity is a description that determines how susceptible

an application is to changes in the rate at which it receives data. An elastic application is one that does not require very high data rates to perform reliably and does not have strict delay requirements in when that data should be received. A web browser being used to view primarily text-based web pages is a typical example of an elastic application. If the data for a web page were to be delayed by a few seconds, most users would not notice a delay had occurred.

An inelastic application is one that has higher rate requirements and a lower tolerance for delay in the arrival of data it is requesting. When those requirements are not met, the user can easily notice a decrease in quality of the received data. Applications for services such as video streaming or voice over IP (VOIP) are common examples of inelastic applications. Since these applications require data to be streamed at a reliable rate and with little delay, any delay or reduction in transfer rate of data would disrupt the user experience.

1.3 Thesis Organization

This thesis presents an implementation which aids with traffic shaping by applying the schedulers on the UE itself. By determining the utility of an application being used on the UE, the UEs will negotiate how much of an eNodeB's resources it requires to reduce load on the link managed by the eNodeB.

This document will go over related work and background research with regards to traffic shaping and mobile networks in Chapter 2. One of the main topics will be the algorithms for the rate negotiation process found in [9]. Chapter 3 will provide background information on parts of the implementation's architecture before leading into details on the functions of the architecture's components. Following this is the process of testing the implementation and its results in Chapter 4.

Chapter 2

Related Work

Traffic shaping is the focus of the implementation to be shown on the UEs. However, traffic shaping is itself a broad aspect of traffic engineering. Its use is fairly common in network administration and there are many queuing disciplines for shaping traffic as well as tools which use those queuing disciplines. As networks have moved from a more wired infrastructure to include wireless networks, traffic engineering techniques have been analyzed and utilized in wireless networks as well due to their viability.

The UE implementation uses traffic shaping to enforce rates determined in an algorithm presented in [9]. The algorithm uses specific utility functions to determine a UE's desired shared of the total resources to be assigned. The algorithm and the utility functions are also covered in this chapter as they contribute to major functionality of the implementation.

2.1 Traffic Engineering

While traffic shaping is the key focus in this thesis, traffic shaping is actually a subset of the topic of traffic engineering. Traffic engineering, in the context of computer networks, is the focus of finding ways to manage a network to optimize traffic flows for all users of the network. It has evolved over time as networks have evolved and mainly deals with queuing theory and various network technologies. The use of traffic shaping and policing received more focus as networks quickly grew and required control to ensure quality of service [10]. Traffic shaping became a tool to control rates by enforcing a delay while traffic policing harshly enforces rates by also discarding packets which do not conform to its rules. Traffic shaping will be the focus as it is used to control data rates without dropping traffic that the user wants.

Most wired networks that utilize traffic shaping use a token bucket or leaky bucket based system on their nodes. The design for the token bucket has been analyzed and implementa-

tions of it have been used widely since its inception. Its ability to shape traffic according to certain parameters based on rate and burst demands make it popular for managing packet based traffic. A general overview of a common implementation, called the token bucket filter, can be found in Section 2.2.

Schedulers based on the token or leaky bucket are common but they are not the only types of schedulers available. Previous research into schedulers has also resulted in classless schedulers for meeting constraints by fairness. Consideration of traffic classes to queue traffic by priority has also resulted in classful disciplines.

Classless disciplines queue traffic as they arrive and all traffic is treated in a similar manner. Without traffic shaping, most queues act in a FIFO manner and without restriction in how fast the packets pass through the queues. To prevent queues from exceeding their limits fair queuing was brought into consideration to manage packets in queues [11]. Schedulers such as the Weighted Fair Queuing (WFQ) and Stochastic Fair Queuing (SFQ) qdiscs resulted in queues which would shape traffic with reasonable fairness for different packet flows on a link. This also serves to allow the queues to conform to rate requirements based on parameters for those qdiscs.

There also exists a standard which can be used for quality of service controls called Differentiated Services (DiffServ) which helps in qualifying traffic. El-Gendy et al. cover DiffServ and a comparison of many schedulers to achieve quality of service, especially for real-time and streaming applications [12]. In IP networks, DiffServ can be used in conjunction with shapers and policers to continually maintain traffic flow at different hops. While there are many methods and utilities that can be used in traffic engineering, the main type used in this thesis will be shaping based on the idea of the token bucket.

TBF is a common classless qdisc since it is based on the token bucket scheme. The token bucket was subject to much analysis as a use for generalized processor sharing as well as its packetized version in [13]. Generalized processor sharing provides an ideal model for schedulers with fluid traffic. Since fluid traffic is not possible in the packet-based internet, several schedulers have resulted from the original work on packetized generalized processor sharing that can be used in traffic engineering.

The analysis of the token bucket through multiple hops is expanded upon in [14] which proves that use of multiple token bucket schedulers are viable which has allowed the case where token bucket implementations are prevalent enough to be used at different segments of networks while still preserving quality of service expectations. A property of the token bucket algorithm is that it is work-conserving, which allows tokens that are saved to allow a burst of transmissions at the start of transmission after idle periods. This works well with packet networks like the internet where burstiness is common for most web traffic.

There also exists the classful variant of the TBF called the Hierarchical Token Bucket (HTB) which has seen use recently due to the capability of filtering traffic by classes and priority. A main benefit of HTB is that it allows resource sharing between classes to make it a good fit

for the diverse types of traffic on the internet. [15] provides an analysis of HTB throughput for traffic classes to show successful shaping on a small link.

The use of schedulers in networks prevents packet flows from oversaturating a queue. This greatly helps in preventing all traffic on a queue from exceeding the link's limits due to unrestricted packet flow. Work such as in [16] and [17] show how network architectures can benefit from traffic shapers.

The use of schedulers in networks prevents packet flows from oversaturating a queue. This greatly helps in preventing all traffic on a queue from exceeding the link's limits due to unrestricted packet flow. Work such as in [16] and [17] show how network architectures can benefit from traffic shapers. Most work expanding on analysis of schedulers have shown that shaping through multiple nodes can still limit rates while reducing the impact of delay as well as techniques to use schedulers to achieve an acceptable level of quality of service. Several of the schedulers mentioned are also implemented by Linux traffic control tools, which are covered in Section 2.2.

Work related to traffic shaping has also progressed to be feasible in wireless networks since their reliability has been noted in wired networks. Research into traffic shaping on wireless networks has included a variety of environments including wireless LANs and mobile networks.

Queuing disciplines were mostly formed before wireless networks became more commonplace. Despite this, studies of HTB in wireless networks have shown that they are a viable option for managing network traffic such as in IEEE 802.11 wireless networks [18], [19]. With how less reliable wireless networks are, Garroppo et al. show that having schedulers which shape traffic can improve performance and keep its intended sharing of bandwidth even if network conditions may be less favorable. Work by Valenzuela et al. shows examples of classful queuing using HTB to allow more fair service amongst users, especially when one user does not need more than a minimum required rate.

While used less often, traffic policing can also be used to maintain quality of service. [20] incorporated traffic policing using the token bucket as well as a couple of classless schedulers on UEs based on priority classes to meet quality of service requirements in IEEE 802.16. In wireless networks, many clients may be connected to a single base station. The use of schedulers on either the uplink or downlink for wireless clients is a reliable way of ensuring all users are able to use their link adequately.

Another common theme that appears in the use of traffic shaping on wireless networks is usually to ensure fairness and the overarching goal of this thesis's implementation is the same. On IEEE 802.11 networks, traffic shaping has used rate limiting to allow multiple clients to use a link evenly. Blefari-Melazzi's work shows how rate limiting connections affects TCP connections as well as allowing room for multiple connections to use uplink and downlink bandwidth [21]. This result is something that will be touched upon in the results in Chapter 4.

Traffic shaping has also been explored on mobile networks as well. Li and Stol have shown a use of UE traffic shaping with traffic policing at the RNC on 3G networks [22]. They also explain the tradeoffs between delays and shapers which allow fast enough traffic for heavier classes like those from video applications. Since wireless is inherently less reliable than wired networks, traffic shapers should be set up to account for various types of traffic, available bandwidth and other conditions.

Traffic shaping can be used at any point of a network but most of the effects can be seen on the edges. Because of this, the effects of traffic shaping mechanisms and schedulers are more noticeable on the core routers which sit at those edges. Doing this allows them to shape them where they have the most control over the traffic flows. In previous works involving traffic shaping on LTE, the tendency is to use the eNodeB as the point in which to apply quality of service adjustments to existing schedulers and utilizing LTE's orthogonal frequency division multiple access (OFDMA) base.

An example of this is in Zaki et al.'s work on modifying the scheduler for an eNodeB to differentiate traffic classes [23]. This relies more on other services such as DiffServ and other protocols to check traffic on the UEs. More of the quality of service control then lies with the eNodeB rather than the UEs. The implementation in this thesis aims more to categorize and separate applications locally on the device leave the eNodeB to just shape downlink rate to meet the rate allocation.

There is also work by Luo et al. which also covered modifying the downlink schedulers in LTE networks and resource allocation for users connected to an eNodeB [24]. Research into downlink scheduling is common due to LTE's use of OFDMA to schedule users on a link. Both [23] and [24] target taking into account available resources and channel conditions for the node as well as determining and traffic types going down to the UEs. The dependency on being able to check different layers of traffic adds dependency on how the UE traffic is formed and how the eNodeB works with the traffic.

Some work has also been done on applying schedulers to the uplink portions of a node as well such as in [25]. These generally consider UEs in groups and apply resources to be allocated depending on their traffic. In all of these cases, the eNodeB shapes traffic globally for an entire user (or groups of users) based on allocated resources and how much each user is using.

The algorithm in Section 2.3 will be a main parallel to the related work mentioned so far in relation to LTE. The algorithm will result in traffic shaping which occurs at the node which lies on the edge of the core network. The core network being simulated in the implementation is to represent what would happen if the system for distributed traffic shaping among the UEs was on an LTE network.

The core network will shape the downlink via the representation of the eNodeB. Since rate allocations are assigned per UE, the downlink shaping works to reserve and control the UE's total download rate. The work of differentiating and classifying applications is left to the UE as the UE itself has the most control and knowledge of its own traffic. The main division

of traffic is through the classification of inelastic and elastic applications.

Consideration of elastic traffic has already been taken into account with regards to mobile traffic. Most of it focuses on quality of service at the mobile network's point of view, such as in [26], which analyzes elastic traffic on OFDMA. Earlier works looked at just elastic traffic since most traffic was web based. Focus on inelastic traffic increased as streaming traffic grew for video. A big contributor to the analysis of inelastic traffic comes from the Frank Kelly algorithm for distributed rate allocation [27]; this algorithm is also a big contributor to the algorithm in [9], which is the basis for the algorithm used for the implementation in this thesis.

The implementation uses utility functions to represent how much usefulness an application receives from a specific rate allocation. Elastic applications tend to have a concave utility function which allows an optimal rate to be determined in the range of the function. The inelastic utility function uses a sigmoidal function which is convex before its inflection point but concave after it. The use of both types of functions has been seen before such as in [28]. More on the topic of utility functions is covered in Section 2.4.

Traffic classification for applications is common when traffic shaping is used. Among queuing disciplines, there are classful and classless queuing disciplines. The main focus will be on a couple of classful queuing disciplines to account for elastic and inelastic traffic. Several Linux tools for network management and policing have support for classful queuing based on identification based on application properties relative to the system it is on [29].

2.2 Traffic Engineering Tools and Implementations

The token bucket scheduler works on the idea that a bucket contains a certain number of tokens which are generated at a specific rate until it reaches its maximum limit. When a packet arrives, a number of tokens are claimed to allow the packet to pass through the scheduler. Some of the scheduling behavior for TBF can be seen in [30], which uses the token bucket as a way to model most schedulers and their end-to-end delay.

For what occurs in the schedulers, if tokens are not available, then the packet is delayed until more tokens are generated. On the other hand, when the token bucket is full, packets may be sent through the queue immediately to allow a burst of packets to be sent. After the burst period, traffic is then regulated by being sent only as fast as tokens can be generated. This simplistic functionality is why the token bucket algorithm is one that is commonly seen in traffic shaping configurations.

Most of the token bucket's properties as a scheduler, although it is labeled as the similarly functioning leaky bucket, can be found in [13]. As it is a reliable scheduler for traffic shaping, it has been widely analyzed in many forms of networks. TBF is a classless scheduler that does not differentiate traffic flows by any properties or classification. HTB is a classful

queuing discipline derived from both the token bucket filter and uses ideas from the class-based queuing (CBQ) queuing discipline. It uses the token bucket from the former to shape traffic but allows for classful queuing based on the latter. Because of its classful queuing, it is a more commonly used classful queuing discipline in traffic shaping trees since TBF is classless.

The implementation to be covered resembles some of the ideas mentioned in that rates will be applied per user via resource allocation but rather than applying HTB queuing disciplines on the network links per user, they will be applied on the UE to take advantage of the classful nature of HTBs. Queuing disciplines have been applied on LTE in conjunction with resource allocation, but the UE implementation will shape the rate on a smaller scale to improve quality of service per user. Combining this with the resource allocation shaping would also improve quality service at the eNodeB.

Traffic shaping is used by network administrators to control how much of a network's resources are used by any particular user or process. The machines running the queuing disciplines for traffic shaping are typically Linux machines as the Linux kernel supports the implementations of those queuing disciplines and other features for managing quality of service via kernel modules. Most of the functionality provided can be accessed in the user-space via utilities such as iproute2 and tc [1]. Since the Android kernel comes from Linux roots, it also has access to these utilities if they are cross-compiled for the system.

The tc utility allows root users to create, modify, and remove queuing disciplines. Qdiscs may be set up in a hierarchy where they have children qdiscs or classes, if they are classful qdiscs. The utility has several classless and classful qdiscs available which include some of the qdiscs previously mentioned, such as TBF, SFQ, HTB, and CBQ. In the qdisc hierarchy, the root classes generally have their ceiling rate to be the maximum rate for all traffic moving passing through the queue. Children classes have a subset of that ceiling rate with varying guaranteed rates based on how much rate should be assigned to those children classes.

In the implementation presented in this thesis, application classes are represented by separate HTB classes to split the whole rate allocated to the UE. Details on this can be found in Section 3.1.2. The tc utility can also set filters so that matching traffic is sent to the correct class. The filters also work with the iptables utility and netfilter rules. Netfilter's hooks can also match the filters to set connection marks to process traffic through specific iptables chains. In the implementation, this is used to direct traffic to the correct class on egress and ingress.

Application of traffic shaping to mobile devices is not as commonplace since the scale of traffic shaping is usually done to increase quality of service in a network and not just a single device. A few works have applied traffic shapers to mobile devices though such as in [31], where the authors apply traffic shaping with hooks closer to the kernel and analyze the effect on power consumption.

The work in this thesis uses hooks at various levels of the Android operating system to apply

the traffic shaping and aims to separate applications into traffic classes to improve quality of service. Most of the underlying tools are similar to those used by Vergara et al. though due to the commonness of the utilities such as netfilter hooks to apply packet marks for iptables.

An important aspect to be covered in Section 2.4 is that the implementation will utilize a server-side algorithm presented in Section 2.3 along with a traffic shaping algorithm locally on the UE to improve quality of service on both sides.

2.3 Distributed Algorithm

The algorithm and results in [9] contribute to the main implementation in this thesis with regard to rate allocation on the network side. The algorithm defines the actions a UE and the resource broker, which is a server which determines the rate allocation for each UE, will undergo a process to negotiate a shadow price to determine rates for each UE. The bidding process uses the algorithms defined in the paper to define the behavior of the UEs and resource broker during the bidding states.

In particular, [9, Algorithm 3, 4] presents the exact functionality for the UE and resource broker, respectively. The original basis of the algorithm was in the Frank Kelly algorithm, which was designed with mainly elastic traffic in mind [27]. The Frank Kelly algorithm originally presented an algorithm to determine rate allocation for users in a network. The idea is that users each have their part of an algorithm in which they determine how much of the network's resources they need and what their desired price' is per unit of resource.

The shadow price is a parameter in the algorithm's function which contributes to the solution in finding an optimal rate for a user's utility function. The price represents how much each user would pay per unit of resource to be allocated. The rate desired by the user and the shadow price affect the result of the utility function for each user. The utility function is an equation which can be used to determine the optimal rate which achieves an adequate quality of service level. In Kelly's algorithm, utility functions represent how much positive gain the user receives based on the rate given to it so an optimal result would give the user the most gain for the price paid for the desired bandwidth. The idea behind the shadow price is what drives the main portion of the distributed algorithm in the implementation.

Kelly shows that the utility functions have a solution which is guaranteed with the convergence of the shadow price during the algorithm. The convergence is possible when the utility functions are strictly concave to allow for an optimal maximum to be found. Since this allows for an optimal rate to be determined for each user, Abdelhadi's extension allows the distributed algorithm to have an optimal and guaranteed rate allocation for all users which participate in the algorithm. The algorithm preserves the use of shadow pricing and utility functions presented by Kelly, but contains some expansion to incorporate inelastic traffic. Users with inelastic traffic use a sigmoidal-like utility function which has convex sections but is still strictly concave in the areas where the rate is most beneficial to the user.

The resulting algorithm is one that focuses on optimizing how much of the network's resources will be allocated to a UE based on its utility function whether that function is for inelastic or elastic traffic. Another aspect adapted from Kelly's algorithm is how each user's main utility function is the result of an aggregate of utility functions defined by the applications it is running. The utility functions for the UE are used to maximize utility while minimizing the rate it needs to request from the resource broker and these functions differ depending on whether the function is meant to represent utility for elastic or inelastic applications. The elastic utility function is a logarithmic-type function which is looser in the manner the function increases to fit the characteristics of elastic applications having a lower rate and/or delay requirement. The inelastic utility function is a sigmoidal-like function whose shape resembles that of a step function to better fit the conditions of inelastic applications requiring a minimal rate to service data adequately.

Both functions are chosen because the logarithmic function is strictly concave and the sigmoidal function is strictly concave past its point of inflection. Any point in the convex portion of the utility function's curve is considered as low utility. The concave nature of the functions allow there to be an optimal point within both functions that will satisfy a utility requirement based on certain rate for those functions. A more in-depth explanation of both utility functions can be found in [9]. In particular, the inelastic utility function is one of a sigmoidal form; [32] presents an example usage of the sigmoidal utility function in a wireless environment. Xiao et al. focus on a utility for power control, but as the definition utility is flexible, the idea can still be applied to the UEs in the algorithm. The utility for the UEs in this implementation is the usefulness of a rate allocation based on application requirements.

Abdelhadi covers utility functions for both elastic and inelastic applications. The resource broker algorithm is used as defined for an eNodeB in Abdelhadi's paper in the implementation for rate allocation in this thesis. The UE also uses the algorithm from the paper. In order to account for the mix of elastic and inelastic applications however, the UE implementation in this thesis also uses the utility function aggregation step to determine its overall utility during the main algorithm. The end result of the algorithm is what will be used to determine appropriate rates with which to shape traffic for applications on the UE.

The algorithm is a distributed algorithm that calculates a shadow price that serves as an objective measurement to determine how much of the total rate the resource broker can allocate can be assigned to a UE based on its utility. It is distributed in that the bid calculation occurs on the UEs and shadow price calculation occurs on the resource broker with their respective calculated values communicated iteratively until a steady state is achieved. The UE makes a bid depending on its utility function result and the shadow price stated by the resource broker. The resource broker takes these bids to determine the shadow price.

The resulting shadow price is used by each UE to determine its rate allocation. The resource broker uses each UE's bid and the shadow price to calculate each UE's rate as well. These results and the algorithms mentioned in this section are an important part of what determines how each UE uses and manages traffic shaping in the implementation in Chapter 3.

2.4 Utility Functions

The algorithm used to determine the rate allocation has two parts. One resides on the UEs and the other is on the resource broker. The algorithms are shown in Algorithms 2.1 and 2.2, which were originally defined in [9, Algorithm 3, 4]. The source provides details into the equations and proofs behind how the algorithm reaches an optimal steady state for the shadow price. This subsection will provide a more general overview to the equations and how they are used in the implementation. There are also some changes and additions to account for the variation in response time by the UEs and to partition the algorithm into segments for separation of logic.

Algorithm 2.1 User Equipment Algorithm [9, Algorithm 3]

```
Send initial bid ω_i(1) to eNodeB
loop
    Receive shadow price p(n) from eNodeB
    if STOP from eNodeB then
        Calculate allocated rate r_i^opt = ω_i(n) / p(n)
    else
        Calculate new bid ω_i(n) = p(n) · r_i(n)
        if |ω_i(n) − ω_i(n−1)| > Δω(n) then
            ω_i(n) = ω_i(n−1) + sign(ω_i(n) − ω_i(n−1)) · Δω(n) { Δω = l_1 e^(−n/l_2) or Δω = l_3/n }
        end if
        Send new bid ω_i(n) to eNodeB
    end if
end loop
```

Algorithm 2.2 Resource Broker Algorithm [9, Algorithm 4]

```
loop
    Receive bids ω_i(n) from UEs { Let ω_i(0) = 0 ∀i }
    if |ω_i(n) − ω_i(n−1)| < δ ∀i then
        STOP and calculate rates r_i^opt = ω_i(n) / p(n)
    else
        Calculate p(n) = (Σ_{i=1}^{M} ω_i(n)) / R
        Send new shadow price p(n) to all UEs
    end if
end loop
```

The algorithm starts when a bidding process is requested and the resource broker broadcasts a message to all UEs to start bidding. The resource broker then collects bids from all registered UEs and calculates the shadow price if the bids have differed from their previous bid by more than a particular threshold. For the initial bid, the previous bid is considered as no bid, or zero. If during any iteration of bid collection the difference between the previous bid and the current bid for all UEs is within the threshold, then the resource broker calculates the rate to be allocated to each UE and sends this information to the traffic shaping daemon on the base station. The resource broker also sends the final shadow price to each UE for them to calculate their own rates.

On the UE side of the algorithm, the initial bid from all UEs is the same and only begins to differ after they receive the first shadow price from the resource broker. The UEs use the shadow price to determine if the potential rate based on its bid compared to the shadow price will allow in a rate allocation that is adequate for the applications on the UE. If the potential rate is not adequate, the UE calculates a new price with which to bid by finding a desired rate that will meet its rate requirements. This calculation is based on a series of utility functions which can be of two forms in this implementation.

Applications are classified as either elastic or inelastic for consideration in calculation through the utility functions to determine the shadow price. As mentioned in Section 2.3, the inelastic utility function is of a sigmoidal-like form and the elastic utility function is of a logarithmic form as used in [9, Equation 1, 2] and are shown in Equations 2.1 and 2.2, respectively.

$$U(r_i) = \left(\frac{1+e^{a_i b_i}}{e^{a_i b_i}}\right)\left(\frac{1}{1+e^{-a_i(r_i-b_i)}} - \frac{1}{1+e^{a_i b_i}}\right) \tag{2.1}$$

The inelastic utility function resembles that of [32], but is normalized so that its values sit between 0 and 1. This allows both utility functions to have values within the same range for equal consideration. The inelastic utility contains parameters a_i, which controls how closely the utility resembles a step function, b_i, which controls the inflection point of the utility function, and r_i, which is the rate being used to determine the application's utility.

For inelastic applications which have a stricter requirement, a_i is usually higher to represent a high dependency on having a particular minimum rate. b_i may also be higher but is generally used to place the function's inflection point where the minimum rate requirement lies. For the implementation, inelastic applications could be further classified as having a strict or flexible requirement for rate and delay. Strict requirements would fit applications such as those used for live content where the minimum bitrate requirement is high and needs to be consistent to maintain adequate quality. A flexible application would be applications for streaming media where the impact of delay may be smaller.

Flexible applications with lower rate requirements would use parameters $a_i = 5, b_i = 2$ for a more step function-like curve with an inflection point which is at 2 Mbps. Strict applications would use $a_i = 3, b_i = 4$ for a less steep slope near the inflection point at 4 Mbps. The

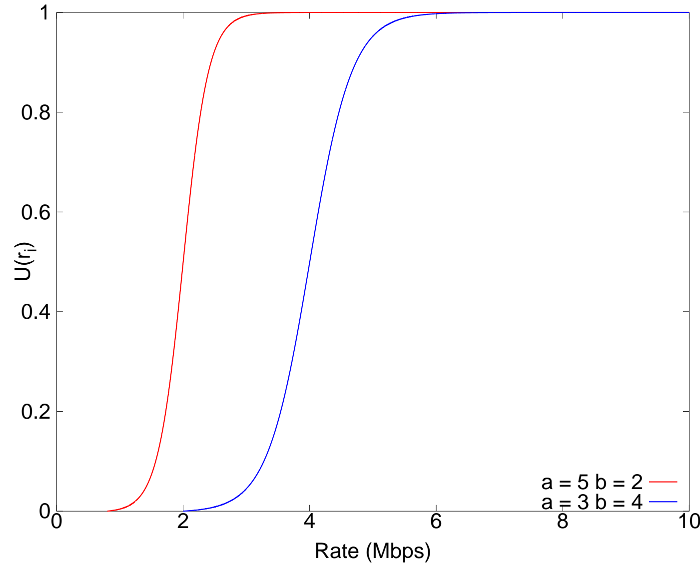


Figure 2.1: Inelastic utility function with varying parameters

slope is a bit looser since applications with higher requirements such as video streaming have less to gain from an increased rate once their rate is adequate. Figure 2.1 provides a visual to show the differences in utility based on the parameters. Since the base station in the implementation is limited to 10 Mbps, the inflection point for inelastic applications is placed to fit within the rate limits.

For high-definition (HD) video, the average bit rate may vary between 5 - 20 Mbps as shown in [33]. However, a minimum bitrate of 2 Mbps should still be viable for inelastic applications, such as for streaming video, since media content providers will have optimized the video via compression. Most videos would then be streamed with lower rate requirements to reduce bandwidth usage. Results of works such as [34] and [35] show studies of user perceived quality among some common video encoding formats; on mobile devices, even with screens that may be of HD resolution and higher, most users may not be able to easily discern changes in quality of the video being streamed.

$$U(r_i) = \frac{\log(1 + k_i r_i)}{\log(1 + k_i r_{max})} \tag{2.2}$$

The elastic utility function is an increasing logarithmic function set to have values that range between 0 and 1. It contains parameters k_i and r_i, where k_i controls how quickly the utility function grows and r_i is the rate used to determine a utility. The r_{max} term represents the rate where maximum utility is achieved, or where the utility is at its max value of 1. The k_i term can be changed so that higher values represent a less delay tolerant elastic application that would benefit from having a higher rate allocation.

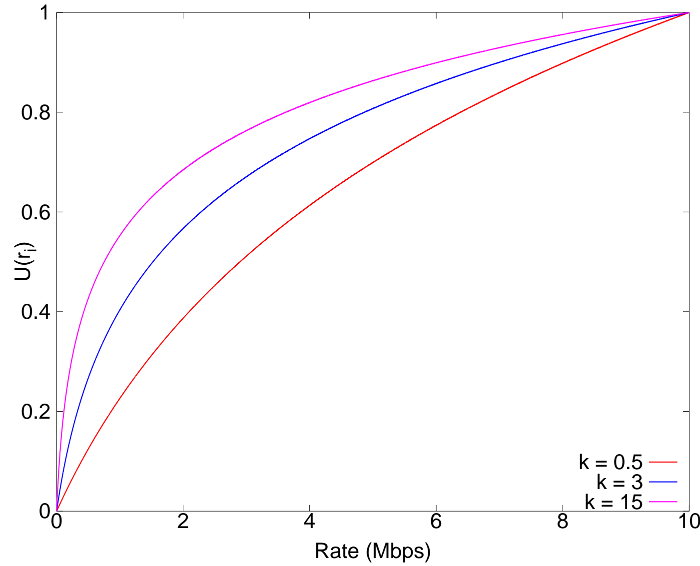


Figure 2.2: Elastic utility function with varying parameters

In a similar manner to inelastic applications, elastic applications may be classified into those having a low, medium, or high rate requirement. For the implementation, this would translate to differing values of k_i, where low sets $k_i = 0.5$, medium sets $k_i = 3$, and high sets $k_i = 15$; these values are borrowed from the simulations in [9]. Applications which may use network connections sparingly and in the background would be considered low priority whereas those which may act through user input could be considered medium or high priority. Figure 2.2 provides a visual to show the effect of the k_i parameter on the elastic utility function.

To find the total utility for the UE, the utility of each application is aggregated into a singular utility. The manner in which this is done is shown in Equation 2.3. Since the utilities are concave functions and the natural logarithms of those functions are also concave, they can be represented as a product of terms. Details on the aggregate utility and the properties of the functions being concave can be found in [9, Equation 3].

$$\sum_{i=1}^{M} \log(U_i(r_i)) = \prod_{i=1}^{M} U_i(r_i) \tag{2.3}$$

$$\frac{\partial}{\partial r_i} \left(\log(U(r_i)) - r_i p(n) \right) = 0 \tag{2.4}$$

Since the applications can be one of two types of elastic or inelastic, having both equations use the same range allows all utilities to be treated the same with regard to the total utility.

With the equations representing how to achieve a utility based on a given rate, the process of finding an optimal rate to achieve a specific utility requires the equations to be in a form to find the a rate which maximizes the utility in a manner as shown in Equation 2.4. The implementation uses the derivative forms since the main goal is to search for an optimal rate to achieve a max utility. The equations used in the implementation are shown in Equations 2.5 and 2.6.

$$\frac{\partial}{\partial r_i} \log(U(r_i)) = \frac{k}{(1 + kr_i) \cdot \log(1 + kr_i)} \tag{2.5}$$

$$\frac{\partial}{\partial r_i} \log(U(r_i)) = \frac{a(e^{ab} + 1)(e^{ar})}{(e^{ar_i} - 1)(e^{ab} + e^{ar_i})} \tag{2.6}$$

For the implementation, since finding an exact rate would be too restrictive, the conditions to meet Equation 2.4 are modified so that the terms are within a threshold of the value 0. The change in the value of the rate used to maximize utility would be trivial compared to what would actually be allocated so a small difference offset should still be sufficient.

The equations covered are what the components in the implementation's network use for rate allocation. Chapter 3 will detail the implementation specifics for both the UE and resource broker in how they use the algorithm, changes to the algorithm to fit the implementation, and extra actions taken by each component.

2.5 Thesis Contributions

This thesis provides an implementation which presents traffic shaping on UEs based on a negotiated rate between the UE and the network's resource broker. Unlike in previous works which used shaping on solely UE downlinks, the implementation uses downlink and uplink shaping to enforce a rate ceiling which matches the rate allocated to a UE. The setup is similar to that of work in [22], but uses shaping on both ends of the network rather than shaping and policing. In addition, all classification is done on the UE side which has direct knowledge of the applications involved in network traffic. There is no reliance on the network core to be aware of the traffic's classification or source. This would prevent cases of traffic not being able to be classified due to transport layer security for example

The manner in which UE traffic is classified is also more refined as applications are put into classes which allow multiple transmitting applications to enter queuing disciplines which match their classification. The use of multiple classifications affecting the allocated rate the shapers use results in a more refined shaping scheme on the UEs. The implementation using the shadow pricing algorithm itself from [9] also allows each UE to have a rate allocation which fits its needs better than having a generalized rate allocation based on its traffic at any instance in time.

The implementation also uses classful queuing disciplines. Earlier work used the classless TBF qdisc which works to shape all traffic but does not differentiate flows or take traffic priority and classes into account. Some recent works have shown the use of HTB in wireless networks such as [36]. The use of HTB in this thesis is on UEs which can send traffic through either their mobile network or through Wi-Fi. While the UEs are tested on Wi-Fi, all data traffic they send and receive is done in the same manner regardless of the interface used. The automatic management of HTB qdiscs on the UEs shows that traffic shaping for UE egress traffic is possible and an aspect to consider and examine in future work with UEs on LTE networks.

An important aspect of this implementation is that it presents the use of traffic shaping on existing devices with results that can be observed and measured. The structure and results of the implementation can be used as a framework to expand upon or supplement simulations in future work. The use of the Android operating system means that practical implementations could target real users in the future provided smartphone hardware and systems resemble current hardware and operating systems.

Chapter 3

System and Architecture

The implementation's system consists of three main components and a general overview of the three main components can be seen in Figure 3.1. The bidding process works to determine rates for each UE as each UE is sending a bid based on an aggregate utility as mentioned in Section 2.3. Since the resource broker only needs to know the rates for each UE and not how the rates are managed on each UE, the difference in granularity logically splits the various tasks apart into the two components.

The system is set to resemble that of an LTE network edge to demonstrate how the system could be applied in an LTE setting. Since an eNodeB would only manage UEs which connect to it, the resource broker exists as a different entity with its main task being to manage the server side algorithm and send rate allocation information to the eNodeB to be enforced. Most of the focus of the architecture will be on the Android phones representing the UEs that would connect with an eNodeB. In this implementation, the eNodeB is represented by an access point which shall be referred to as the base station.

The base station is a combination of a router and a computer acting as a DHCP server which controls the overall link rate to each phone or UE via commands it receives from the resource broker. The resource broker, which controls the shadow pricing algorithm as explained in Section 2.4, is run on another server that communicates with the computer running the DHCP server to apply queuing disciplines on the downlink to each UE.

Both the resource broker and the base station are placed on the same subnet allowing the resource broker to directly communicate to the base station's traffic shaping daemon. The same network is used by the base station to assign addresses to the UEs by DHCP, which could allow the UEs to communicate directly to the resource broker during the bidding process. The resource broker uses Google Cloud Messaging for Android (GCM) as it is a standard library used for push notifications to Android devices. A slight drawback is that this requires a path to the internet for both the resource broker and the UEs. In a practical implementation, another method of push notifications

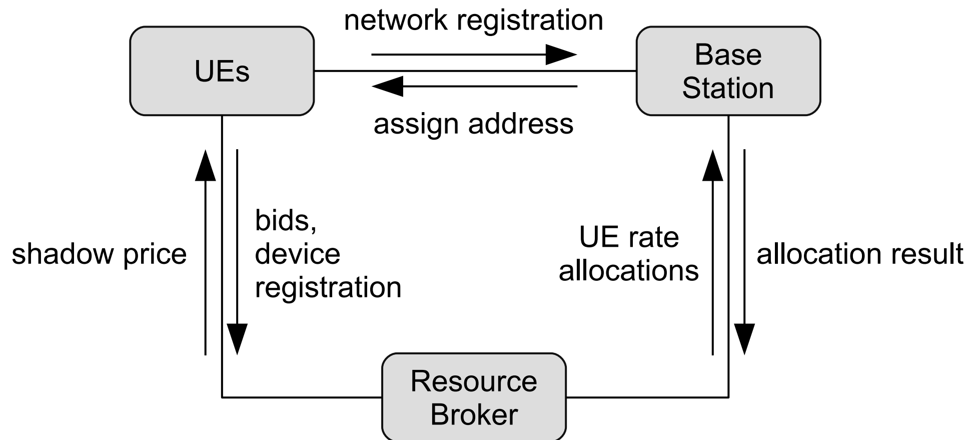


Figure 3.1: Basic overview of system components

3.1 User Equipment

For the implementation, Samsung Nexus S phones were used as UEs for their ease of modification since open source code is available for the operating system and kernel. The bootloader for the phones can be unlocked; having an unlocked bootloader allows easy flashing of a self-compiled operating system and kernel to use them on the phones. Each UE used in this system has a modified version of the 4.1.2 Android operating system through AOSP, specifically build JZO54K for the crespo device. They also use a custom build of version 3.0.31 of the Linux kernel provided by Google for the chipset family to which the Samsung Nexus S belongs. The UEs communicate with the base station via Wi-Fi to establish and maintain connectivity to the network and with the resource broker to participate in the shadow pricing algorithm.

The operating system was modified to allow for a custom system service to be added which handles communication between a user-level system application and a root level daemon which controls the traffic shaping. For this implementation, the user interface was not modified for practical use, but the user-level application could be used as a front end to display information from the background service to the user.

The main functionality for the application is to have the background service participate in the shadow pricing algorithm on behalf of the UE. Separating the tasks into the different layers of the architecture serves to make sure each component sits in a layer where they have the minimal permissions needed to perform necessary tasks. The division of tasks also serves to allow the root daemon to act without requiring authorization from the user. Normally, applications require explicit privilege escalation by the user provided the permissions have been unlocked for the device.

Kernel options were also modified to enable traffic shaping and quality of service modules

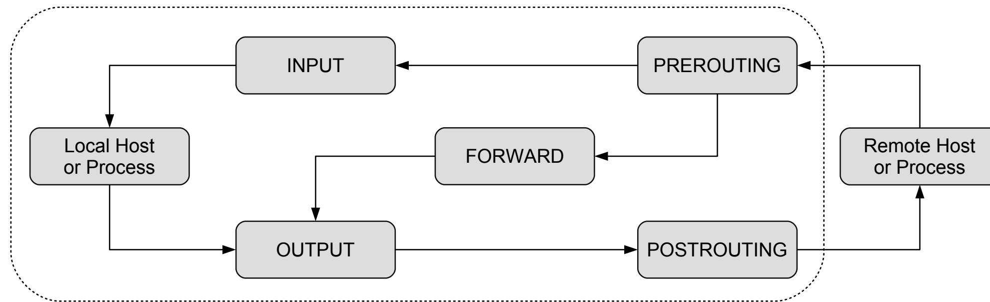


Figure 3.2: Standard packet routing flow

to be used on the phone. Specifically, the modules to allow for the use of the HTB qdisc and packet classification and marking for netfilter were enabled to be built into the kernel for the system. These modules allow for structures such as the queuing disciplines to be usable by the system through the tc command. Most of the components which contain the capability for enabling traffic shaping are a part of iproute2 or netfilter and the tc command works with those. Details on usage of the tc command in this implementation as well as the queuing disciplines are utilized is covered in Section 3.1.2.

There are also kernel hooks categorized under the name netfilter that are used for connection based packet routing at the kernel level. Netfilter contains the capability to mark packets which are used to filter packets through to the proper queuing disciplines via firewall rules set in iptables. Figure 3.2 shows how packets are normally routed through different iptables chains. Rules used to mark packets in this implementation are placed in the mangle table along with rules on the INPUT chain to retag incoming packets with the connection markers originally placed on outgoing packets.

3.1.1 Background Service

The UE's logic for participating in the shadow pricing algorithm all takes place in a service run by Broker Agent, the user-level application developed for the system. The service is run in the background on the device whenever the resource broker broadcasts a message requesting all UEs connected to the base station to participate in a new series of bids. A background service is used as the goal is to have the application participate in the rate negotiation for the device without user interaction.

The service contains two main methods that are invoked upon the service's creation: `onCreate()` and `onHandleIntent()`, which is called immediately after `onCreate()`. The `onCreate()` method contains functionality for the application to determine which applications are currently active or in memory on the device and store them to determine the phone's utility for the bidding process.

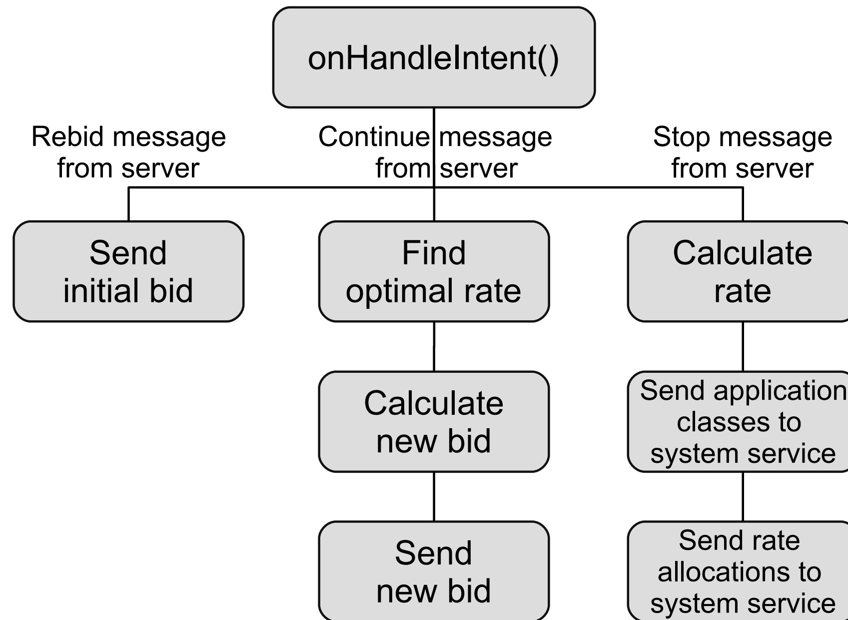


Figure 3.3: BidService `onHandleIntent()` actions

Only applications which are found to use the permission required for network access are taken into account for this process. In addition, system applications are ignored. This results in only user-installed applications to be classified. Most system applications are not applications a user would normally have in the foreground and none of them would be considered inelastic. The assumption is that if a system application needs to use the network, then the default elastic queue should be sufficient. Any user-installed applications are classified as either elastic or inelastic and then represented by Java classes which represent those applications and contain parameters to influence the utility calculation process depending on the classification.

Classification is manually determined to demonstrate the implementation's ability to use the utility functions and to have the utility functions impact the UE's bids. Under an automatic classification system, deeper modifications would be required into the Android operating system which would affect standardized APIs. The reason for the lack of proper classification is that the manner in which Android applications are packaged does not provide detail which is refined enough to create classifications on whether an application is elastic or inelastic.

Classification could become possible if the metadata structure for Android applications is modified to include application characteristics or categories. Several methods were utilized in testing the automatic classification of applications, however none proved viable for the implementation in tests. Thus, for the purposes of demonstrating the implementation's traffic shaping capabilities, the assumption is that applications are classified appropriately.

The service's `onHandleIntent()` method is where the main logic for the UE's part of the shadow pricing algorithm takes place. Any actions performed within that method are directed by the command sent by the resource broker through GCM; the GCM messages contain commands such as when to start or stop the bid and when the resource broker is expecting a bid from the UE. Figure 3.3 shows the possible actions the service can take depending on the message sent by the resource broker.

The default bid sent for the initial bid is statically set at 0.1 for all UEs for the very first iteration. Since UEs which have not been allocated resources are locked at 100 kbps by the base station, the initial bid to start is always set at 0.1 since calculations and price is done per megabit. All calculations done on the resource broker are also calculated on the megabit scale with regards to the shadow price.

The calculation of an optimal rate follows the idea from Equation 2.4. The standard form of the utility function is not used but is still relevant for being able to objectively compare utilities. Using the derivative form of the utility function, the result is compared with the current shadow price to determine if the rate being tried is indeed optimal. In the original equation, the result of the comparison must be 0 in order for the optimal value to be found. Due to the nature of floating point values on computers, finding the value which results in a direct match is infeasible. As a result, an optimal rate is found when the comparison is within a threshold defined in the implementation. The assumption with the threshold value is that the difference in the optimal rate is negligible when the final rate to be allocated is decided.

The rate allocation step is when the application passes messages to the native service by sending them to the ShapeService in the system server. Since each application is assigned a unique user ID (UID) at installation time, the UID is used to uniquely identify each app during classification. Each application has its UID and classification sent to the ShapeService to have filters set to send traffic to the correct queues. Since netfilter and iptables support tracking by UID, the use of UID with classification can be used with those capabilities. Process ID (PID) tracking is less reliable as applications can be killed and restarted which would change their PIDs. The lack of support by netfilter and iptables for PID tracking in the Android kernel also prevents its usage. For long term filtering for the qdiscs, UID tracking works for application-based shaping.

3.1.2 Queuing Disciplines

HTB is the primary queuing discipline used in creating the traffic shaping hierarchy on the UE. HTB's classful queuing allows differentiation of traffic through the use of filters rather than treating all traffic in the same manner as a regular token bucket qdisc would through TBF. The work conserving nature and the allowance of token sharing allows the best usage of resources to make sure no traffic is delayed any longer than is necessary for traffic shaping.

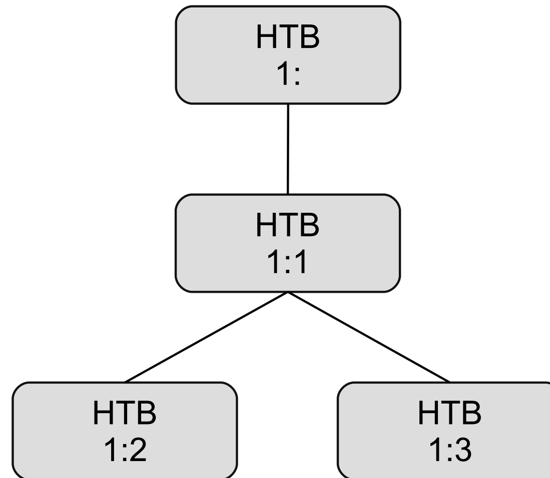


Figure 3.4: HTB hierarchy

Since the HTB qdisc is able to set limits on maximum rates going through it as well as being very accommodating for busty traffic and being a work conserving discipline it is one of the better choices for egress traffic shaping. The ability for sibling and parent classes to share tokens if they are idle is what makes the traffic shaping part of the implementation more refined than usual. If an application is the only one sending data, then it can use as much rate as is allocated, but if multiple applications are sending, then there is a bit of fairness while still preserving the full rate allocation overall.

Each HTB class can have a ceiling rate and a guaranteed rate specified when it is created. The ceiling rate is the maximum possible rate at which traffic can be processed through the class's queue. This is possible when the class borrows tokens from its parent or siblings if sharing is possible with the given parameters. The guaranteed rate is the minimum rate reserved for the class from its parent class. When the ceiling rate and the guaranteed rate are the same value for a class, that class cannot utilize sharing since the guaranteed rate is not allowed to exceed the ceiling rate.

The manner in which HTB qdiscs are used on the UE implementation for egress traffic shaping can be seen in Figure 3.4. The idea of splitting traffic into two classes is that smartphones have limited screen real estate and the focus of applications is usually one active application with a small amount of background applications. In the most extreme case, a user would have three or four applications active where a user may be browsing the web while using streaming audio application or downloading data in the background. The two classes allow enough sharing of resources for applications on the UE to be used adequately depending on their classification.

In the qdisc hierarchy, the root qdisc with the handle 1: contains all of the actual HTB queues that egress traffic will pass through. The first child class with handle 1:1 contains

the maximum rate for all egress traffic on the UE and allows token sharing among its two children classes with handles 1:2 and 1:3. The 1:2 HTB class is the queue that all inelastic traffic will pass through and 1:3 is for all elastic and unclassified traffic.

The main parent class 1:1 has a guaranteed and ceiling rate that matches the rate allocated to the UE globally. This rate is then split among the children 1:2 and 1:3. Both have half of the parent's ceiling rate as their guaranteed rate with the ceiling rate matching the total rate allocation. This split allows both classes will have a minimum rate at which they can send traffic but can borrow tokens from its sibling to reach the maximum rate if needed. During events where elastic and inelastic applications are active, they will have at least some of the UE's total rate to use due to the guaranteed rate for each leaf class.

By default, all traffic flows through the 1:3 queue unless they match filter rules which mark the traffic as coming from an inelastic application. The tc command is used to set filters by handles matching UID for uniqueness. Each filter corresponds to an application UID and the classification given to the application determines which HTB class to which the packets are routed.

Netfilter as shown in [1] is the main method of directing network packets during the process of egress and ingress. Filters are created using the tc command and are paired with iptables rules which apply marks which match the filter for each application; netfilter's marks tag traffic by UID with a handle of the same value.

Since traffic can only flow through 1:2 or 1:3, the 1:1 class controls the egress traffic's maximum rate, which is the rate allocated to it during the shadow pricing algorithm. Having both child queues under the same parent allows sharing from a less active queue to one that needs tokens if it has run out. Having the potential for sharing ensures that rate for a different class is not wasted during idle time if it is not being used.

On the ingress queues, an Intermediate Functional Block (IFB) device is attached to shape incoming traffic. All traffic that is received by the UE is mirrored to the IFB device and its queue before being received and processed by any applications receiving traffic. The IFB device has its own hierarchy of HTB classes set up in exactly the same manner as the egress qdisc. The purpose of IFB mirroring is to use connection based tracking via netfilter marks to have traffic mirrored and shaped on the ingress side exactly as it is done on egress.

Due to limitations in the implementation of the IFB device, netfilter hooks cannot be applied to traffic before it is mirrored to the IFB device and thus, traffic cannot be fully shaped on ingress once it is past the HTB class queues. Some of the effects of this are explained in Chapter 4 and a more in-depth overview of the Android operating system and the necessary modifications are presented in Section 3.4.

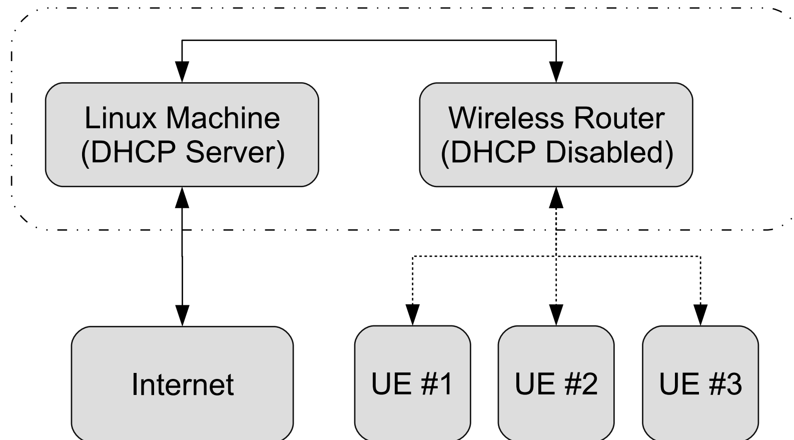


Figure 3.5: Base station layout

3.2 Base Station

The base station uses a computer running a Linux-based operating system to act as the DHCP server in order to be aware of the UEs since the UEs are connected to the router that the computer is connected to. A wireless router was used as the UEs can only connect via Wi-Fi. The router is connected to the computer via Ethernet, with its DHCP and routing functionality disabled effectively turning it into a switch. The computer's main internet connection is then used as the route to the public internet. Figure 3.5 shows the layout of the base station's components.

The DHCP server needs to be aware of the UEs in order to use the tc command, the same command that a UE uses to shape traffic on its own system, on the downlink for each UE. Similar to how the UE uses mirroring with the IFB device on ingress, the base station also uses mirroring to shape traffic coming from the UE. This serves to preserve the maximum rate for uploads from the UE in the case of media streaming coming from the devices themselves. Since the DHCP server controls addressing, the machine can handle rate allocation for the UEs by address. Its functionality allows the base station to emulate how an LTE eNodeB would shape traffic for UEs connected to it.

With the DHCP server being on the same subnet as the UEs and controlling their addressing, the base station can control the downlink rates to each UE by applying queuing disciplines for each address via tc in a similar manner as done on the UEs themselves. A daemon that listens for commands from the resource broker also runs in addition to the DHCP server on the base station. The daemon takes the commands to determine the rate allocated to each UE and apply the tc commands mentioned earlier to represent how the LTE network would limit each phone's download speeds.

3.2.1 Rate Allocation Daemon

The rate allocation daemon is a server process that runs on the base station. It listens for messages from the resource broker to determine shaping parameters for each UE connected to the base station. The daemon's initial state creates an HTB qdisc with a main HTB inner class limited to 10 Mbps on the interface which is connected to the router. Under the main class is a single leaf class where all traffic goes by default which is limited to a ceiling rate of 100 kbps. The default class exists in order to have UEs which have not received rate allocation a reasonable amount of resources to use during the bidding process.

The resource broker sends each UE's information to the rate allocation daemon with two parameters per UE: the UE's address and its rate in kbps. An HTB leaf class is then added to the main HTB inner class for each UE that is given a rate allocation. The guaranteed and ceiling rates assigned match that of the rate allocation exactly to prevent sharing among the HTB classes. This serves to strictly limit a UE's maximum speed to the rate assigned. By having the base station only managing addressing and shaping the downlink to each UE, busier logic is placed away from the base station. An eNodeB would serve the same roles in managing connected UEs and performing the actual rate allocation.

Even though traffic shaping is already present on each UE, shaping is also done on the base station to control the UE's maximum rate and to represent an eNodeB allocating resources per UE. There may also be cases where traffic shaping may not be available on the UE even though it has participated in the bidding process or if shaping is removed, so the act of limiting the UE by its rate allocation can still partially shape traffic. In cases such as these, the shaping by application classes would no longer be present but downlink shaping would still control resources per UE on a different scale. The use of downlink shaping ends up being similar to the ideas presented in [23] but works on a higher level of the network stack.

3.3 Resource Broker

The resource broker follows the algorithm shown in Algorithm 2.2, which is a reproduction of [9, Algorithm 4]. The implementation for the resource broker is done in Python 2.7 and uses pyzmq, a Python version of the ZeroMQ networking library, for handling communication between the UEs and the resource broker, and the GCM service for broadcast push notifications to each UE during the bidding process.

The algorithm for the resource broker's main server mainly follows that as found in Abdelhadi's paper; a slight change exists in that parts of the algorithm are split into multiple states within the server's main loop. This is done to take into account the fact that the resource broker must accept messages from many UEs at certain stages of the algorithm. Since the UE response time to messages from the resource broker is unreliable, the server must wait for all UEs to respond before proceeding with certain segments of the loop.

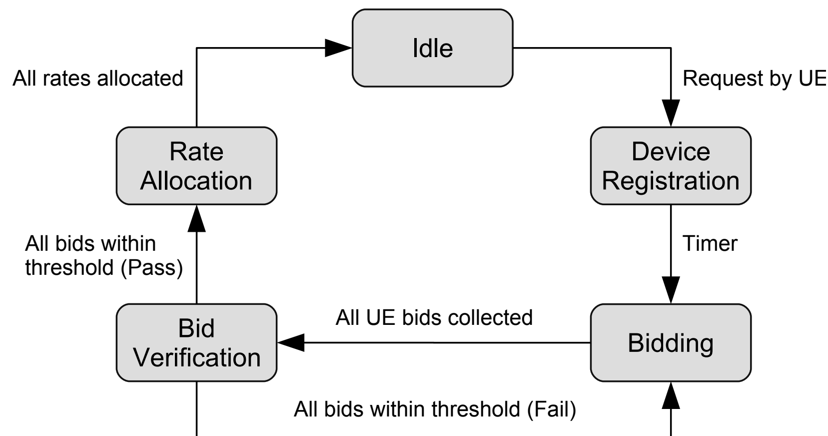


Figure 3.6: Resource broker states

3.3.1 Implementation States

The resource broker listens for messages from all UEs and responds based on the state it is currently in. Figure 3.6 shows a representation of the states in the resource broker's loop. The resource broker starts the shadow pricing algorithm when a request is made by a UE. The rate allocations are calculated at the end of the algorithm and are passed to the base station when the bidding process has completed.

Since the resource broker must be aware of all connected UEs that will participate in the bidding process, there is an additional step where the resource broker broadcasts a request for UEs to register themselves with the resource broker. The resource broker will register each UE along with its GCM ID, for GCM to send push notifications, and IP address, for sending to the base station for traffic shaping. All UEs registered at this stage will be considered in the bidding process since there may be cases where UEs have disconnected from the network since the previous set of bids. Once all UEs connected to the base station are registered, the bidding process begins.

The bidding process is where Algorithm 2.1 and 2.2 occur. Once all participating UEs have sent their bid, the resource broker verifies if each bid passes the condition check in the algorithm. Bidding continues if verification does not pass for any UE. The resource broker enters the rate allocation state if all UEs pass the verification of bids.

Rate allocation is the state when the resource broker calculates each UE's rate to be sent to the base station. Once calculated, the resource broker sends each UE's IP address and rate to the base station's daemon to set up the traffic shaping rules on the UE downlinks. The resource broker also informs each UE of the final price of the algorithm. This allows each UE to calculate their allocated rate for applying traffic shaping rules accordingly.

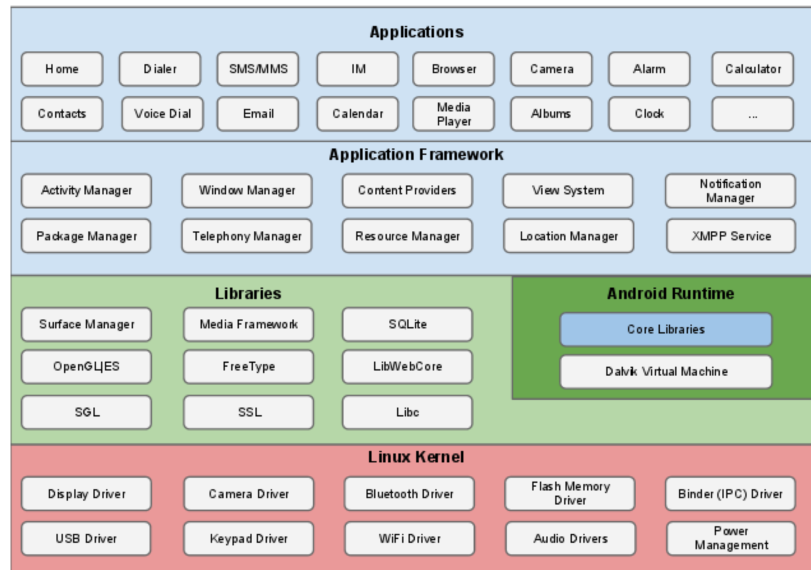


Figure 3.7: Layers of the Android software stack used with permission of [37][1]

3.4 Android System

The Android mobile operating system consists of several layers which separate the operating system's core tasks and capabilities. As a result, modifications to the framework of the Android operating system were required to achieve the necessary functionality to be able to a create traffic shaping system that could communicate across the layers of the system's architecture. Figure 3.7 provides an image from the Android Open Source Project's security overview which depicts the various layers and examples of components that exist within those layers.

The layers exist due to differences in implementation of some of the layers and to serve as separation of privileges as a result of process sandboxing between applications. Visible system layers that the user interacts with are mainly controlled by the Dalvik runtime. The Dalvik runtime utilizes virtual machines to start various system level services as well as applications. The first Dalvik process created early during boot, called the zygote, is responsible for forking virtual machines to start the system server and any user applications when they are opened. The system libraries and native services consist of programs that are usually found in most Linux distributions. However some of these are stripped down due to the different needs on mobile platforms or they are not needed for most common tasks on the operating system.

[1]Portions of this page are reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.

3.4.1 Android System Architecture

At the top of the system is where user-installed applications sit. Each application is sandboxed from the rest as they are run in their own virtual machine processes and is installed as a separate user on the system. Every time an application is started, the zygote creates a virtual machine with a set user ID, which is assigned at install time, to the application's process to isolate it and limit its capabilities.

As standard users, the applications cannot execute commands or actions that require root privileges. Due to this restriction of privileges, modifications were added to the system allow the application to send messages across the layers. Since applications and services at the Dalvik layer have easier access to application data and only the root layer has the capabilities needed to enable traffic shaping, the cross-layer communication changes were necessary to make the end goal easier to achieve. Information about applications, system components, and other information that would be harder for the native service to obtain could be passed down to it to be used as necessary.

System level services and processes are all run as the "system" user with a UID of 1000. System services have access to all permissions available to Android applications including some that are restricted to system applications which have the system's build signature. While system services still do not have root access, there exists an internal communication framework that allows system services and helper applications to communicate with root level processes and daemons at the native layer.

Android's system services are those that manage background functionality for the functions of most of the system's core such as networking, management of installed applications, notifications, and so on. For this implementation, a service called ShapeService was added to receive commands from the user-level application. The ShapeService then passes those commands to a daemon that runs at the native level of the system's architecture between the Dalvik layer and the kernel. The ShapeService is also responsible for monitoring when applications are started or stopped and notifies the daemon accordingly to apply the proper traffic shaping rules.

At the native level, all programs run as the root user with a UID of 0. Most processes and services available are the same as those found on typical Linux operating systems but may or may not be stripped down for the mobile operating system. Using the code from the networking daemon (netd), a modified version of the daemon was used to handle the application traffic shaping rules and filters by adding a controller named ShapeController.

The controller listens for commands from the ShapeService and enables, disables, or modifies traffic shaping rules as needed. These rules are a combination of applying queuing disciplines with the proper filters and classes to allow outgoing and, to a limited extent, ingress traffic to be shaped properly. Firewall rules applied through iptables by the daemon are also a component used in categorizing traffic.

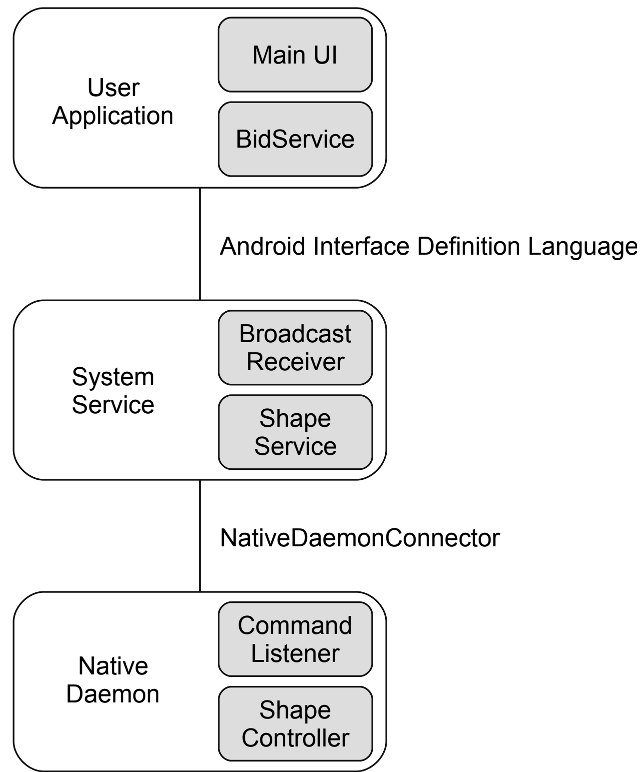


Figure 3.8: Traffic shaping communication

To utilize a tiered approach to passing messages between the layers, interfaces for cross-layer communication were utilized from the user-level application's service all the way down to the controller added to netd. Splitting the functionality across layers best utilizes the different strengths of each layer in the Android stack. It also provides options for future work to utilize certain aspects of this implementation by modifying as many parts of Android as needed.

3.4.2 User Application

The application developed for this traffic shaping system, named Broker Agent, sits at this layer although with a few differences from standard applications that would be installed from elsewhere. On devices where root can be passed to applications, escalation must be requested and privileged commands can be executed if access is granted by the user. The application is developed to not rely on those requests to make the process more automatic and less intrusive. Instead, the application communicates to the ShapeService added to the system server and access to ShapeService is limited to the application as it is created as a system application.

The application runs a background process called an IntentService, which is a service that

runs in the background when invoked and then destroys itself when finished. The service registers the phone with the resource broker and participates in the shadow pricing algorithm to determine its rate allocation. The service is where [9, Algorithm 3] is implemented. Each UE connected to the base station can have its service triggered by a push notification sent by the resource broker using GCM. Registration into the resource broker's device list is taken care of by the resource broker itself and devices are uniquely identified using the ID assigned by GCM's servers. The service also uses a Java build of ZeroMQ to connect to the resource broker to directly send bids during the biding process.

There are also various utilities present on the Android system that are used to store data from the shadow pricing algorithm. The most recent bid, price, and rate are stored in the application's private storage for primitive data called SharedPreferences as a means of history tracking. The application also has utilities to measure data usage per application, although this functionality and the history tracking were not expanded upon for the end result of the implementation.

Since the application is installed as a system application with the platform build key for the modified version of Android in this implementation, it can access the modified interface calls to enable and disable traffic shaping. Android has an interface named the Android Interface Definition Language (AIDL) that allows applications to communicate with other services that follow the same definitions, enabling simple inter-process communication between applications and services which may not belong to the same owner.

Using the AIDL to pass parameters to the system service allows the lower level control of the traffic shaping structures to the root level daemon and does not require the application itself to request root privileges. Having the application remain in the user-space can allow it to be present for the user to view information, if needed, without giving the application too much control. For the purposes of this implementation, the user interface was only developed as a debugging tool.

The system service modification to be explained in the next subsection details how the AIDL defined for the service can be used by the application to pass the necessary information down to the root level to enable traffic shaping. As the application is built as a system application for the modified operating system in this implementation, it is able to recognize the added system service. This allows the application to call methods defined in the AIDL to remotely use the system service's functionality.

3.4.3 System Service

As mentioned, the user-level application developed for the traffic shaping implementation sits in the layer where all other standard Android applications exist. One property that makes it differ from most applications is that it is installed as a packaged system specific application. This prevents it from being uninstalled and allows it to access APIs that are

only available to packaged system applications such as the ones added to the system service.

The system server is one of the initial processes forked from the zygote and is a key process in the Android system's framework. It is special in that it runs as the system user with UID 1000 and is a persistent process that runs while the operating system is active. The traffic shaping service is a secondary service that is run through the system server under the name ShapeService. The service is added as a separate component that is launched by the system service on boot. As a system service, it can be accessed via the service manager or direct service binding and also has methods which can be accessed through the AIDL defined for it.

ShapeService mainly acts as a middle man to pass messages down to the root daemon from the application level. Because of its positioning in the architecture and its status as a system user, ShapeService has the most capability to act on system events at this layer. Aside from passing the user-level application's messages to the ShapeController on netd, ShapeService also notifies the netd of application states to shape applications by the correct classification.

Receivers were added to allow ShapeService to detect when applications are started or stopped on the system. The messages sent to the daemon direct it to add or remove filter rules to match traffic by UID. The filters are used to send matching traffic to the correct HTB class and are added whenever an application is started or deleted when an application is stopped.

The various framework modifications were done following the same steps as detailed in [38]. These steps allowed ShapeService to be added and run independently of other system services and its AIDL to be used by applications and services which are also created under the same Android build. Since the main programming language for processes used is Java, the AIDL allows data to be serialized as Java primitives through remote method calls. Any application or service which needs to call a remote service can bind to the service and then call the method in the same manner as any other method.

ShapeService communicates with the root level daemon through a different type of internal connection as the system service is run in a Dalvik virtual machine and netd is a native service running directly on the Android operating system. Whereas the user application binds to the ShapeService to invoke methods remotely, ShapeService communicates with netd using the NativeDaemonConnector which works like a regular network socket.

3.4.4 Native Root Level Daemon

Since netd was borrowed for use for applying the traffic shaping rules, the method for invoking netd remotely was also borrowed from other portions of the Android code. The NativeDaemonConnector is a remote interfacing class which serves the same purpose as the AIDL but between Dalvik based services and native services. The manner in which it works is similar to that of a network socket where the server is the daemon and the system service

is the client with messages of functions to be executed being written into the socket.

Netd uses a class called the CommandListener to process commands sent to it through means such as the NativeDaemonConnector. Extra commands were added to the CommandListener to allow netd to invoke ShapeController's functionality when ShapeService sends commands through messages to netd. The commands are all mapped to invoke certain methods within ShapeController to enable, disable, or modify traffic shaping rules.

When the ShapeController is instructed to enable shaping, it sets up all of the qdiscs and subclasses in the manner as shown in Section 3.1.2. The egress queue will have traffic going through one of two HTB classes depending on if the traffic comes from an application classified as elastic or inelastic; by default all traffic flows through the elastic queue. Ingress traffic will all get mirrored to an IFB device with an HTB qdisc attached.

Commands to mark applications are initially sent by the user application to the daemon after every instance of the bidding process being completed. Each application has a filter set for it uniquely identified by the application's UID with the filter directing it to the 1:2 HTB class if it is inelastic or the 1:3 HTB class if it is elastic. The corresponding iptables rules to mark packets to work with the filter are also applied at the same time.

When an application is stopped, a message comes from the ShapeService to have its filter is removed. If an application has been classified and ShapeController has been informed, it will have its filter re-applied when it starts again. Just as with setting filters, removing filters also removes any iptables marking rules until the application is started again. Any unclassified applications will have their traffic sent to the elastic queue by default even if no rules are applied as long as the queues are active.

The communication flow presented in this chapter is how key system applications and services in Android also function to pass data and messages to other parts of the operating system. By extending existing services to add the traffic shaping functionality, the set of applications and services added can be added as an extension to any version of Android if desired. Since the extensions are added to follow existing frameworks, practical implementations could use the implementation provided as a base or modify it further without impacting the original system's services.

Chapter 4

Methodology and Results

4.1 Equipment

The key components that participate in the system can be considered in three main groups: the resource broker, the base station, and the UEs. Figure 4.1 provides a visual representation of the groups and how each component is connected in relation to the others. The resource broker also has its own connection to the internet in order to use GCM; this is only necessary for this implementation, an implementation in LTE may use a different method for pushing messages to UEs.

The resource broker runs on a Linux machine and needs public internet access in order to use GCM as GCM servers require a whitelisted IP address. The machine running the resource broker is also connected to the same network as the base station through an Ethernet connection to the base station's router. Although the push notifications using GCM requires communication to GCM servers to handle the messages, the bidding process involves messages directly communicated between UE clients and the resource broker's server socket using ZeroMQ sockets.

The base station, as covered in Section 3.2, consists of the wireless router acting as the switch to connect all devices on the private network for the system and the Linux machine representing what an eNodeB would do with connected devices. All routing is done by the Linux machine as it is the DHCP server and default gateway and the router acts as the switch to create routes for the entire network. All UEs connect to the router via Wi-Fi but are assigned addresses by the Linux machine. Through NAT, the UEs have a connection to the public internet provided by the Linux machine's alternate network interface.

The UEs are Samsung Nexus S phones with unlocked bootloaders. Each UE has the custom Android build flashed as its operating system with the custom kernel modified to support various traffic shaping and networking modules. The operating system itself visually appears

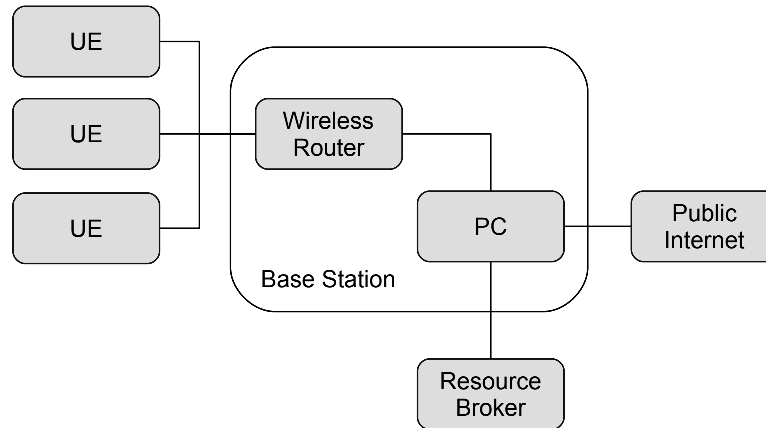


Figure 4.1: System layout

to be the same as other versions of the stock Android operating system with the exception of the user application, Broker Agent, which was added for the traffic shaping process. All other functionality is transparent to the user and would not be noticed through ordinary usage. The UEs do not have mobile data enabled, since they are only being tested using the Wi-Fi interface which acts as a representation of a mobile network.

4.2 Testing and Verification

The two main parts of the system that must be active before anything begins are the base station and the resource broker. Both components must be set up and connected to each other before any of the bidding process and rate allocation steps can occur. Once they are set up, UEs can connect to the base station and the process can begin once there is more than one UE on the base station's network.

The base station's private network is set up to cover the private range 10.0.0.0/24 and assigns addresses via DHCP from 10.0.0.102 and up. The base station itself reserves the address of 10.0.0.100 for the Linux machine and 10.0.0.101 for the wireless router; the resource broker is accessible via address 10.0.0.120. When the base station's daemon is started, it sets up default preliminary qdiscs on the downlink with the maximum rate set to 10 Mbps due to the limitations of the hardware used and amount of UEs for testing. A default queue is also set to limit any UEs which are not properly shaped to have 100 Kbps on their downlink. This is to prevent UEs which have joined the network but have not participated in the bidding process from hoarding bandwidth from other devices on the network while still allowing them enough bandwidth to participate in future bidding events.

The resource broker starts up and connects to the base station. Once connected, it notifies the traffic shaping daemon on the base station to reset traffic shaping to the default rules mentioned previously. The resource broker begins in the idle state after all initialization is finished in order to start listening for messages from UEs on the network.

The user application contains code which allows its background service to send a bid request to start the bidding process at set intervals to make sure rate allocations are up to date. For this implementation, a button on the application UI was purposed to manually trigger events such as registration and requests to start the bidding process as well during testing.

When the bidding process first starts, the resource broker sends a GCM push notification requesting all UEs that have registered themselves with GCM's servers to also register with the resource broker to keep track of which devices are active on the network. Since the application first runs when the system is booted, all UEs on the network will have registered with GCM provided they can access the internet. Once all active phones are registered with GCM and the resource broker, Algorithm 2.1 and 2.2 take place.

The end of the resource broker's cycle involves sending the resulting shadow price to all UEs and the resulting rate allocations for each UE to the base station. The base station allocates each rate on the downlink assigned to each UE by its IP address on the network. At the same time, the application on each UE is propagating information for rate allocation information and application classification down to the native service to create the queues and appropriate filter rules for each application.

This process continues for as long as there are UEs on the network to request the shadow pricing algorithm to be started. On the devices themselves, information on the traffic shaping structure can be viewed using the tc command; tc can be used to view but not modify the queues unless if root access is obtained in a terminal on the UE itself. The UEs tested in the implementation have root access available since the operating system was built as a debug and engineering build. The Android debug bridge (adb) allows access to the device's terminal to view information via the logcat utility. Using adb as the root user allows the use of utilities such as tc and iptables which can be used to verify the traffic shaping implementation.

The rates determined for each UE should weight more towards devices that are using inelastic applications. Likewise, UEs which only have elastic applications active receive a lower rate allocation based on the differences in the utility functions. In the case when all three devices have the same active applications, such as after device boot, only the default applications are active on the device. In a case such as these, the expectation is that the rate allocations for all devices are the same, since their utility functions are the same. Table 4.1 shows this for the results of the bidding process where all UEs are using only elastic applications.

A test involving all three UEs participating in the bidding process after boot completion is shown in Figure 4.3. On the 10 Mbps link provided by the base station, each UE receives a rate allocation of 3.333 Mbps after the completion of the bidding process. To test the rate ceiling set for each UE, the speedtest application was used to measure download and upload

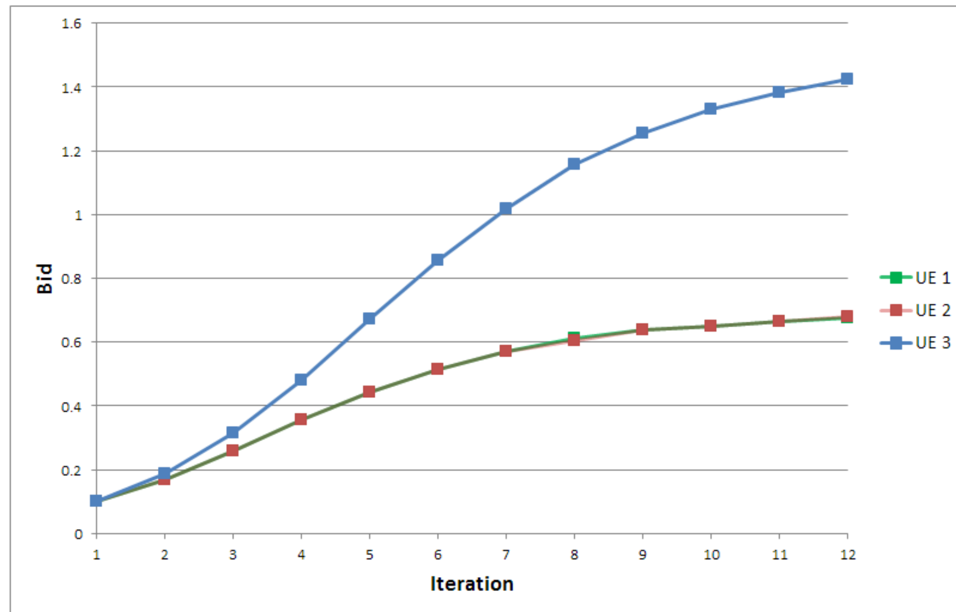


Figure 4.2: Bids by each UE with 1 inelastic/elastic UE and 2 only-elastic UEs

speeds on the device. The test downloads test files and then uploads random data to a test server in order to determine the device's network speeds and ping. The first test shows each UE limited to about the same rate when their speeds were measured. In the figure, this can be seen as the maximum upload rates for each UE are within in the 3 Mbps range.

As mentioned, a UE which has an inelastic application will cause the inelastic utility function to have a greater influence on the UE's bids since it needs a higher target rate allocation to meet its needs. In a test following the initial test, a UE has an inelastic application started and running on the system. The next series of bids results in that UE receiving a higher rate allocation. Figure 4.2 shows the bids for all three UEs over a series of iterations during the bidding process. Using the same test with the speedtest application, the results of the UEs bitrates during the test can be seen in Figure 4.4. A small summary of the rate allocations for the UEs in this test can be seen in Table 4.2. Comparing the table and the figure shows the inelastic application with its 5.1 Mbps allocation to be able to upload at a higher rate than the other two UEs which are limited to their allocation of 2.4 Mbps.

Using tc, the entire hierarchy can be viewed using the list commands for both qdiscs and classes which reveals the queues in a hierarchy as mentioned in Section 3.1.2. The listing also provides details into the work serviced by the queues such as packets processed and tokens borrowed; a sample listing is shown in Figure 4.5. This particular listing shows the root qdiscs on all of the UE's network interfaces. The HTB qdiscs can be seen on its main interface wlan0 while the ingress device is mirrored to redirect ingress traffic to ifb0 which also has an HTB qdisc as its main queue. The tc utility can also list information for every class and filter that has been applied by it as well.

The iptables utility provides statistics for packets that pass through iptables's chains. Information for the amount of packets that match the application specific rules set up by the ShapeController can be cross-referenced with the qdisc listing to see traffic being directed by the netfilter hooks.

On the base station, the same commands can be applied to view the HTB qdisc and class listing. Each class in the main HTB qdisc on the network interface going back to the UE should represent a UE's allocated rate, with the maximum rate being the total rate allocated for the device. The same structures can also be seen on the IFB devices with an HTB class representing each UE to shape upload speeds as well.

In the testing of the implementation, three UEs were used with varying applications to show the effect of a UE with inelastic applications receiving a higher rate allocation. An example of one test used had one UE set up to have an active inelastic application and the remaining two UEs with only elastic applications running. Each phone also contained the stock applications that are built into the system by default which are all directed to the default queue as they are not taken into consideration in the utility calculation. None of the default system applications are applications which have a high rate requirement, thus the default queue should be adequate for their functionality.

Table 4.1: Network rate allocations for all elastic UEs

ID	Types of Applications	Shadow Price	Bid	Rate (Mbps)
1	Only elastic	0.133726	0.445753	3.333
2	Only elastic	0.133726	0.445753	3.333
3	Only elastic	0.133726	0.445753	3.333

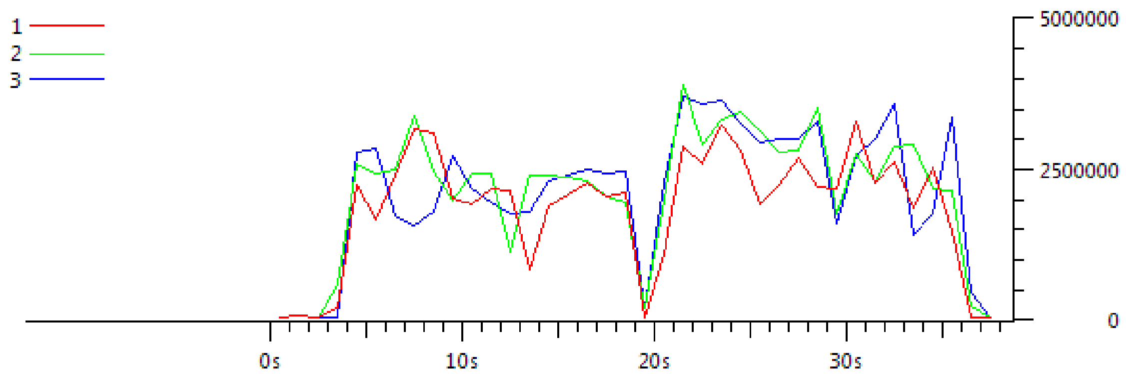


Figure 4.3: UE bitrates with all even allocations

Table 4.2: Network rate allocations for 1 inelastic UE and 2 elastic UEs

ID	Types of Applications	Shadow Price	Bid	Rate (Mbps)
1	Only elastic	0.277876	0.678293	2.441
2	Only elastic	0.277876	0.677213	2.437
3	Inelastic/elastic	0.277876	1.423249	5.122

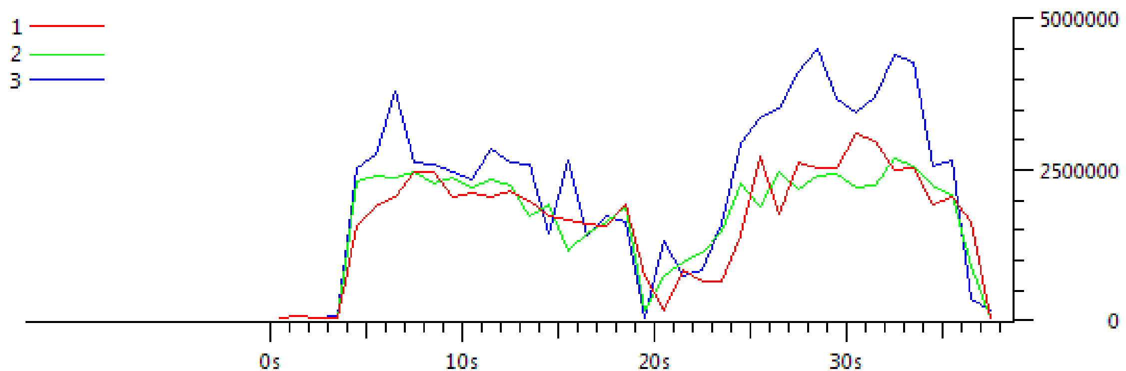


Figure 4.4: UE bitrates with an inelastic UE receiving a higher rate allocation

```
qdisc htb 1: dev ifb0 root refcnt 2 r2q 10 default 3 direct_packets_stat 0 ver 3.17
 Sent 4654041 bytes 5018 pkt (dropped 70, overlimits 5178 requeues 0)
 backlog 0b 0p requeues 0
qdisc htb 1: dev wlan0 root refcnt 2 r2q 10 default 3 direct_packets_stat 0 ver 3.17
 Sent 4719973 bytes 5813 pkt (dropped 0, overlimits 7032 requeues 0)
 backlog 0b 0p requeues 0
qdisc ingress ffff: dev wlan0 parent ffff:fff1 ----------------
 Sent 4423453 bytes 5113 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Figure 4.5: Sample tc qdisc listing from a UE

```
class htb 1:1 root rate 5119Kbit ceil 5119Kbit burst 8b/8 mpu 0b overhead 0b
        cburst 8b/8 mpu 0b overhead 0b level 7
 Sent 4721905 bytes 5815 pkt (dropped 0, overlimits 0 requeues 0)
 rate 1804Kbit 202pps backlog 0b 0p requeues 0
 lended: 1777 borrowed: 0 giants: 0
 tokens: -34922 ctokens: -34922

class htb 1:2 parent 1:1 prio 0 quantum 31987 rate 2559Kbit ceil 5119Kbit
        burst 9b/8 mpu 0b overhead 0b cburst 8b/8 mpu 0b overhead 0b level 0
 Sent 132 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 2 borrowed: 0 giants: 0
 tokens: -3031 ctokens: -1516

class htb 1:3 parent 1:1 prio 0 quantum 32000 rate 2560Kbit ceil 5119Kbit
        burst 9b/8 mpu 0b overhead 0b cburst 8b/8 mpu 0b overhead 0b level 0
 Sent 4721773 bytes 5813 pkt (dropped 0, overlimits 0 requeues 0)
 rate 1801Kbit 202pps backlog 0b 0p requeues 0
 lended: 4036 borrowed: 1777 giants: 0
 tokens: -69828 ctokens: -34922
```

Figure 4.6: Sample tc qdisc class listing from a UE

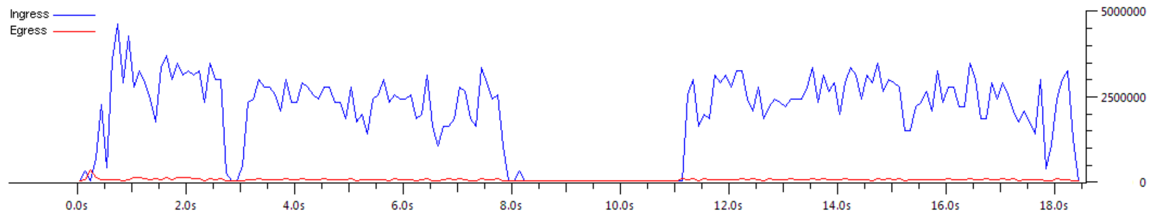


Figure 4.7: UE bitrate during streaming audio measured locally

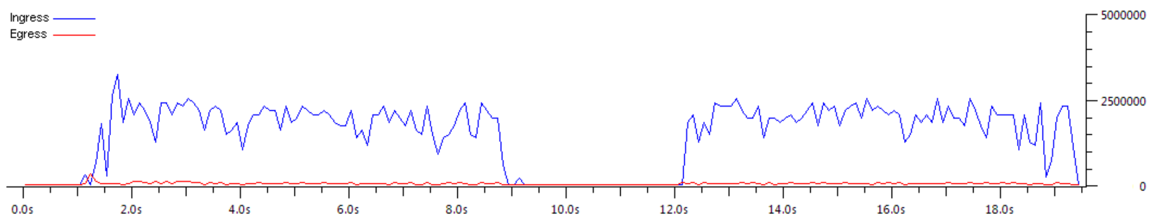


Figure 4.8: UE bitrate during streaming audio as measured by base station

4.3 Analysis

As mentioned in Section 3.1.2, the queues are set up with the root HTB qdisc holding an inner HTB class limiting the maximum rate and allowing token sharing amongst two sibling queues as the main inner class's children. The tc command can be used to view this information in the same manner as for the qdiscs on all of the network interfaces. Figure 4.6 shows the HTB class listings on the UE's main network interface. As can be seen by the handles next to the HTB classname, the handles are assigned in the same order as seen previously in Figure 3.4.

While the implementation is set up to handle shaping on both ends to preserve the rate allocation on an entire link, only the egress shaping works as intended to throttle the rate at which packets pass through the UE's network queues. Shaping on the downlink from the base station to the UE works to control the maximum rate at which traffic can be sent to the UE. This works as shaping is done as egress from the base station to the UE where traffic must pass through specific HTB classes per UE, so the rate allocation is met via traffic shaping from that direction.

The combination of the downlink shaping as well as the ingress HTB shaping via ifb0 control the rate at which the UE receives data. Figure 4.7 shows a graph of data rates in bits per second using analysis tools in the program Wireshark; in this figure, the download bitrate is shown in blue and the upload bitrate is shown in red. The data was captured using the tcpdump utility while the UE was actively streaming data from an audio streaming application. This capture took place after the bidding process had completed with three UEs participating in the network.

Table 4.3: Network rate allocations with 1 strict elastic UE and 2 flexible inelastic UEs

ID	Types of Applications	Shadow Price	Bid	Rate (Mbps)
1	Using video application	3.531048	8.885787	2.516
2	Using audio application	3.531048	13.212345	3.742
3	Using audio application	3.531048	13.212345	3.742

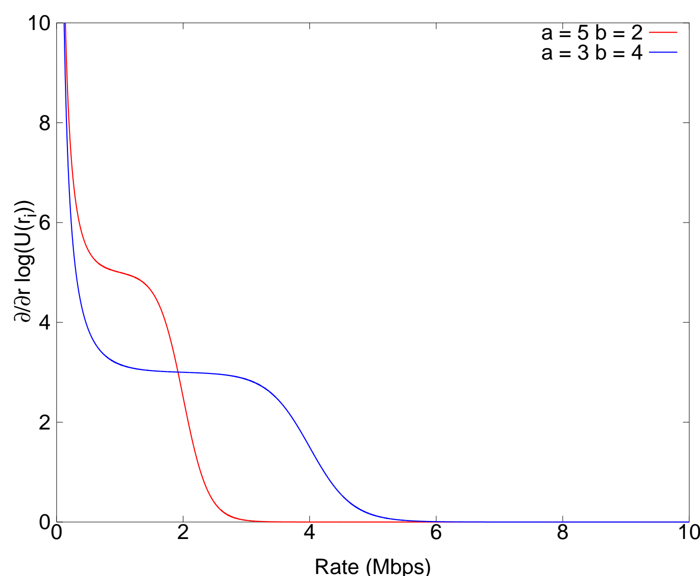


Figure 4.9: Partial derivative of inelastic utility function with varying parameters

Both captures were run simultaneously with a difference in starting timestamp of less than one second. Comparing Figures 4.7 and 4.8 show similar bitrates over time but with a slower bitrate measured on the base station. Since the rate allocated to the UE is also applied as egress shaping on the base station, the UE's ingress rate is limited regardless if the UE has its rules applied; the UE's traffic shaping rules serve to refine the traffic through separation of inelastic and elastic applications into different queues.

The rate allocation for the test where one UE has an active strict inelastic application and two UEs have flexible inelastic application active can be seen in Table 4.3. The strict inelastic application is a video application while the flexible inelastic application is an audio streaming application. As mentioned in Section 2.4, the values used for the parameters which influence the inelastic utility function are $a_i = 5, b_i = 2$ for flexible inelastic applications and $a_i = 3, b_i = 4$ for strict inelastic applications. The bid by the UE with the video application fits within the main area before the inflection point, as seen in Figure 4.9, since as the price increases, the amount of utility received is less. The parameter which controls the step-

function quality of the function is also lower which gives a higher room for possible rates. This seems to influence the remaining rate being allocated to the rest of the UEs which may be higher despite the lower utility.

The limits of the allocation can be measured in various ways. The UE with the video application can be seen with a ceiling rate of 2.516 Mbps. The guaranteed rate for both children of the inner class is around 1.258 Mbps with a ceiling that matches the inner class's ceiling. The ceiling limit can be seen near the 2.5 Mbps mark in Figure 4.7 where the UE cannot exceed its allocated rate due to shaping on both ingress and egress. Towards the beginning of the stream, the rate contains spikes which, for a brief moment, exceed the ceiling. This may be attributed to both the burst allowed by HTB when traffic picks up on the scheduler as well as TCP congestion control causing the UE to slow its transmission since egress traffic is being delayed on the UE through the schedulers.

A spike can be seen when the initial burst of data is sent which shows the highest bitrate that the stream achieves during this session. This results in a faster initial reception of data in the download stream which does exceed the 2.516 Mbps available to the UE due to the burst available from HTB. The rest of the stream stays within the bounds set by the shapers.

The UE's ingress bitrate does exceed the guaranteed rate from the default HTB class on the ifb0 device since netfilter hooks cannot be applied to packets before they are redirected to IFB devices as mentioned in [39]. This results in one queue that is never used except for sharing since the classes on ifb0 are created to match the egress classes. Since traffic is directed to the HTB class for elastic applications by default on ingress, ingress can only arrive as fast as the class allows and as fast as tokens are able to be lent.

In most cases, the upload speed will be low and the majority of traffic shaping effects will be through the HTB classes on the ifb0 device. However, there are cases where the upload bandwidth may be heavily used, such as live communication using video and audio applications. A third party application which measures network speed was used to test the limits of both the UE's download and upload speeds. The application downloads and uploads test files to measure the speeds and report them to the user.

The application was run in the same instance where the UE with an active inelastic application was allocated a total rate of 2.516 Mbps. This rate was split into inelastic and elastic HTB classes with a guaranteed rate of 1.258 Mbps and a ceiling of 2.516 Mbps. When the application completed its test, the rates it reported for the UE's maximum download and upload were within the range allowed by the traffic shaping schedulers. Figure 4.10 provides a screenshot of the application's UI with the result. The 2.53 Mbps reported for the maximum upload is within the range allowed by the ceiling of the HTB classes. The reported download speed is within the allowed global ceiling rate but may be lower partially due to throttling from lack of tokens or due to the quality of the base station's link.

Captures were also done on the UE during the application's test as well, the resulting rates, shown by sent packets in red and received packets in blue, are shown in Figure 4.11, which

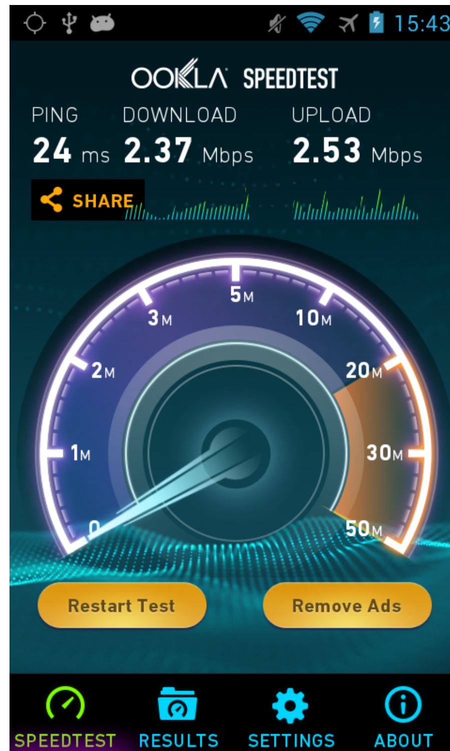


Figure 4.10: Screenshot of the Speedtest application on the UE

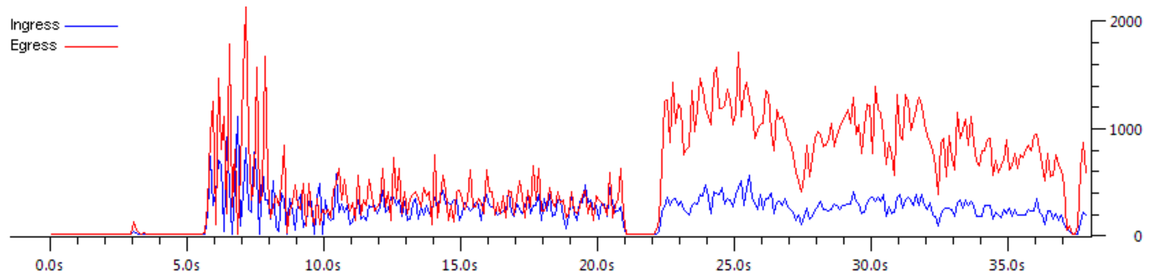


Figure 4.11: Packet transmit and receive rates during speed test without traffic shaping

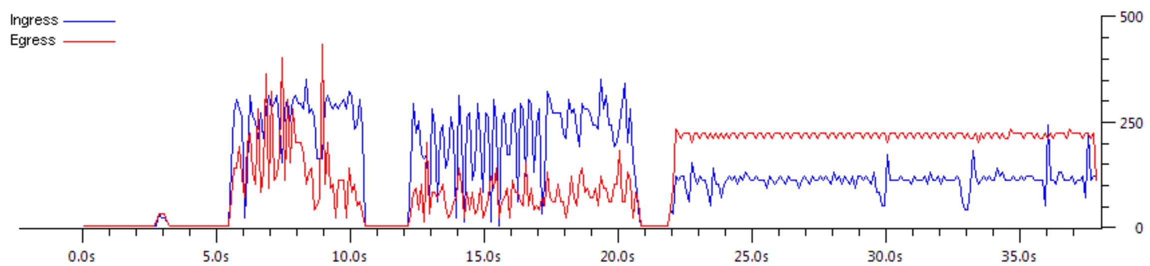


Figure 4.12: Packet transmit and receive rates during speed test with traffic shaping

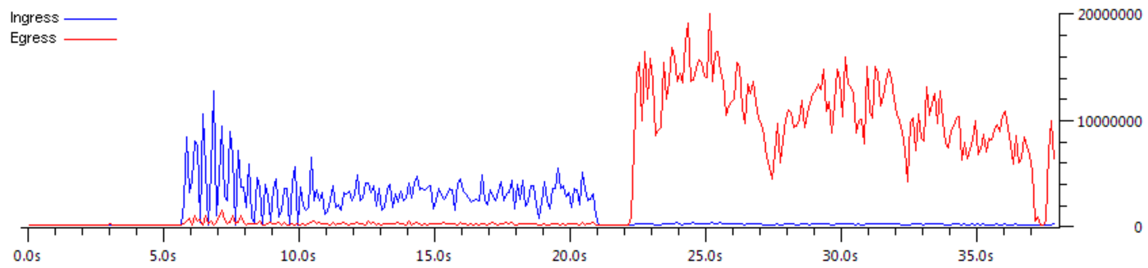


Figure 4.13: UE maximum bitrates with no traffic shaping

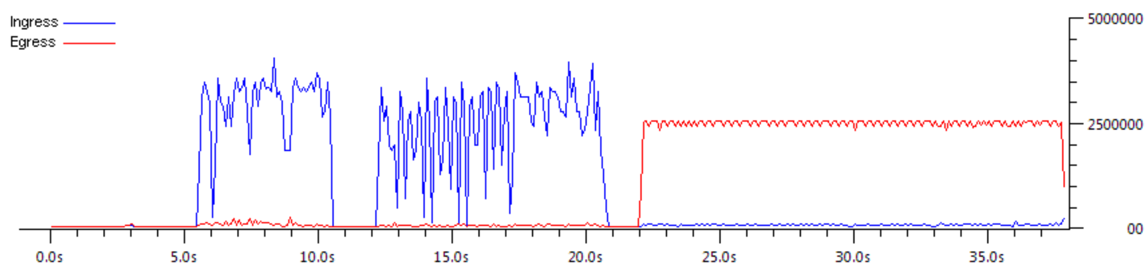


Figure 4.14: UE maximum bitrates with traffic shaping

shows the packet transfer rates for the application when there is no traffic shaping, and Figure 4.12, which shows it when traffic shaping is active.

For the case with traffic shaping, the initial burst given for traffic allows data to be received with a reasonable initial buffer before being slowed down to the rates allowed by the schedulers either by TCP congestion control as mentioned previously or by HTB delays from lack of tokens. For the case with inelastic applications which deal with streaming media, the data is usually buffered before playback begins to reduce the occurrences of pauses and interruptions in playback. The initial speed burst would then facilitate in data buffering while preserving an acceptable quality of service for the user.

In the case without traffic shaping, much higher amounts of data is consumed in both directions of data transfer. On a dense network with many other users connected to the same node, UEs which are heavily utilizing the network might negatively impact service from other UEs which are requesting less data. As mentioned for the utility functions, once an application has access to a certain level of bandwidth, any more bandwidth given to it will lead to diminishing returns with regard to utility.

The aim of maximizing utility and rate allocation is better management of resources. In a practical implementation, an eNodeB that may be saturated with users can better serve its existing clients as well as having resources available for additional clients if necessary. Figure 4.13 and 4.14 represent the same data as previously but with total bitrates instead of packet transmission rates.

Table 4.4: Network rate allocations with 2 inelastic UEs and 1 elastic UE

ID	Types of Applications	Shadow Price	Bid	Rate (Mbps)
1	Inelastic/elastic	1.181322	5.325217	4.508
2	Only elastic	1.181322	1.162789	0.984
3	Inelastic/elastic	1.181322	5.325217	4.508

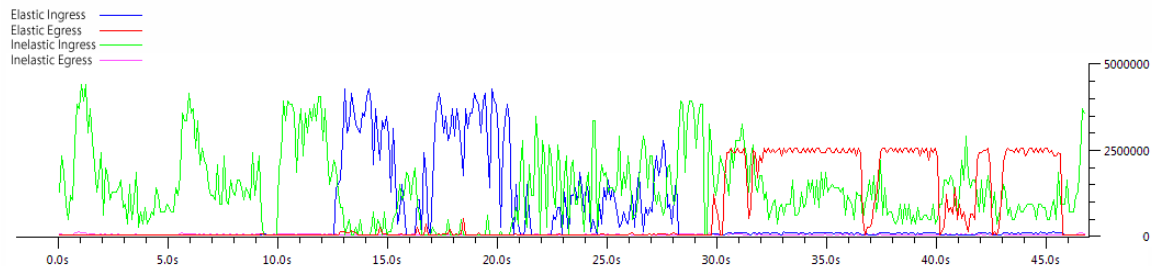


Figure 4.15: UE bitrate with applications using both inelastic and elastic HTB classes

Amongst the UEs, any one UE would not need to use 10 Mbps or more of a network link to upload data as shown in Figure 4.13. A process such as a file upload may be run in the background where the user may not need it to complete very quickly as the application is not in focus. In this case, a controlled rate, such as the 2.5 Mbps shown in Figure 4.14, would still be adequate to achieve the task and be less taxing on the network.

The use of traffic shaping on UEs also better utilizes bandwidth among applications on the devices themselves. Since the allocated rate is split between HTB classes for inelastic and elastic applications, applications of different types can easily share network resources locally on the UE. In a test where two UEs were using an inelastic application and one UE only had elastic applications, the rate allocations were determined as shown in Table 4.4. Figure 4.15 shows an example of aggressive usage of network resources by both an elastic and inelastic application at the same time on one of the UEs using the inelastic application.

On this UE, the rate allocated to it is higher since it had applications which were classified as inelastic active. Its ceiling rate was 4.507 Mbps with each leaf class having half of the ceiling rate as its guaranteed rate. The elastic application used is the same application used for testing network speed while the inelastic application is the audio streaming application in the example preceding the network test application.

The download rate reaches the maximum rate around the allocated 4.507 Mbps and also exceeds it at points but also slows down intermittently similar to captures from previous cases due to bursting and rate control. The inelastic application downloads media in segments to buffer data during playback which causes the inelastic rate to drop occasionally. The speedtest application is the elastic application which maximizes the download rate when the

inelastic application is idle after buffering; its rate is maximized due to HTB sharing during the inelastic application's idle time.

When both applications are downloading, the rates appear to match the ceiling for each HTB class which is half of the total 4.507 Mbps mark around the 20 to 30 second timeframe. During the timeframe in which the speedtest application is testing the upload speed, the upload increases and remains between 2 and 2.5 Mbps which means the application is receiving at least or near the guaranteed rate for the elastic HTB class on its upload as well. The figure shows that the ceilings reached are reasonable and match the implementation's application of queuing disciplines on the device.

Each UE behaves as intended individually. The rate allocation decided by the shadow pricing algorithm is enforced at the lowest level of the operating system and HTB classes for inelastic and elastic applications rate limit traffic appropriately. The HTB classes also share tokens when available. All of these actions also take place transparently to the end user of the device and there is no effect on the user's experience unless if the device has not received a rate allocation and only has the default rate provided.

4.4 Evaluation

For the system presented, the traffic shaping system works to have rates assigned to UEs and the rates are assigned on both ends of the network links used by the UEs. The base station shapes UEs as a whole group and UEs shape applications by their groups as well. The possible measurement of application utility is easily determined by the parameters set for the utility functions. The measurement of utility among a user of a UE is more subjective and could be something that is expanded upon in future work in conjunction with tests on networks of varying size and capacity.

Based on the rate allocations determined from various tests, the rate allocations fit within expectations for the network presented in the implementation. UEs with inelastic applications receive larger allocations compared to those that are not using them. UEs with varying types of inelastic applications receive differing amounts based on their classification as flexible and strict inelastic applications. The automated traffic shaping framework on the UE works to conform to the rate allocations decided by the resource broker and without requiring interaction by the user.

The design of sorting application traffic into two classes works to share the UE's internal queues between different applications while guaranteeing a specific rate to each class. This works for current operating systems as the number of applications a user has active on software architecture for current smartphone operating systems will be limited. Generally there will be one user facing application with background processes that may run on behalf of other applications. As such, the use of general classifications of elastic and inelastic applications is adequate in the current implementation presented. If this paradigm were

Table 4.5: Network rate allocations for 2 inelastic UEs and 1 elastic UE

ID	Types of Applications	Shadow Price	Bid	Rate (Mbps)
1	Inelastic/elastic	1.181322	5.325217	4.508
2	Inelastic/elastic	1.181322	5.325217	4.508
3	Only elastic	1.181322	1.162789	0.984

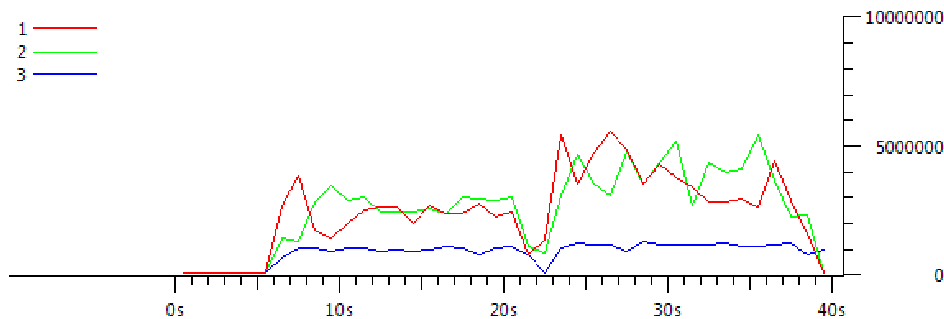


Figure 4.16: Bitrate for three UEs using the base station's 10 Mbps rate allocation

to change in the future, the system may require more refined application classification and shaping rules.

For the whole of the network, the rate allocations are assigned to maximize utility for the base station's resources. In a test with three UEs where two UEs were using inelastic applications and one was only using elastic applications, rate allocations were calculated and assigned as shown in Table 4.5. Using the speedtest application again, but on all three UEs, tcpdump captures were run on the base station to record the bitrates for each device. The resulting bitrates during the capture can be seen in Figure 4.16.

For all tests, the maximum rate set for the network was 10 Mbps. The algorithm distributes all of the 10 Mbps among the UEs when the bidding process is complete. In the case with all UEs running speed tests to show utilization, the maximum upload rate shown for all UEs matches the maximum rate allocated for each UE. The rates allocated also match the expectation that users which use more inelastic applications have a higher rate requirement. Their higher utility needs result in higher bids which ends up with a higher allocation in relation to other users.

A big dependency for this system is the responsiveness of messages between the UEs and the resource broker. In an ideal environment, the calculations and messages would be produced and processed with no delay. In reality however, there will be delays which would slow down the bidding process. There could also be factors such as UEs dropping out due to connection changes or cell handoffs between nodes.

The demonstration provided for this implementation was done on a small Wi-Fi network but shows that the algorithm and logic from [9] could be applied to larger networks. Some methods to expand upon the implementation to account for scale may be a topic for future work. During the tests to set up the network, each component in the network kept logs to record events in the bidding process and rate allocation.

A few relevant details that were logged include: number of iterations per bidding process, the shadow price at each iteration (including the final price), and allocated rates. In the case with the three UEs where one contained an active inelastic application and the other two had no active inelastic applications, the bidding process took 11 iterations to reach a final price using a bid threshold of 0.05. This means each UE's bid had to be within 0.05 of its previous bid in order for the bidding process to end so that the resource broker can trigger the rate allocation steps.

In a smaller network such as the one tested in Chapter 4, the delays are small and the number of iterations is lower in comparison to one with a higher bandwidth as simulated in [9]. With UEs that bid higher due to applications requiring a higher rate to achieve a desired utility, the number of bids would usually be higher. And with a network where communication delays would be involved, the bidding process would take longer for nodes which cover more area and more UEs. In networks such as these, flexibility in the network would be required to optimize time and resources for the traffic shaping system.

A side effect of testing in a smaller network shows that if the base station has enough bandwidth to support all of the devices, then the rates allocated might exceed what is needed for all applications. This occurred when in tests where each UE had no inelastic applications active resulting in all UEs being allocated the same rate during a bidding process that ended in a few iterations. As such, the system would be a best fit for areas where, for example, an LTE network has an eNodeB with many UEs connected to it and the total rate must be managed tightly. This could also be a focus of future work where many clients are connected to nodes with very high or very low bandwidth to check for flexibility in handling rate allocations for UEs. Even with these observations however, the implementation works to present a framework on which the system works to do rate allocation and traffic shaping.

On the UEs themselves, the communication pipeline used by modifying the Android framework works well for implementing a set of services that are transparent to the user. The communication layers work as long as APIs are hidden to make sure developers of standard user applications are aware that they cannot and should not use the commands for traffic shaping. Most hidden APIs are already restricted to system applications only, but there may be alternatives which do not rely on heavier API changes.

AOSP allows anyone to make changes to the stock operating system. The changes shown in this implementation show one of many possible ways in which the traffic shaping framework can be implemented. The one presented tries to fit with the ideas of the software stack and isolation of privileges as best as possible. The end result of the system modifications allow traffic shaping to be used in a way that does not disrupt the user while still being effective.

Chapter 5

Conclusion

5.1 Summary

As a whole, the system provides an implementation that expands upon algorithms from prior work to create a distributed system for rate allocation. This rate allocation system is simulated on Wi-Fi but is one that could potentially be used on mobile networks such as those using LTE. The algorithms are split between an implementation that exists on the modified Android operating system on the UE and one that lies on a server designated as the resource broker which also communicates with a base station as presented in Chapter 3.

The main focus of the implementation is the traffic shaping framework on the UEs which use the rate determined by the algorithm to limit the UE's total rate and rates per application by classes. The implementation uses services that are added into the Android mobile operating system that work on devices with no interaction required from the user. Applications on the system contribute to a pair of utility functions that work with the shadow pricing algorithm when the UEs communicate with the resource broker.

The results of the algorithm are passed from a user-level application to a lower level service that sets up traffic shaping rules to enforce the rate allocation on the UE. Traffic shaping is enforced using the HTB queuing discipline with a main inner class and two leaf classes. These classes limit the rate for each UE on the network but also limits between classes of applications which can be elastic or inelastic. The two leaf classes are used to split the rate between inelastic and elastic applications while still allowing token sharing from one class to another if one of the classes has resources free.

As shown in Chapter 4, traffic shaping on UEs results in a more fair distribution of a network's resources per user and ensures that UEs do not use more resources than they need. It also serves to optimize the entire bandwidth available from a base station, which in this implementation parallels that of an eNodeB in LTE networks. The results of this

implementation show that a system of distributed rate allocation and enforcement of that rate by user and by applications for those users is possible. With the framework provided, the implementation could be expanded to other networks and devices to improve mobile networks such as LTE networks as well.

5.2 Future Work

As mentioned in Section 4.4, the key focuses to allow an implementation based upon the one presented to succeed are to make the system more flexible to change and have it well tested.

The implementation provided contains several ideas which were not fully implemented in the final result. Future work would expand upon these ideas as well as those mentioned about flexibility and security. Most work would refine the processes for application classification and utility functions for the applications and the UE.

For expansion upon application classification, development could be done to improve how to detect applications can be classified as inelastic or elastic by their behavior or packaged data. The Android manifest file contains data for an application's permissions and other metadata. Classification could be expanded to analyze patterns in that metadata or detect how applications use resources on the operating system itself.

The utility function calculation on the UEs could be improved to be more efficient and take into account historical data. All components of the implementation log and record data, but the data is not utilized heavily. Expansion upon that would improve how past rate allocations may affect future algorithm iterations so that heavy users may receive more or less resources depending on how they have used their past allocations.

Security in maintaining the traffic shaping system would also be something to note. On Android devices, if the user gains access to root privileges, the user can modify and run any code they wish. The implementation makes use of the tc tool for managing queuing disciplines on the network devices. This tool makes it easy for the implementation covered to set up the HTB qdiscs when traffic shaping is active. However, any root user can run one command with the tc tool to remove those. An avenue of expansion could be to integrate the services and commands deeper into the framework and system components to manage the qdiscs to be less susceptible to interference by experienced users.

Expansion to other operating systems used on UEs would also be a topic to consider. The implementation shown uses the Android Open Source Project due to its open source nature. This makes modification of the operating system easier for vendors compared to other major mobile operating systems in the market. If this system were to be implemented, it needs to be expanded to those also include systems which are closed source as well. Expansion in this regard may be to standardize the details of implementation in order for hardware and software vendors to implement the system properly.

Bibliography

[1] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2002, p. 213.

[2] L. Goasduff and C. Pettey, "Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth," *Visited April*, 2012.

[3] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018," 2014.

[4] "Ntt docomo "xi" lte subscribers top 5 million," NTT DOCOMO, 2012, accessed: 2013-05-08. [Online]. Available: https://www.nttdocomo.co.jp/english/info/media_center/pr/2012/001606.html

[5] Cisco, "Vni mobile forecast highlights, 2013 - 2018," 2013, accessed: 2014-05-08. [Online]. Available: http://www.cisco.com/assets/sol/sp/vni/forecast_highlights_mobile/index.html

[6] M. Butler, "Android: changing the mobile landscape," *Pervasive Computing, IEEE*, vol. 10, no. 1, pp. 4–7, 2011.

[7] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 179–194.

[8] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 281–287.

[9] A. Abdelhadi and C. Clancy, "A robust optimal rate allocation algorithm and pricing policy for hybrid traffic in 4g-lte," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 2185–2190.

[10] D. O. Awduche, "Mpls and traffic engineering in ip networks," *Communications Magazine, IEEE*, vol. 37, no. 12, pp. 42–47, 1999.

[11] J. Nagle, "On packet switches with infinite storage," 1985.

[12] M. A. El-Gendy, A. Bose, and K. G. Shin, "Evolution of the internet qos and support for soft real-time applications," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1086–1104, 2003.

[13] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks-the single node case," in *INFOCOM'92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE.* IEEE, 1992, pp. 915–924.

[14] A. K. Parekh and R. G. Gallagher, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Transactions on Networking (TON)*, vol. 2, no. 2, pp. 137–150, 1994.

[15] D. Ivancic, N. Hadjina, and D. Basch, "Analysis of precision of the htb packet scheduler," in *Applied Electromagnetics and Communications, 2005. ICECom 2005. 18th International Conference on.* IEEE, 2005, pp. 1–4.

[16] L. Georgiadis, R. Guérin, V. Peris, and K. N. Sivarajan, "Efficient network qos provisioning based on per node traffic shaping," *IEEE/ACM Transactions on Networking (TON)*, vol. 4, no. 4, pp. 482–501, 1996.

[17] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong, "Scalable architectures for integrated traffic shaping and link scheduling in high-speed atm switches," *Selected Areas in Communications, IEEE Journal on*, vol. 15, no. 5, pp. 938–950, 1997.

[18] R. G. Garroppo, S. Giordano, S. Lucetti, and E. Valori, "The wireless hierarchical token bucket: A channel aware scheduler for 802.11 networks," in *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a.* IEEE, 2005, pp. 231–239.

[19] J. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, and O. Sallent, "A hierarchical token bucket algorithm to enhance qos in ieee 802.11: proposal, implementation and evaluation," in *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, vol. 4. IEEE, 2004, pp. 2659–2662.

[20] K. Wongthavarawat and A. Ganz, "Packet scheduling for qos support in ieee 802.16 broadband wireless access systems," *International Journal of Communication Systems*, vol. 16, no. 1, pp. 81–96, 2003.

[21] N. Blefari-Melazzi, A. Detti, I. Habib, A. Ordine, and S. Salsano, "Tcp fairness issues in ieee 802.11 networks: Problem analysis and solutions based on rate control," *Wireless Communications, IEEE Transactions on*, vol. 6, no. 4, pp. 1346–1355, 2007.

[22] F. Y. Li and N. Stol, “Qos provisioning using traffic shaping and policing in 3rd-generation wireless networks,” in *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, vol. 1. IEEE, 2002, pp. 139–143.

[23] Y. Zaki, T. Weerawardane, C. Gorg, and A. Timm-Giel, “Multi-qos-aware fair scheduling for lte,” in *Vehicular technology conference (VTC spring), 2011 IEEE 73rd*. IEEE, 2011, pp. 1–5.

[24] H. Luo, S. Ci, D. Wu, J. Wu, and H. Tang, “Quality-driven cross-layer optimized video delivery over lte,” *Communications Magazine, IEEE*, vol. 48, no. 2, pp. 102–109, 2010.

[25] M. Salah, N. A. Ali, A.-E. Taha, and H. Hassanein, “Evaluating uplink schedulers in lte in mixed traffic environments,” in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–5.

[26] M. K. Karray, “Analytical evaluation of qos in the downlink of ofdma wireless cellular networks serving streaming and elastic traffic,” *Wireless Communications, IEEE Transactions on*, vol. 9, no. 5, pp. 1799–1807, 2010.

[27] F. P. Kelly, A. K. Maulloo, and D. K. Tan, “Rate control for communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research society*, pp. 237–252, 1998.

[28] P. Hande, Z. Shengyu, and M. Chiang, “Distributed rate allocation for inelastic flows,” *Networking, IEEE/ACM Transactions on*, vol. 15, no. 6, pp. 1240–1253, 2007.

[29] “netfilter,” 2010. [Online]. Available: http://www.netfilter.org/

[30] D. Stiliadis and A. Varma, “Latency-rate servers: a general model for analysis of traffic scheduling algorithms,” *IEEE/ACM Transactions on Networking (ToN)*, vol. 6, no. 5, pp. 611–624, 1998.

[31] E. J. Vergara, J. Sanjuan, and S. Nadjm-Tehrani, “Kernel level energy-efficient 3g background traffic shaper for android smartphones,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE, 2013, pp. 443–449.

[32] M. Xiao, N. B. Shroff, and E. K. Chong, “A utility-based power-control scheme in wireless cellular systems,” *Networking, IEEE/ACM Transactions on*, vol. 11, no. 2, pp. 210–221, 2003.

[33] M. Alvarez, E. Salami, A. Ramirez, and M. Valero, “A performance characterization of high definition digital video decoding using h. 264/avc,” in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*. IEEE, 2005, pp. 24–33.

[34] S. Winkler and C. Faller, “Perceived audiovisual quality of low-bitrate multimedia content,” *Multimedia, IEEE Transactions on*, vol. 8, no. 5, pp. 973–980, 2006.

[35] S. Jumisko-Pyykkö and J. Häkkinen, “Evaluation of subjective video quality of mobile devices,” in *Proceedings of the 13th annual ACM international conference on Multimedia*. ACM, 2005, pp. 535–538.

[36] R. G. Garroppo, S. Giordano, S. Lucetti, and G. Risi, “A comparison of htb based channel-aware schedulers for 802.11 systems,” in *Wireless Internet, 2005. Proceedings. First International Conference on*. IEEE, 2005, pp. 2–9.

[37] “Android security overview,” Android Open Source Project, 2014, accessed: 2014-03-03. [Online]. Available: http://source.android.com/devices/tech/security/

[38] “Android-adding systemservice,” Texas Instruments Wiki, 2012, accessed: 2013-04-23. [Online]. Available: http://processors.wiki.ti.com/index.php/Android-Adding_SystemService

[39] “ifb,” Linux Foundation, 2009, accessed: 2014-03-28. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/ifb