Performance and resource usage of multi-link HTTP in modern smartphones

Marius Alexander Skjolden

Thesis submitted for the degree of
Master in Informatics: Programming and Networks
60 credits

Institutt for informatikk
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019

Performance and resource usage of multi-link HTTP in modern smartphones

Marius Alexander Skjolden

Performance and resource usage of multi-link HTTP in modern smartphones

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The user experience of smartphones heavily rely upon functional internet connectivity, but, even though smartphones have multiple network links for accessing the internet, they are rarely able to utilize more than one link at the time.

This thesis proposes algorithms for multi-link HTTP in smartphones, which uses an increasing amount of input parameters enabling analysis into which input parameters are needed for optimal performance. Further, it studies the performance and resources used in modern Android-based smartphones when utilizing the proposed algorithms for link aggregation over Wi-Fi and 4G/LTE.

We have performed numerous experiments in various scenarios to observe the behavior and evaluate the performance of our method. The results show that the proposed multi-link algorithms were able to utilize an increasing degree the available aggregated bandwidth, and in the best cases, the throughput gain was considerable reaching numbers of 130Mbps. The multi-link algorithm utilizing all of the available input parameters was the one who managed to utilize most of the aggregated bandwidth, in both an isolated scenario, as well as dynamic scenarios. Further, the CPU and memory resources were affected to a lesser degree compared to energy usage. In some scenarios, the energy usage for the multi-link algorithms was comparable with that of a single-link approach. However, in scenarios where the link bandwidth differed significantly, the multi-link algorithms presented an increased energy usage due to a baseline energy-peak. Meaning, a decreasing throughput does not decrease the energy usage below a certain point.

Concluding that, the input parameters; throughput, bandwidth, and RTT were all needed to utilize most of the available aggregated bandwidth. Further, in scenarios where the bandwidth of both links reaches above 30Mbps, the energy usage was comparable to utilizing the best single-link approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Smartphones are becoming increasingly more powerful for every new release, and manufacturers are often releasing at least one new model every year. The newest models promise larger screens, more processing power, more memory, and more. These improvements enable smartphone users to adopt many of the traditional computer tasks such as; gaming, video streaming, emailing, and social media. The user experience of such services relies heavily on good connectivity and high bandwidth access to the internet. Where desktop computers often used wired network links for accessing the internet, modern smartphones primary rely on multiple wireless networks links like Wi-Fi and or LTE.

However, even though the smartphones have multiple network links, they are rarely able to utilize more than one link at the time. The devices often turn off or hibernate the LTE link as soon as connected to a Wi-Fi network.

By utilizing both the links simultaneously and aggregating the bandwidth, or sharing the network traffic over multiple links, many smartphone users could get a significant boost in download performance, as well as overall connectivity stability. In ideal situations, aggregating links could achieve a sum of the combined available bandwidth.

Currently, the most available solution for multi-link on Android smartphones are; Samsung's "Download Booster" [37]. However, this functionality has some limitations and works only with certain services [5]. Apple iOS devices has a function called "Wi-Fi Assist" [1]. This function is not to be confused with concurrent multi-link functionality. Apples "Wi-Fi assist" only enable faster switching to the LTE link if the device detects a poor Wi-Fi link.

Due to the limitations of the Download Booster, found in [5], in particular, it only works with selected download services, and it only utilizes 80-90% of the combined bandwidth, the first motivation for this thesis is to propose dynamic multi-link approaches. The multi-link approaches should be capable of; performing in networks with varying signal strength, bandwidth, and capacity, and, communicate with all HTTP

servers supporting Range Request. The approaches should use different input parameters, analyzing what it takes to utilize more of the combined aggregated bandwidth and to see how it affects the devices.

Most of the current research into multi-link on smartphones are a couple of years old, [44], [52], [41], and since then newer generation of smartphones and newer wireless technologies have become available. These smartphones and networks are providing improved efficiency and bandwidth numbers in order of magnitudes higher compared to the previous work done in the field.

Besides, for monitoring energy-usage, the previous work had to rely on hardware measurements with external equipment connected directly to the internals of the device. This kind of measurements are costly, intrusive, and makes it difficult to compare multiple devices. Most modern smartphones are now equipped with high precision power-management chips, enabling monitoring energy-usage with software.

Therefore, the second motivation for this thesis is; utilize the advantages of modern smartphones and look into how multi-link HTTP perform and affects the devices in combination with newer wireless networks like LTE-A and 802.11ac.

1.2 Problem definition

With smartphones and wireless networks in constant development and becoming increasingly more powerful, effective, and capable – one starts to question how today's smartphones would be affected by using multiple links concurrently. Are the devices capable of utilizing all of the available bandwidth? If so, how do these higher bandwidths affect CPU, memory, and energy usage? Are the newer hardware more efficient than older? How do multi-link compare across multiple modern smartphones?

The research questions for this thesis are;

Question 1 – Input parameters Based on the first motivation from the previous section, we decided to propose several multi-link HTTP approaches with a increasing degree of input parameters. Having multiple approaches makes it possible to analyze what is the most critical input parameter in a mobile environment. Therefore; *What input parameters are most important for multi-link HTTP approaches in a mobile environment consisting of smartphones and wireless networks?*

Question 2 – Performance Inspired by the previous work, we are going to monitor how the smartphones are affected by the performance gain and resource usage of the multi-link approaches, and compare them to each other, including standard single-link HTTP downloads. Therefore; *With the correct input parameters, what multi-link performance gains can we expect in modern smartphones, and how are the device and its limited resources affected?*

1.3 Scope and limitations

Modern smartphones primary consist of Android-based smartphones, Apple iPhones, and Windows phones. However, smartphones and their operating systems are a fast-moving and volatile target. New features and patches are released all the time, with unofficial sources reporting mayor releases happening about once a year. Based on the research questions above and the wast smartphone landscape, we decided to limit the work by looking into multi-link HTTP using Android-based smartphones with wireless networks like Wi-Fi and LTE.

For the experiments, we decided to look into two primary groups of scenarios, an isolated and a dynamic. The isolated scenarios acted as a controlled environment, where we were able to isolate the Wi-Fi network from other devices. In the dynamic scenario, we used two public Wi-Fi's, representing more typical Wi-Fi situations. For both the scenarios, we utilized the publicly available LTE network.

1.4 Main contributions

The main contributions of this thesis are;

Propose approaches for multi-link HTTP, using an incremental degree of input parameters. Chapter 4 studies the challenges of multi-link and evaluates which input parameters are available and needed to perform dynamic request-size calculations. Five approaches are proposed; a default single-link approach, a naive multi-link approach, and Algorithm 1, 2, and 3 for the final multi-link approaches.

Propose implementations prepared for proxy service functionality, to be used for all HTTP traffic on a smartphone. Three implementations are proposed; a single-link implementation that is supporting the single-link approach, an implementation for the naive approach, and finally a client application supporting Algorithm 1, 2, and 3 through configurations.

Perform experiments with the approaches on modern smartphones connected to modern networks. In total, five experiments are performed over two main scenarios; the isolated and dynamic scenarios. These two scenarios presents unknown variables like availability and signal strength of the Wi-Fi network.

Compare the proposed approaches, and analyze what input parameters are the most important, and show how the selected smartphones are affected. Three result chapters 5, 6, and 7 are provided with results and analysis from the experiments. The result chapter is divided into the metrics; throughput as a representation for time, CPU and memory for resources, and energy usage also for resources.

Conclude A conclusion that shows important findings, and recommended improvements for further study. In the final chapter 8 we take a final look at the presented results and compare the cumulative effect they have on the smartphones. Besides, we propose extra features for the approaches and implementations, like; slow link fail-over, request continuation, and link banning. These extra features are a start of the work needed for implementing a fully featured proxy service.

1.5 Research method

Analyzing performance in a computer system can be done in a variety of ways. According to [57], the three most common are; *analytical modeling*, *simulations*, and *measurements*. This thesis decided to use measurements, due to the difficult nature of simulating energy usage on a smartphone. The measurements make it possible to evaluate the performance (time and resources) of the proposed multi-link approaches. Several scenarios are needed to perform a thorough evaluation. We will be spending some time setting up a test-bed for the experiments, addressing and discussing various implementations alternatives, followed by several measurements to determine performance and resource usage.

During the experiments we will be monitoring the metrics;

- *Time:* Throughput
- *Resource:* CPU and memory usage
- *Resource:* Energy usage.

1.6 Outline

The following chapters organize this thesis;

Chapter 2 – Background In this chapter, we start by introducing some central terms and functions for multi-link, next, we go through current research on multi-link and multi-link on smartphones. Finally, we describe technologies for enabling high-level multi-link with HTTP on smartphones.

Chapter 3 – Methodology In this chapter, we start by presenting the methodology used in this thesis. Next, we present the setup for the test-bed, and finally we explain the numbers and figures used in the result chapters.

Chapter 4 – Proposed Algorithms and Implementation In this chapter, we present the motivation and background of the proposed algorithms. Next, we present the proposed Algorithm 1, 2, and 3. Finally, we present the implementations enabling the algorithms.

Chapter 5 – Results: Throughput This chapter presents and analyzes the throughput results from the experiments.

Chapter 6 – Results: CPU and Memory usage This chapter presents and analyzes the CPU and memory usage results from the experiments.

Chapter 7 – Results: Energy usage This chapter presents and analyzes the energy usage results from the experiments.

Chapter 8 – Analysis and Discussion In this chapter, we analyze further the results presented in the previous three chapters. After, we discuss some limitations and remaining questions.

Chapter 9 – Conclusion Final conclusion for this thesis.

Chapter 2

Background

In this chapter, we start by introducing some necessary background information on central terms and methods used in this thesis. Next, we present related work in the field of; multi-link challenges, multi-link using smartphones, energy usage in mobile devices, and tools for monitoring energy usage. Finally, we present current technologies and functionality with information on Samsung's Download Booster, HTTP/2, and Mobile communication networks. These sections are the main motivations and inspirations for this thesis.

2.1 Basics

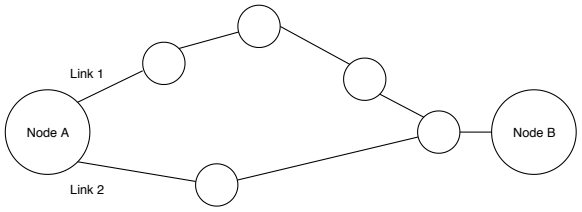


Figure 2.1: Basic multi-link node network

Multi-Link In this thesis, the term multi-link refers to a network node utilizing more than one path for communicating with others. In figure 2.1, *Node A* have two links, and therefore paths, for communicating with *Node B*, and instead of only utilizing only one of these paths at the time, the multi-link *Node A* utilizes both paths concurrently. More specifically, a multi-link node can be a network device installed with multiple network cards (links) for communication. Such a setup can use the links for boosting bandwidth in a closed environment, like a server-to-server scenario.

One example with the figure above; Node A is a multi-link server with Link 1 and 2 of bandwidth 30bps and 50bps (bits per second), respectively. Node B, the other server, only have one network link with a bandwidth of 100bps. If Node A utilizes only one of its network links for communicating with Node B, it would in the best case have a bandwidth of 50bps through

Link 2, regardless of what Node B is capable of. If Node A utilizes both links concurrently, and share the communication load between both paths, the node would be able to aggregate the bandwidth and achieve 80bps.

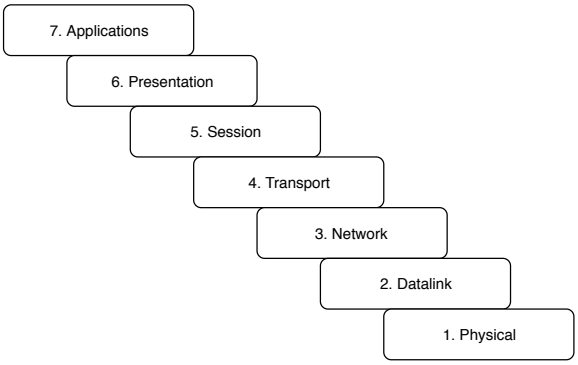


Figure 2.2: The seven layers of the ISO/OSI reference model.

Application layer Implementing multi-link in network devices can be done in many ways, and one of the most common ways is to focus on one of the layers in the ISO model – visualized in figure 2.2. Some examples of existing solutions, focusing on one layer, are;

- *4.Transport* : "Stream Control Transmission Protocol (SCTP) [70]" , and "Multi-Path Transmission Control Protocol (MPTCP) [54]".
- *3.Network* : "Equal-Cost Multi-Path routing (ECMP) [56]".
- *2.Datalink* : "Multi-link Point-to-Point Protocol (MPPP) [47]".

Without going further into the details of how these lower level protocols operate, approaches utilizing them all have in common that they need to integrate with the operating system at a kernel level. Higher level approaches can be implemented without this concern, and the HTTP protocol is an example of the highest level, the application layer, protocol.

HTTP "The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems" – [61]. With HTTP being an application protocol, it is layered on top of TCP, having the advantages of error-free transportation, in-order delivery, and unsegmented data streams. HTTP is mostly used for communication over the world wide web between browsers and servers. An HTTP transaction consists of a request from the client, and a stateless response from the server. Each message, both request and response, is divided into a head and body part, where the header is used for metadata, and the body is used for the data payload. Some of the HTTP features for enabling multi-link are; Range-Requests and Pipelining.

Range Request With the introduction of HTTP 1.1 in 1997, range as a header request field was added [16]. This feature makes it possible for HTTP clients to request a part of a file using a range notation that the server supports. An example of this is the first 100KB of a 1MB payload. If the server validates the request, it sends back an HTTP response code 206 (Partial Content) instead of 200 (OK), and a content range header field describing the range [17]. The 206 response also includes a slimmer header to prevent metadata redundancy. HTTP code 416 is reserved for illegal range requests, like negative ranges or ranges outside the size of the payload. HTTP servers are not obligated to support the range request feature, but if they do, they must only allow it for "GET" request types [60]. With this feature added, clients can segment a download request into parts, and download managers typically use this feature for supporting the continuation of a download at a later time in a stateless manner.

Pipelining Traditional HTTP pipelining is a technique for sending multiple requests to a server, over the same TCP connection, without waiting for the responses. This enables pipelining to improve download times by utilizing more of the available bandwidth. However, this feature does not come without a set of problems, one of them being "Head of Line" blocking [8]. Pipelining has been superseded by a technique called "multiplexing" in HTTP/2, which addresses this among other problems – section 2.3.2.

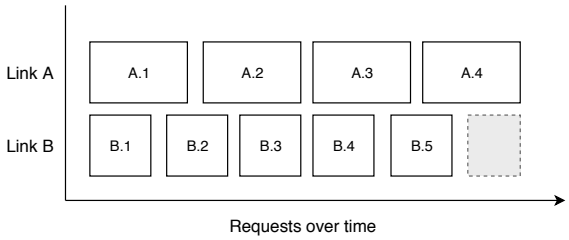


Figure 2.3: Visualizing the Head of Line problem.

Head of line Head of line (HoL) blocking is a phenomenon which makes a queue or line of packets waiting for the first packet [39]. For HTTP multi-link with range requests, this translates to an over-sized request at the end of a download. Figure 2.3 visualized this where Link B if left underutilized due to Link A's large request A.4. Both links should be utilized at all times, and the term is later referred to as Link Δ in this thesis. Meaning the difference between when the individual links where finished, or the size of the gray box in figure 2.3.

Share calculations The *Range Request* and *Pipelining* feature of HTTP, allows a client to request parts of a file asynchronously. With these features, it is possible to implement a system where multiple links split an HTTP request, into multiple smaller requests. For this, there exist two primary

ways of calculating the size of the smaller requests; the static and the dynamic.

Static request-sizes A static request-size approach shares the load equally between the links and do not make any adjustments according to the current link throughput, and thereby creating a less than an optimal algorithm for link aggregation. If one of the links experiences a drop in the expected throughput, or an increase in packet loss, or becomes unstable, the static request-size approach could amplify the HoL problem and thereby creating a bottleneck. The approach would then have to wait for the slower link to download its equal share before completing. This could be circumvented by evolving the algorithm and adapting it to handle unresponsive links by transfer their load over to the more responsive ones; however, this could still degrade overall performance. The static request-size approach is best suited in a closed environment like directly between server-to-server.

Dynamic request-sizes With the static request-size approach providing a naive algorithm for sharing HTTP requests over multiple links, a more effective approach is a dynamic request-size approach. This approach adjusts the share between the links dynamically based on the multiple input parameters like current throughput. As an example, the algorithm could distribute the load equally for the first requests, but as soon as possible it would adjust the request sizes based on the measured throughput. The dynamic request-sizes can be calculated by, first provide a share between the links and then multiply by a segment size:

$$\text{Link A request size} = \frac{\alpha}{\alpha + \beta} * segmentSize$$

$$\text{Link B request size} = \frac{\beta}{\alpha + \beta} * segmentSize$$

Figure 2.4: Calculating dynamic request-size for link A and B. α and β is representing the throughput of link A and B, respectively.

The *segmentSize* in figure 2.4 acts as an multiplication factor and upper limit for the request-sizes.

2.2 Related Work

In this section, we present research previously done in the field of multi-link. We start by introducing multi-link challenges, next, multi-link over HTTP, and then, multi-link on smartphones with various approaches. In the last subsections, we introduce research done in the field of energy usage with smartphones, and existing tools for monitoring, before we finish with a discussion.

2.2.1 Multi-Link – Challenges

Section 2.1 described an example of a simple multi-link setup, but often in such setups, there exist challenges like; the paths have different bandwidth, the paths have different response time, and one of the paths could become unavailable, experience increased packet loss, or suddenly become slow due to capacity limitations. A multi-link approach, running on Node A, would have to account for such challenges and dynamically adapt to utilize all of the aggregated bandwidth. This thesis focuses on the situations where a smartphone have multiple links with separate internet service providers (ISP), like one ISP for the Wi-Fi link and another for the LTE link. A client with multiple different ISPs probably has very different paths to a remote server, causing an amplification of the challenges compared to the closed environment, like direct server-to-server.

Multipath Transmissions Multipath transmissions over the Internet is a well-researched area, and in [62], Li et al., they make a survey focusing on previous work that has been done. One of the main challenges they bring up is the way traditional TCP/IP is implemented to choose the best path according to routing metrics. By the time this protocol was created the available hardware did probably not include a multitude of high-speed wireless links, like the smartphones we see today. As a result of this, TCP/IP was not developed with a focus on multiple links sharing a workload, and today's smartphones are, often, under-utilizing the available network resources by only using one link at a time. When developing a new protocol addressing this issue, the focus is on improvements, and some of the objectives they mention are:

Reliability by having multiple links connected to a single source, data transfers can be quickly moved from one link to another if the link experiences failures or sub-optimal performance.

Bandwidth aggregation by having two or more links to a source, the bandwidth and thereby the throughput is increased, which leads to improved performance.

Security by splitting transmissions into multiple paths, it would become harder for a malicious third party to gather information.

2.2.2 Multi-Link – With HTTP

In section 2.1 multiple ways of approaching multi-link was mentioned. This thesis focuses on the application layer approach using the HTTP protocol. Therefore we present work using HTTP for sharing the traffic between multiple links, both for general traffic and a solution for video streaming specific service. These works were the primary motivation for the later proposed algorithms in chapter 4.

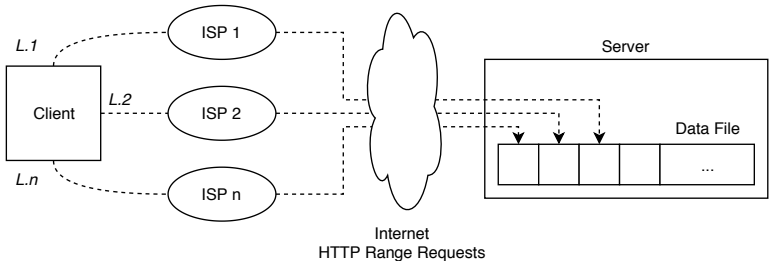


Figure 2.5: Visualizing a client having multiple links, and how these links are responsible for requesting parts of a bigger file from a server. *ISP* = Internet Service Provider.

Aggregating the Bandwidth of Multiple Network Interfaces In [50], Evensen, describe approaches of aggregating bandwidth across multiple network links, and in the chapter of "Application-specific bandwidth aggregation" an HTTP based multi-link approach is described. HTTP is enabling many of the technologies and services used on the Internet today. Therefore, implementing a multi-link approach using HTTP makes it highly available and has a wide range of uses, as opposed to a custom protocol that would demand specific server applications, making widespread deployment and adaptation a challenge.

To achieve load sharing with HTTP, the "Range retrieve request [60]" feature was used. This feature allows the requester/client to specify a smaller part of the payload through request header parameters, like the first 100KB of a 1MB payload. In addition to range requests, the HTTP pipelining [24] feature was also utilized. Evensen writes: "By combining the aforementioned technologies, range request and pipelining/multiplexing, we could hope to utilize the total of the available aggregated bandwidth" – [50]. This should work for any file size, and be network independent. Also, an implementation of this could adjust the segment sizes dynamically based on available throughput and the network delays. This has the advantage of adapting to changes in network quality over wireless networks like Wi-Fi and LTE. Two main subsegment approaches were described; a static and a dynamic approach. These two approaches stand as the foundations for the static and dynamic request-size calculations – section 2.1.

Improving performance of video streaming In the paper [49], Evensen, a video streaming specific implementation is described. With video streaming being an essential and bandwidth consuming activity on the Internet, the approach described here is an optimization of the previous mentioned dynamic-subsegment approach [50] and is specified for the DAVVI video streaming platform [58], Johansen et al. Each video is divided into independent parts or segments with constant duration. This allows for the segments to be split into subsegments, and then requested over different links, utilizing all of the combined available bandwidth. Splitting videos into segments like this, and further splitting the segments into

subsegments enables a solution which reduces the number of interruptions and improves the quality since the total bandwidth is increased. The way the video streaming service splits a large video into smaller segments was the inspiration behind the sizes of the segments acting as upper limits – section 2.1.

2.2.3 Multi-Link – Using Smartphones

This thesis has a goal of proposing multi-link approaches using the application layer protocol HTTP and testing them on Android smartphones. We decided to use HTTP due to the wide adaptation of this protocol for many of the services used on the internet today. Gaining insight into how others have approached multi-link on smartphones, we present similar work using HTTP or other custom solutions. The works all have in common that they approach multi-link on the application layer.

Application-level link aggregation on Android terminals The approach described in [68], Takiguchi et al., is entirely different from the HTTP range request approach previously mentioned. In this paper, both the client and server application uses a custom protocol. Creating a custom application layer protocol has both advantages and disadvantages. The advantages of this are control, and the possibility to specify and implement a protocol tailor-made for load sharing applications, with the disadvantaged being challenges in regards to communicating with other servers outside of the closed environment. The custom protocol they describe has many of the same characteristics as the HTTP approach described in [50], with dynamic distribution between the links based on the available throughput. What is particularly interesting in the approach they described is how they made it work on Android smartphones. They write: "the application software uses Android runtime to issue Linux command to specify and manage the network interface cards for link establishment."

An application layer protocol for energy-efficient bandwidth aggregation In [69], Tang et al., they develop a bandwidth aggregation prototype named: "Application Layer Protocol based Aggregation (ALP-A)". The ALP-A prototype is implemented to work on Android devices with an energy-aware design. The device used in this study is a Samsung Galaxy Wonder phone and according to gsmarena.com was released in October of 2011. This device supports 3G connectivity using WCDMA, and the Android version at the time of measurements was 2.3.3, released late 2010. For single link connectivity, they conclude that Wi-Fi has the minimum energy usage under all scenarios and that 3G always leads to the worst energy performance. Finally, ALP-A, compared with other solutions at the time, saves about 16% energy under web browsing and video streaming.

GreenBag: Energy efficient Bandwidth Aggregation In the paper [44], Bui et al., they contribute to a bandwidth aggregation solution for video

streaming, to minimize video playback time and energy usage. The implementation is a middleware supporting real-time delivery over mobile wireless networks, utilizing HTTP pipelining, medium load balancing, and energy-aware link mode control. The implementation is also compiled to run on an Android smartphone. They conclude by saying that the GreenBag implementation can conserve energy by 14%-25%, compared to a non-energy-aware throughput maximization case.

An energy efficient HTTP adaptive video streaming In [52], Go et al., they utilize HTTP for an adaptive streaming algorithm. To quote their main goal; "The goal of the proposed algorithm is to provide a seamless, high-quality HTTP adaptive streaming service in energy and networking cost-efficient way over heterogeneous wireless networks." – [52]. In their experiments, they use a Samsung S4 LTE-A version and measure energy usage via an external hardware device. The performance of the algorithm in this study is compared to GreenBag [44] and DynSub [50], and they conclude with an average of 11.62% improvement in energy usage.

The resource optimization of heterogeneous network interfaces In [41], Ban et al., they describe their motivation by; "design and build a theoretical framework including power and good-put constraints via a two-level convex optimization problem." The implementation is middleware running on the client side and requiring no changes to the server side. They tested this on Android phones with multiple network access enabled including LTE and Wi-Fi. Comparing this with an LTE only method, they show that the approach can gain 1.8 times good-put increment, with only 50% of the energy usage. The term *good-put* they are referring to is a subset of throughput where only the useful data is accounted for.

Optimal multi-interface selection for mobile video streaming In [65], Moon et al., they approach the challenge of multi-link HTTP in mobile devices by focusing on video streaming and link selection. Their approach is both data-usage-aware and energy-aware with a target of completing the video playback before the battery is depleted. Compared to the other papers mentioned above, this is the only one with a focus on Apple iPhones. They enable this by using native iOS system functionality creating a user-defined link. With these links, they can make requests through iOS internal HTTP engine. They prove that this performs in comparison with conventional video streaming.

2.2.4 Energy usage in mobile devices

Smartphone development aims to match the functionality and performance of traditional desktop computers; this includes a powerful CPU, a large amount of memory, and wireless connectivity across multiple networks like Bluetooth, GPS, Wi-Fi, GSM, LTE, NFC, infrared, and more. Also, a prominent and bright touchscreen, graphics hardware and audio. Devices

being more packed with functionality leaves less space for a battery, and therefore energy usage becomes an important factor. In this subsection, we present work related to analyzing the energy footprint of smartphones, and how to measure energy usage, because of how this is relevant in the later experiments.

In [45], Carroll et al., they point out that the majority of energy usage can be attributed to the GSM module and the screen. Further, they write that the GSM module consumes a lot of both static[1] and dynamic[2] power, and keeping the link available contributes notably to the total power usage. The following table shows their energy usage findings of the Android devices; Openmoko Neo Freerunner, HTC Dream, and a Google Nexus One.

Benchmark	Freerunner	HTC Dream	Nexus One
Suspended state	103.2	26.6	24.9
Idle	333.7	161.2	333.9
Phone call	1135.4	822.4	746.8
Network Cell	929.7	1016.4	825.9
Network Wi-Fi	1053.7	1355.8	884.1

Table 2.1: Data from [45] showing average system power in mW, excluding screen backlight.

Even if [45] admits these devices are outdated, with the Freerunner lacking 3G, they comment that higher data rates do not appreciably affect energy usage in practical situations.

In [40], Balasubramanian et al., they find that energy usage is closely related to the workload of the transfer, not only the total transfer size. With 3G as the transfer method network, a few hundred bytes transferred intermittently can consume more energy than, e.g., one megabyte in one shot. Further, they show that, with 3G, a significant fraction of energy, can be contributed to tail energy – with numbers almost as high as 60%. *Tail energy* is a concept where some system components, such as Wi-Fi, LTE, GPS, can exhibit a state of high energy usage, and remain in this state for some time beyond the end of the trigging routine. The other end of tail energy is referred to as ramp energy. *Ramp energy* is the energy spent in switching to a high power state before a transfer can be started. Tail and ramp energy are constants over both large transfer sizes, and smaller ones. Another exciting finding they show is that with Wi-Fi the tail energy overhead is comparable with 3G, but the actual transfer is considerably more efficient for all transfer sizes.

Tools for monitoring energy usage Monitoring energy usage in smartphones can be done in multiple ways. In [66], Serrano et al., they describe a test bed consisting of a hardware box powering and simultaneously monitoring energy usage at 5000 samples per second. In [45] they describe a

[1]Power consumed when there is no activity in the circuit – base load.

[2]Power consumed when the circuit is under load. Includes the static power.

testbed consisting of probing directly at the PCB (Printed Circuit Board) of the device, with both knowledge and modification to the internal schematics of the device. Both these hardware monitoring methods can result in accurate and high data definitions but depend on the device under test is compatible. It can also be intrusive, and might not scale across multiple devices because it can demand a specific solution for each device. Another way of monitoring energy usage is by software, and below are some approaches to this.

PowerAPI [30] is a middleware toolkit for tailor-made power measurements to accommodate user requirements. In [46], Colmant et al., they build a toolkit on top of PowerAPI (WattsKit) for monitoring energy usage of distributed systems, with an average error of 1.35%. Also, as mentioned in [43], Bourdon et al., they refer to research contributions using PowerAPI to measure the energy usage of programming languages, and the location of hot-spots within applications. This means PowerAPI have the capability to measure the energy usage at the application level, and also inside the application for specific parts like; networking, memory access, and disk access.

AppScope In [74], Yoon et al., they describe an Android kernel module for estimating the energy usage of Android applications, by using Kernel Activity Monitoring. It is event-driven using Linux *Kprobes*. AppScopeViewer provides real-time energy usage through a graphical interface (typically installed on a separate computer), by interacting with AppScope on a target device. Unfortunately, this project seems unmaintained for some years.

PowerTutor [73], Yang et al., is an Android application for displaying energy usage in significant system components like; CPU, network, display, GPS, and more. It does so by polling the Android *BatteryStats* service. Unfortunately, as of writing this, the project seems unmaintained for some years.

Batterystats and Battery Historian are tools for collecting and visualizing battery data on Android devices. As opposed to the tools developed from research like previously mentioned, these tools are developed and maintained by Google, which means the broader device and release support. For a regular Android device, Batterystats is always running in the background as a core service but can be reset for direct measurements. After a workload is completed, the data can be pulled from the device and loaded into the Battery Historian tool for visualization. The visualization allows for aggregated statistics, panning and zooming functionality, as well as A/B comparisons of two data dumps. These tools are not as granular due to the low sampling rate.

2.2.5 Discussion

In this section, we started by presenting; basic multi-link, multi-link using HTTP, and then some of the most current research into multi-link on smartphones. All of these works have different approaches and goals for why they implement multi-link. The two first works on multi-link with smartphones build a custom client-server environment, making the use very limited. The later works focus either only on video-streaming with goals of completing the video before the battery are drained, or are using older devices. This gave us the motivation to look a general purpose multi-link HTTP solution, utilizing modern smartphones and newer wireless networks. First, because of the previous work are specific to one goal, we want to find out what input parameters works best in a general purpose approach. We also want to see how newer smartphones are affected by the increased download performance of multi-link approaches.

The energy usage part presented both results from research previously done, in addition to hardware and software approaches to monitor energy usage. The tools for monitoring with hardware are in today's smartphones hard to implement without breaking or destroying the device. Modern smartphones are often glued together, with glass on all significant surfaces, requiring special tools to open them. The software solutions listed are, unfortunately, outdated, or not directly suited to the focus in this thesis. Because of this, we decided to rely on data from PMIC chips enabling accessible software monitoring. The chips are later described in section 3.2.1.

Other sources Android development is a fast-moving and volatile target. New features and patches are released often. This leads to depreciation and quickly outdated documentation, with data from only a couple of years ago no longer reproducible with the latest version of Android OS. These changes serve as a reminder of the ever-growing progression in the field. There is much knowledge to gather from sources like Google and StackOverflow, but it can be challenging validating them all. This thesis is therefore bound to make some predictions or assumptions based on such references.

2.3 Current Technologies and Functionality

In this section, we start by presenting Samsung's Download Booster, and why it lacks in some situations and functionality. Next, HTTP/2 and new features. Finally, we present LTE and Wi-Fi networks in mobile communications networks. Technologies like carrier-aggregation found in LTE networks are important for the later proposed algorithms in chapter 4.

2.3.1 Samsung's Download Booster

Some Samsung smartphones are equipped with a function called Download Booster. This function enables the use of multi-link in situations where

the device is connected to LTE and Wi-Fi. Samsung describes the download booster by; "The Download Booster is a feature that will allow your device to download large files (over 30 megabytes in size) faster by using a Wi-Fi connection and a mobile data connection simultaneously" – cited from the website [37]. Celerway [6] tested the download booster [5], and some of the key limitations they found were;

- High-quality LTE connections only, no support for 2G or 3G
- 80-90% combined bandwidth utilization
- Static segment size even if network quality changes
- Utilizes Wi-Fi even if the quality is poor
- Selected download services only

Getting further information on the Download Booster proved to be a challenge, and specific information on which devices are supporting this feature was not found.

2.3.2 HTTP/2

In 2009 Google developed the SPDY/2 protocol [33], and by that, they laid out the foundations for what today is known as HTTP/2. SPDY/2 was meant to overcome many of the challenges with HTTP/1.1 by reducing web page load and improving security. Some of the basic functionality with SPDY/2 was; Multiplexed streams, Request prioritization, and HTTP header compression. It also supported more advanced features like; server push, and server hint. SPDY/2 was however never meant to replace HTTP as a complete protocol, instead replace some parts of it. During this time the HTTP Working Group (HTTP-WG) made notes of this work and used that towards the official HTTP/2 standard [20]. This foundation made it possible to use SPDY/2 as a development branch for testing ideas before implementing them into the new HTTP/2 standard, and by November 2012 the first draft of HTTP/2 was ready. HTTP/2 main focus is on performance, and some of the important features are:

Multiplexing (successor to HTTP/1.1 pipelining) with then head-of-line problem being addressed. It is also expected to have some performance gain.

Header compression with HPACK serving as a custom compression algorithm, that leverages Huffman encoding to get compression rates that approach GZIP. Header compression is essential due to HTTP/1.1 having big headers in certain situations like additional cookie information and other metadata.

Designed to work over single TCP/IP connection as opposed to HTTP/1.1 use of multiple connections (up to 6). This is important due to TCP's congestion control algorithms (like Reno).

Back reference header values from the previous request makes it possible to chain headers, and thereby prevent the server and client from sending redundant data for each request, which introduces less overhead

Server Push gives the server the ability to send files to a client, without the client requesting them. This could be useful if a client requests a file like index.html, then the server can assume that style-sheet files and javascript files are almost certain to be requested for next. Instead of the server waiting for this request, it starts sending them to the client right away. Testing has shown that this technique can improve performance by 20% to 50%, as shown on page 72 in [64].

TLS encrypted by default. While not a requirement for the protocol, almost all of the major browsers require that TLS encryption is enabled for the use of HTTP/2.

All the experiments done in this thesis are done with the HTTP/2 protocol with TLS encryption enabled. Therefore, some of the improvements listed above are gained like multiplexing and header compression. We decided not to make a comparison study of HTTP and therefore did not utilize HTTP/1.1 for any experiments.

2.3.3 Mobile communication networks

Most of today's smartphones are equipped with at least two network links, and the most common are types WLAN (commonly referred to as Wi-Fi), and cellular networks like GSM, UTMS, and LTE(4G) with more. In 2013 mobile data grew 81% globally and is expected to grow significantly more in the coming years [53]. These networks have many physical differences which affect performance and ways of working. This section covers some of the histories of cellular networks, including LTE and its features, as well as current and future Wi-Fi technologies.

History of cellular networks GSM was developed in 1982 for real-time services like phone calls. GSM is circuit switched, with data only possible at meager data rates – [55] Hansen et al. GPRS was introduced around the year 2000 and was an evolution through the traditional packet-switched IP networks, but still using the same air interface and access methods – [59] Johnsen et al.

To enable higher data rates, UMTS (3G) was created as a successor to this using the then newly developed technology WCDMA (Wideband Code Division Multiple Access) – [67] Stette et al. This meant that the UTMS emulated circuit switched technology for real-time services while using packets switched communication for data. UMTS allocates IP addresses to the user endpoint when data service is established, and released as soon as the data service has ended. Incoming data services were still reliant on circuit switched technology.

After this, the purely IP based Evolved Packet System (EPS) was developed, with both real-time services and data services using a packet switched IP technology. The IP address is therefore allocated when the device powers on the link card – usually on startup. LTE (4G) is the access part of EPS.

LTE – Long Term Evolution The new access solution of EPS, introduced in March 2009, meant that a large bandwidth up to 20MHz with high data rates could be achieved. The highest theoretical peak is 75Mbps in the uplink, 300Mbps in the downlink. This, however, requires spatial multiplexing, which is a technology for transmitting data signals independently and separately known as streams. The potential of this technology (often referred to as MIMO – Multiple Input Multiple Output) depends on the number of antennas in the transmitter and receiver, meaning the standard maximum number of antennas; $P = min(N_r, N_t)$ where N_r is the antenna count for the receiver, and N_t is the antenna count for the transmitter. However, the multi-antenna count has shown to reach a limit in regards to device size, complexity, and cost.

LTE-Advanced is a further improvement upon LTE with features like; *carrier aggregation*, multi-antenna techniques, and support for relay nodes – [72] Wannstrom et al.

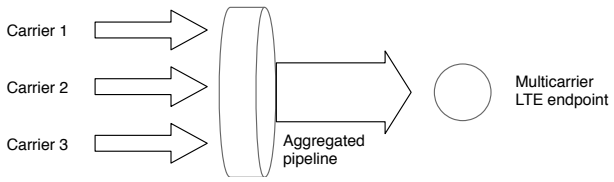


Figure 2.6: Figure illustrating carrier aggregation

Carrier aggregation (C.A) described in [71], is a technology for LTE-Advance for enabling increased bandwidth. The maximum bandwidth for LTE is 20 MHz due to backward compatibility with older devices, but carrier aggregation increases the bandwidth up to 100MHz by aggregating five carriers – often referred to as *component carriers* [75]. C.A. allows for data rates up to 1Gbps for downlink and 500Mbps uplink and works by combining multiple carriers from the same intra-band, or different inter-band networks. Described as:

Intra band (contiguous) uses a single band and is the easiest way to aggregate component carriers. This method is the best in regards to complexity, cost, capability, energy usage, and without making to many changes to infrastructure [75].

Figure 2.7: Intra-band, contiguous

Intra band (non-contiguous) still uses a single band, but have gaps. It can be difficult to allocate 100MHz bandwidth continuously, therefore this method provides a practical way to utilize the spectrum resources, including unused scatter frequency bands [75].

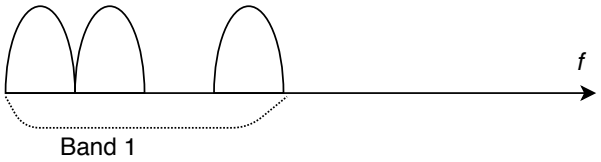


Figure 2.8: Intra-band, non-contiguous

Inter band (non-contiguous) uses different bands, and therefore exists only in non-contiguous form.

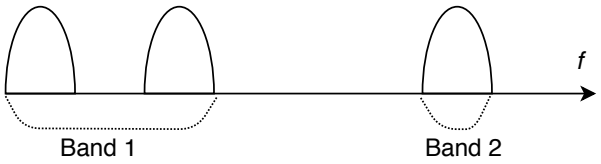


Figure 2.9: Inter-band, non-contiguous

Testing with carrier aggregation shows that when starting a network stream, the connections have to build up over time to utilize multiple carriers, and clients can experience a buildup in throughput over multiple seconds before stabilizing. Therefore, a smaller request might not be large enough to allow for this buildup. Also, a mobile device can not expect to acquire full carrier aggregation in every situation, even if the current location supports it, the network can be out of capacity. This can affect the way the multi-link approaches need to be implemented, as the only way an LTE link can get high throughput through carrier aggregation is to give it more significant request sizes even if it reports lower throughput than expected.

Differentiating a slow LTE link from an LTE link which has not yet reached its potential through carrier aggregation could prove to be complicated. A possible solution to this is a trial and error system which provides a slow LTE link with larger request sizes over a certain amount of time. This solution, unfortunately, could also amplify the last segment problem if the LTE link is slow and not in a buildup state.

5G LTE was introduced in 2009, and since then both 4.5G also referred to as LTE-Advance-Pro, and 5G has been announced. These newer networks promise higher bandwidths, with 5G aiming to reach 20Gbps. In additions to this, 5G's main performance targets are; reducing overall energy usage, reduced latency, cost reduction, and increased system capacity with larger device connectivity. At the time of this writing, consumer-grade equipment with 5G networks was not available.

Wi-Fi Wi-Fi, or WLAN technology, based on the IEEE802.11 standards [19], is a set of media access control and physical specifications for implementing wireless computer networking. The 802.11 is a family of standards like; 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, and more - ranging in frequency from 900MHz to 5GHz. Current Wi-Fi technologies face challenges like population density and transmission rates, and with today's Internet trends leading towards more consumption of high-quality video and audio content, this further increases the throughput needs, [42] Bellalta et al.

The population density in situations like train stations, sports events, and airports - can be a challenge to address when there is a need for high throughput. The traditional way of solving this is to increase the number of access points to accommodate, but unfortunately, this introduces interference issues, which increases packet errors and collisions, as well as preventing neighboring Wi-Fi networks from accessing the selected channel. With Wi-Fi deployments becoming more and more popular, coexisting networks in, e.g., an apartment building or an office building has to be taken into consideration. Deploying, optimizing and coordinates such environments can be very demanding.

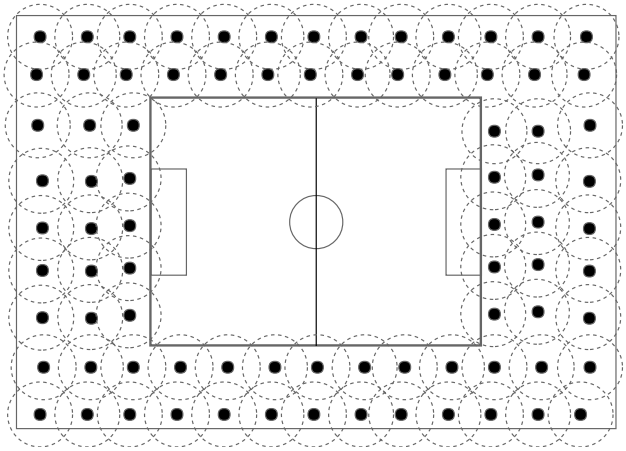


Figure 2.10: Visualization of Wi-Fi access point density in a football arena

Figure 2.10 display an ideal access-point deployment in a football arena, but these types of deployments are not always easy to accomplish and can be susceptible to interference from other business such as nearby coffee-shops, merchandise shops, internet of things equipment, and more. Below

is a list of the most current standards, aiming to address some of these issues.

802.11ac – 2013, was developed as an evolution to 802.11n-2009. 802.11ac adds 80MHz and 160MHz channel bandwidths, among throughput enhancements for multi-user capability with downlink multi-user MIMO (DL MU-MIMO). This differs from single-user MIMO found in 802.11n, which only allows a single multi-antenna receiver communicating with a single multi-antenna transmitter, and means that the access point can simultaneously transmit data streams to multiple clients. The most significant advantage of this is that a client with one antenna does not degrade the network capacity by occupying too much air time.

802.11ah – 2016, the future for wirelessly connected IoT (Internet of Things) devices. With the ever-increasing popularity of IoT devices operating on WLANs, collisions are starting to become a challenge. WLANs no longer exists as a commodity for a small number of high-speed devices; they now serve a role for connecting hundreds or even thousands of smart sensors, meters, cameras, and more. 802.11ah proposes a sub 1GHz (900 MHz) bandwidth used for IoT devices with ten times narrower channels than those in 802.11ac, [51], Khorov et al. This as a result of most IoT devices not needing high data rate connectivity. Aside from this, 802.11ah focuses on are; long-range communication, power saving mechanisms, and machine to machine communication.

802.11ax – 2019, is due to replace both 802.11n and 802.11ac by 2019, [42], Bellalta et al. This increment introduces a new media access control and physical layer to further improve on throughput and energy usage, as well as both downlink and up-link multi-user MIMO. One challenge with the previous generations is client density for physical space in scenarios like football stadiums, train stations, airports, and more. 802.11ax improves upon this and more by introducing features and concepts like:

- **Spatial reuse** – by; dynamic adaption of transmitting power, and beamforming.
- **Temporal Efficiency** – through control packets, efficient re-transmissions, simultaneous transmit and receive, and collision-free MAC protocols.
- **Spectrum sharing** – for improving unplanned deployments of WLANs which are causing problems with neighboring WLANs. The introduction of Dynamic Channel Bonding and OFDMA adds functionality for filling spectrum gaps and narrow channels.
- **Multiple antennas** – continuing the evolution of single and multi-user MIMO. 802.11ax might support Uplink MU-MIMO, Massive MIMO, and Network MIMO.

2.4 Summary

In this chapter, we started by introducing some relevant terms and methods used in this thesis. This included explaining that multi-link is a term for sharing network traffic over multiple links with the goal of increased connection stability and bandwidth aggregation. We also explained that HTTP is an application layer protocol circumventing the need for low-level operating system integration and that the features "Range Request" and "Pipelining" enable multi-link using HTTP. Lastly, we described some basic share calculations, with both the static and dynamic request-size approaches. The dynamic request-size approach is most suited in mobile environments.

Next, we presented some relevant work in the field of multi-link. We started by presenting some common challenges and what a new multi-link protocols main goal should be; connection reliability, bandwidth aggregation, and security. In this thesis, the main goal is first and foremost bandwidth aggregation and connection reliability using HTTP. Therefore, we presented work done with HTTP and multi-link and found the inspiration for the dynamic and static request size calculations. In the section on multi-link on smartphones, we presented work done with using application layer approaches. Most of the smartphone-related work was done on devices a couple of years old, and utilizing wireless networks with lower bandwidth capacity compared to today's technologies. This motivated us to look into a general purpose multi-link HTTP approach running on modern smartphones.

Further, we presented some work on energy usage in mobile devices, including tools for monitoring energy usage with both hardware and software. These findings showed that energy usage is closely related to the workload, not only the transfer size. They also conclude that with 3G, a significant fraction of the energy usage can be attributed to tail-energy – a prolonged state of high energy usage after the transfer is done. The presented monitoring tools for energy usage proved to be challenging to use on modern devices with newer Android versions not being compatible, or devices being glued together making hardware monitoring destructive.

Finally, in the technology section, we started by presenting Samsung Download Booster in detail. This function enables concurrent multi-link on Samsung's Android smartphones but has some limitations like; only working with certain services, and only utilizing 80-90% of the combined aggregated bandwidth. To gain some knowledge of HTTP/2, relevant improvements were described. HTTP/2 multiplexing addresses the HoL problem with pipelining, and header compression allows for a smaller header footprint compared to HTTP/1.1. This later described experiments in this thesis use HTTP/2. In the mobile communication networks subsection, we described the current and future state of Wi-Fi and LTE wireless networks. LTE carrier aggregation build up throughput over time and presents some challenges that have to be solved for the following approaches. Future networks like 5G and 802.11ax both promises increased bandwidth as well as reduced energy usage, removing some of

the challenges associated with carrier aggregation and more.

Chapter 3

Methodology

In this chapter, we present the methodology used to study the problem stated in section 1.2. This problem statement has two main objectives; the first is evaluating the performance of the approaches and finding out which input parameters works best for multi-link HTTP in a wireless mobile environment, and the second is how modern smartphones perform running the approaches. For this, we need to select metrics, parameters, an evaluation technique, and workload – this is presented in the first section. Next, we design the experiments by presenting the test setup including hardware and software, and then the scenarios in which the experiments are conducted. Finally, we explain the numbers and figures used in the results chapters that are to follow.

This chapter is inspired by some of the steps in the "Systematic approach to performance evaluation" – [57], Jain.

3. *Select metrics* – subsection 3.1.1
4. *List parameters* – subsection 3.1.2
6. *Select evaluation technique* – subsection 3.1.3
7. *Select workload* – subsection 3.1.4
8. *Design experiments* – section 3.2 and 3.3

3.1 Research methodology

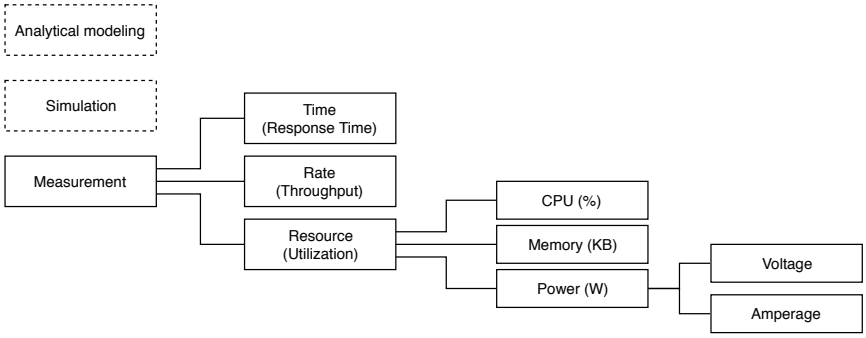


Figure 3.1: Figure visualizing the three main ways to evaluate performance in a system, with connected metrics.

This section starts by selecting metrics and parameters, step; "3. Select Metrics" and "4. Select Parameters". Next, we present "6. Select Evaluation Technique", and finally "7. Select Workload".

3.1.1 Selecting Metrics

Assuming that the system works correctly; "performance is measured by the time taken to perform the service, the rate at which the service is performed, and the resources consumed while performing the service" – [57]. These three metrics can be summed up to "Time", "Throughput", and "Resources" as seen in figure 3.1. In this thesis, the selected metrics are;

Throughput One of the most central questions regarding time and speed is throughput. How fast are the approaches able to download data and at which speeds? Under which circumstances can the approaches reach total aggregated throughput all the links. Can this occur under any circumstances? Are there overhead challenges related to this, making perfect throughput sum possible? Which algorithm is best suited, in terms of adaptability, for this? The selected value used for this metric are Mbps – Megabits per second.

Resources – CPU and memory These numbers are necessary because of how limited resources on a smartphone are. A multi-link approach running in the background consuming too much CPU and memory resources can lead to some situations where neither it or other processes are performing well. The selected values used for these metrics are for CPU; system-wide usage in percentages, and memory; application usage in KB (Kilobytes).

Resources – Energy usage Power integrated over time – energy usage is an important metric for smartphones since most are running on an internal battery. The selected values used for this metric are system-wide energy usage.

Response time was not selected as a metric, only as a input parameter, due to the fact that response time for this thesis would, for the most part, represent limitations outside of the test setup.

3.1.2 Selecting Parameters

The approaches use various input parameters, and the most important is;

Payload The HTTP URL for reaching the payload, and size for the start and end of the download. This parameter is essential for the upper limit calculations and the request sizes sent to the server.

Links The number of links, and which to use. All the multi-link approaches use LTE and Wi-Fi, and the single-link approaches use only one of them.

Throughput represents the current rate of received data. This input parameter is needed for calculating the best load share between the links. This parameter is continuously monitored during execution.

Bandwidth represents the maximum capacity of a network. In ideal situations, it is the same as max throughput. This parameter is the first factor of the Bandwidth Delay Product – section 4.2.3.

RTT is the time it takes from a request to be sent to the response is received. This parameter is the second factor of the Bandwidth Delay Product. This parameter is continuously monitored during execution.

3.1.3 Selecting an Evaluation Technique

Evaluation performance in computer system can be done in a multitude of ways and according to [57], there are three techniques for performance evaluation in computer systems; *Analytical modeling*, *Simulations*, and *Measurements*. When selecting an evaluation technique some considerations are;

Criteria	**Analytical Modeling**	**Simulation**	**Measurement**
Stage	Any	Any	Post-prototype
Time required	Small	Medium	Varies
Tools	Analysts	Computer languages	*Instrumentation*
Accuracy	Low	Moderate	Varies
Trade-off evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High
Saleability	Low	Medium	High

Table 3.1: Criteria for selecting an evaluation technique. Table copied from [57]

Table 3.1 shows that analytical modeling demands a level of simplifications and assumptions, making accurate results difficult. Analytical modeling works well if no simulations or prototypes exist for measurement.

Simulation can incorporate more details and make fewer assumptions, making it closer to reality. Simulating the unpredictable and unreliable nature of a Wi-Fi network, and an LTE network could become a challenge. Tools like 'netdelay' [7], and 'netem' [27], exists for general purpose network delay simulations, but integrating these tools to work on an simulated Android ARM-based operating system, with the Android security model [3], can be difficult. The Android project provides a *emulated* option, with the difference being that the whole computer stack including the hardware is "simulated" – compared to a standard simulation where, often, only the top-level layers are simulated. This device emulator does provide the option of further emulating the cellular connection with standards ranging from GSM, HSCSD, GPRS,... to HSDPA and LTE – with the option of setting discrete signal strength qualities. Using this option would be more reliable than the previously mentioned tools, but the emulator does not seem to have any options for emulating the Wi-Fi link, neither does it seem to support continuously changing signal qualities that would naturally occur in dynamic scenarios. Further, the fact that everything is emulated means that measuring energy-usage would probably result in data from a predefined model. These two factors combined make simulation not suitable for this thesis.

Measurements, while having to deal with unknown variables, can be best suited for a performance review on smartphones because energy measurements are challenging to simulate. Measurements can be both difficult and costly with varying accuracy; it does, however, results in real-life challenges not easily simulated.

Therefore, this thesis uses measurements with physical hardware connected to real-world wireless networks.

Monitoring According to [57], monitoring can be classified into four parts; software, hardware, firmware, and hybrid monitoring.

Software-based monitoring does not require knowledge into specific hardware schematics and can, therefore, be more readily available. Software monitors do consume system resources; therefore, the priority and sampling rate of software-based monitoring should be low enough that it does not affect the measurements.

Hardware-based monitoring does not affect system resources which makes it possible to increase the input rate, it handles a vast amount of data, has a separate hardware clock to provide time, can measure multiple events simultaneous with multiple probes, and can monitor the system even if it is malfunctioning.

Firmware-based monitoring required modifications to the processor microcode, and are therefore not very practical.

Hybrid-base monitoring are a combination of the classes mentioned above.

Some of the smartphones used in this thesis did not provide easy access to the battery and circuitry, making hardware-monitoring destructive to the devices. Because of this, we decided to use software-monitoring for all data

monitoring.

3.1.4 Selecting Workload

For measuring the effectiveness of the approaches, real-life workloads are needed to perform an evaluation. File downloads are an example workload, and another is video streaming. The evaluations are not affected by the contents of the workloads, the payload, only the size, and therefore file downloads were selected. During the initial testing with both single-link and multi-link application, the observed average carrier-aggregation buildup time was approximately 2.5 second, with some cases of significantly higher buildup times around 15-20 seconds. When downloading smaller file sizes in a dynamic multi-link setup, a Wi-Fi link with a significant bandwidth can download a large portion of the payload in under 2.5 seconds, resulting in the LTE link never reaching its potential. Eliminating this problem, large payloads was needed to make sure the LTE link gained its potential, and therefore, the selected workload was a file download of;

- 250MB random noise

Using a payload this large, even a Wi-Fi link with 500Mbps bandwidth would take about 3.6seconds downloading 90% of the payload, a good margin over the observed 2.5 seconds average seen for carrier-aggregation buildup. The Wi-Fi networks used in the scenarios averaged well below 500Mbps, but the file-size was kept large due to the observed occurrence of high buildup times around 15-20seconds. Under normal circumstances, a smaller file-size would be sufficient, but this thesis focus on throughput performance wanted to make sure the LTE connection was allowed to perform at max capacity.

Workload vs. Payload The term workload refers to the work that is assigned. In this thesis, the workload is the file download, and the term payload refers to the actual load being carried out – the 250MB file.

Execution order For the measurements, a randomization routine was used to prevent results from being affected by the time and capacity of the network. The execution script included configuring all the various approaches, and executing them in random and sequential order. Each execution downloads the payload, and to make sure that no approach was executed more than a certain threshold, a counter was implemented. This execution method was only used for the dynamic scenarios – section 3.3. In the isolated scenarios a standard sequential method was used.

Execution count The number of measurements, or the times the 250MB file were downloaded, was set as a default to 10 times for each scenario.

3.2 Test Setup

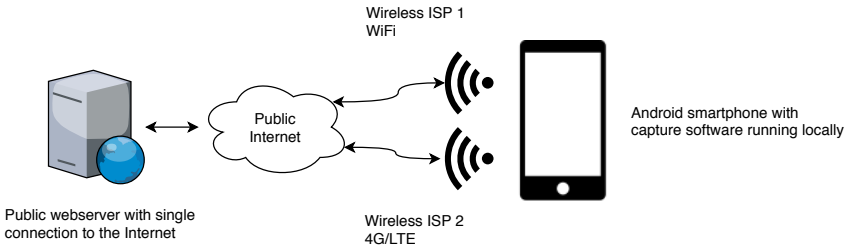


Figure 3.2: Visual representation of the main test setup

Defining the system in this thesis, many software and hardware components were used. Figure 3.2 visualizes the test setup using an Android smartphone with multiple links – Wi-Fi and LTE. This section starts by presenting the hardware, then the software.

3.2.1 Hardware

The list below describes the hardware used for the test setup;

A workstation acting as a client during the validation and simulations stages. The workstation used is a laptop running Arch Linux with kernel version 5.0 and configured with multiple network cards and varying bandwidths.

Server The test setup included a public server connected to the Simula darkroom, with a close physical approximation to the client. The server ran Arch Linux with kernel version 4.19, on an Intel NUC hardware.

PMIC chipset A Power Management Integrated Circuit is a small chip directly interfacing the battery of the device and controls power management. In some cases, this includes powering various LEDs, flashes, charging, and more. Smartphones equipped with a PMIC chip makes monitoring and capturing energy usage precise and accessible. The chips communicate by exposing information on a regular i2c [18] serial bus, and it is up to the device manufacturer to supply source code for parsing the protocol on this bus. The requirements for such a chip is that it can provide information with a high resolution and frequency. For accurate measurements, that means supplying voltage and amperage numbers with at least three decimals.

During researching hardware for use in this thesis, little to no information was found on these chipsets. Therefore, the easiest way was to gain root access for each smartphone and look into the specific hardware and drivers to verify if the smartphone were compatible with the requirements. Samsung equips most of its devices with

PMIC chip from maxim-integrated [26], and some of these chips expose information with a high degree of precision.

Samsung S6 released in April 2015, with LTE-A Cat6 300/50 Mbps capabilities. The S6 has good LineageOS support and excellent maxim chipset hardware. This maxim-chipset has an order of three times higher precision than its S4 and S5 counterparts, making it very suitable for software energy usage monitoring. The device also has a very capable LTE modem, which can reach throughput numbers similar to many Wi-Fi networks.

Samsung S7 released in March 2016 with an LTE-A (3CA) Cat9 450/50 Mbps modem. The Samsung S7 was one of the first devices used in early testing and performed very well during the initial tests and was therefore selected as the primary smartphone to use for measurements. This device has good LineageOS support, and excellent maxim chips on par with the S6, and even better LTE modem.

Smartphones excluded from final measurements. In the initial stages, several phones were flashed with LineageOS, rooted, and tested for compatibility. However, due to high precision PMIC requirements, these devices were excluded:

Samsung S4 version I9505 released April 2013 sporting LTE Cat3 100/50 Mbps. The device has good support for LineageOS and does not support LTE-A/Carrier Aggregation.

Samsung S5 released April 2014 with LTE-A Cat4 150/50 Mbps capabilities. This device has excellent support by the LineageOS community with Android versions surpassing the S6 and S7 devices. It is also, apart from the Samsung S4, the only device in the test setup with an easily removable battery making hardware monitoring accessible.

Nexus 5 is an older smartphone released in November of 2013 with an LTE Cat4 150/50 Mbps modem. The device is similarly capable as the Samsung S4 as it also does not support Carrier Aggregation – section 2.3.3.

Nexus 5X released October 2015 with an LTE-A Cat6 300/50 Mbps modem, making it comparable with the Samsung S6.

3.2.2 Software

In order to make the implementation run on an Android smartphone, the following software is used;

LineageOS is an alternative operating system (OS) for Android smartphones [63]. An alternate operating system was needed to control the environment in a way that the stock OS would not allow. LineageOS allows a more fine-grained control by limiting additional and unnecessary software running in the background. Such software could potentially affect the results by using system resources during the experiments. Another reason for selecting an alternative operating system

was to gain full administrative rights (also referred to as root access), making control over physical resources like LTE, Wi-Fi, and more possible. The implementation of multi-link used in this thesis was not possible on stock Samsung Android phones due to the access control. Replacing the stock operating system with an alternative, could in of itself introduce performance differences both positive and negative. Answering this question by comparing stock and alternative OS, are challenging and outside the scope of this thesis. LineageOS does not support every Android smartphone in existence, so the choice of operating system and smartphone device influenced each other. All the devices used in this thesis was installed with; *LineageOS version 14.1*.

Libcurl a library for making HTTP requests. The libcurl [21] project was released by Daniel Stenberg in April 1997 and has since become a well known and widely used library for the C/C++ family of programming languages. Some other libraries are used to support extended features. The library versions used in this thesis were;

- *Libcurl version 7.59.0* – as described above
- *zlib version 1.2.8* – library for compression functionality [38]
- *OpenSSL version 1.0.2d* – library for encryption functionality [29]
- *Libnghttp2 version 1.24.0* – library for HTTP/2 functionality [28]

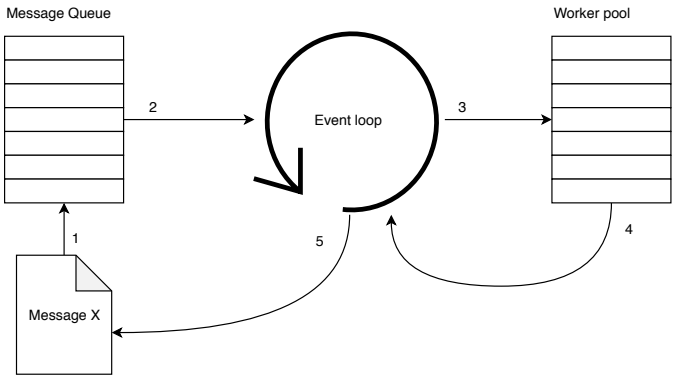


Figure 3.3: Illustrates a simple event loop

Libev Is an event-loop library for the C programming language [22]. An event-loop works by creating a queue for receiving messages, and a worker pool for processing the messages. As soon as a message is received, the event-loop offloads the work to an available thread in the worker pool, and when a message is done processing in the selected thread, a callback is sent to the sender of the message. This enables asynchronous non-blocking architecture on a single thread. Libev was selected as the event-loop library for handling the concurrent nature of the client application described later, as it integrates nicely with libcurl and many example resources were available. According to Libev, their implementation of an event-loop

performs better than other alternatives [23], and the library is fully-featured and high-performing. This thesis uses *Libev version 4.19*.

Web server The server side of the test setup is a web server, capable of serving static content over HTTP 1.1 and 2.0 with support for Range Request – section 2.1. It is also capable of handling TLS certificates as most client implementations of HTTP/2 demand that a valid TLS certificate is in place. The web server software "H2O [14]" was selected because it claims to outperform many similar products. Since all of the results in this thesis were generated using the HTTP/2 protocol, support for this was very important – which the H2O server has. Docker [10] was used to make testing and deployment easier. This thesis uses *H2O version v2.3.0-beta1*.

TCPDump For capturing the network transmission data at a low level, TCPDump was used on both the workstation and directly on Android smartphones. TCPDump creates captures which can easily be used later for visualizing and analyzing the network traffic data.

Capture and visualization for capturing resource metrics on Android smartphones, some bash scripts were created to read out and log system files during downloads. Matplotlib [25] and pandas [31] were used to generate statistics and graphs from the log files.

3.3 Experiments

This section describes the scenarios used for the experiments in this thesis. This thesis uses the term *scenario* for describing a test environment, consisting of a physical location with available wireless networks like LTE and Wi-Fi. The scenarios consisted of an isolated and dynamic scenario.

Why the different scenarios? Section 3.1 argues that measurements through real-life experiments are the best way to analyze the performance of multi-link in modern smartphones. The test setup must, therefore, include a real smartphone as well as a real Wi-Fi network and LTE network. Because of the unknown factors associated with wireless networks, an isolated scenario was created. The isolated scenario was created in an attempt to eliminate as many unknown factors, with the Wi-Fi network, as possible. Examples of such factors are; internet bandwidth capacity, network equipment quality, other devices connected to the network, and more. The dynamic scenarios, however, were created with a different motivation – by introducing unknown factors in unpredictable Wi-Fi networks. These two main categories of scenarios, the isolated and the dynamic, test a variety of environments and generates results for analytics.

Device selection The test bed in this thesis consist of a Samsung S6 and a Samsung S7 device. During initial testing, the devices displayed approximately the same pattern of throughput and resource usage. Not

equal, but the pattern was similar. Because of this, the Samsung S6 device was *only* included in the isolated high-speed scenario. The rest of the experiments were done with the Samsung S7 device.

Device setup During the experiments, the smartphones were charged at least 50% before each test. Further, because how the Android operating system enables deep sleep, called Doze [2], when the screen is off, all the experiments were done with the screen on set to the same brightness level. No devices were connected to external power sources like a battery charger during the experiments.

Approaches In the scenarios, different approaches are tested and measured. "Single-Link Wi-Fi" refers to an approach using *only* the Wi-Fi link to download the payload. "Single-Link LTE" refers to the same, but for the LTE link. The "Naive approach" is the first multi-link approach utilizing both the Wi-Fi link and the LTE link concurrently. "Algorithm 1, 2, and 3" are also multi-link approaches like the naive approach, but shares the load between the Wi-Fi link and LTE link more dynamically. All of these approaches are later described in detail in section 4.3.

3.3.0.1 Isolated scenarios

The isolated scenarios included a Wi-Fi network under full configuration and control. The Wi-Fi network used was an 802.11ac network, with an 80Mbps bandwidth to the Internet, and a router capable of throttling individual devices. The experiments done in this scenario did not suffer from other devices using bandwidth resources and were conducted with the device located at the same physical location for all tests. The various configurations in this scenario were:

High-Speed Wi-Fi The first scenario benchmarks the devices by measuring the performance of the individual links separately and then using multi-link. In this scenario the following approaches were measured with both the Samsung S6 and S7, with an execution count of 10 times each;

- Single-Link Wi-Fi
- Single-Link LTE
- Naive approach
- Algorithm 1, 2, and 3

Low-Speed Wi-Fi In the other isolated scenario, the internet bandwidth was restricted by ten times, from 80Mbps to 8Mbps, compared to the high-speed scenario. The physical Wi-Fi network remains the same. The Low-Speed, 8Mbps restricted Wi-Fi, scenario were selected to see how the algorithm would perform in a scenario where one network was significantly slower than the other. Doing this in an isolated scenario included throughput predictability. In this configuration the

following approaches were measured on the Samsung S7 only, with an execution count of 10 times each;

- Single-Link Wi-Fi
- Naive Approach
- Algorithm 1, 2 and 3

3.3.0.2 Dynamic scenarios

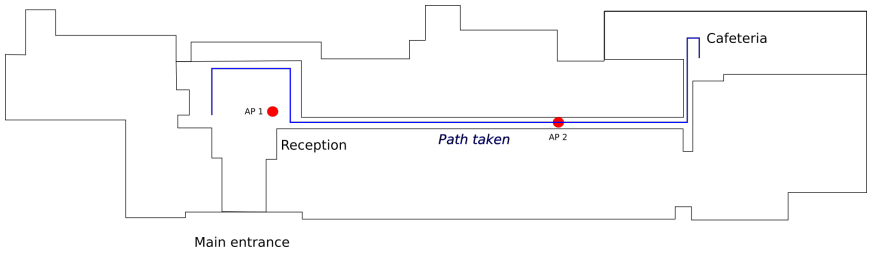


Figure 3.4: Simplified building schematic of Technopolis. Red dots are showing access points, and the blue path is visualizing the movements inside the building.

The dynamic scenarios consisted of two central locations and provided public Wi-Fi networks, introducing unpredictable throughput results. The dynamic scenarios were selected to test the adaptability of the algorithms, and testing in real-world scenarios. The dynamic scenarios were;

TechnoWLAN The first of the dynamic scenarios tested was Technopolis [34] TechnoWLAN Wi-Fi. This public Wi-Fi is available to anyone visiting the premises of IT Fornebu and was, therefore, a suited target for testing the approaches. During initial measurements, this Wi-Fi network had an average throughput of about 20Mbps, with LTE significantly higher. The TechnoWLAN measurements consisted of two parts; one stationary at the reception in a high traffic area, and one moving down a narrow hallway leading from the reception to the cafeteria. This hallway has a lot of construction material in every direction in addition to multiple access points, making handover a part of the moving measurements. Both the stationary and the moving scenario were conducted with a random execution order, section 3.1.4, and the execution count were set to 10 for each approach and each scenario. Only the Samsung S7 was measured with the following approaches;

- Single-Link Wi-Fi
- Single-Link LTE
- Naive approach
- Algorithm 1, 2, and 3

AlwaysOnline The other dynamic scenario used a Wi-Fi network in Celerway's [6] offices. During the experiments, no considerations were taken to make this network empty for testing. The measurements done with the AlwaysOnline network were similar in many ways to the TechnoWLAN measurements, but differed in one important aspect; the AlwaysOnline network consisted of only one access point. Only stationary experiments were performed in this scenario due to difficulties with the moving experiment. Only the Samsung S7 was used in this scenario. The approaches following approaches were executed 10 times for each, randomly;

- Single-Link Wi-Fi
- Single-Link LTE
- Naive approach
- Algorithm 1, 2, and 3

3.4 Presentation and Analysis

This section explains the numbers and figures presented in the result chapters 5, 6, and 7.

Figure 3.5: Figure visualizing the *median of medians*. Both the columns and the rows are sorted, and in this case from left to right, and from bottom to top ascending. The sort order of a median calculation does not matter. The red box in the middle represents the median-of-medians number in this example.

Median of medians The numbers presented in the results chapters are the median from a set of medians. Figure 3.5 represents a simple case of multiple data sets displayed vertically. These vertical data sets

are all sorted, and the blue boxes represent the individual data sets median number. Calculating the median-of-medians, all the individual median numbers are sorted and inserted into a new data set; the blue horizontal row. From this blue horizontal row, a new median number can be calculated – the red median-of-medians number. In terms of, e.g. throughput data, the calculations are the same, but instead, the vertical columns are individual measurements, and the boxes are sample data from the measurements – sorted by throughput. This thesis uses median-of-medians numbers because they better represent the resource usage, and also makes it easier to compare a noisy measurement with a clean one because small dips in the measurements do not affect the results as much as average calculations do. The noise can be an external factor outside the test scenario, but a side effect of this is that median numbers can masquerade noise that is due to internal factors. The alternative to using median-of-medians would be to use the average-of-averages.

Percentiles Used for representing a value below a given percentage. The 25th percentile is often called the first quartile, the 50th percentile is the same as the median value and often called the second quartile, and the 75th percentile is often called the third quartile. This thesis uses the 50th percentile for medians, as described above, and the 5th percentile (p05), and the 90th percentile (p90).

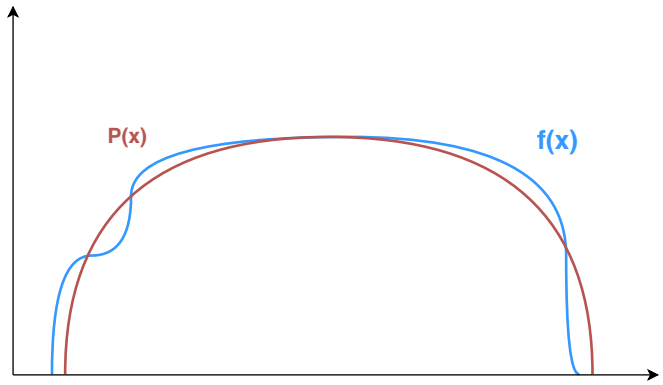


Figure 3.6: Figure visualizing Simpsons rule. F(x) is the actual data, and P(x) is an approximation by quadratic interpolation

Integrals The integral numbers are the combined factors of time and resource, and represents the most important data for comparison between the different configurations. The integral numbers are calculated by using the Simpsons Rule as visualized in figure 3.6. Simpsons rule are an numerical approximation approach to calculating definitive integrals. Scipy [32] was used for the actual calculations.

Standard Deviation In the results chapters, the mathematical notation of standard deviation is used throughout the results. This number represents

the amount of variation found in a data set. A low standard deviation number indicates that there is a little variation from the mean, and a high number indicates the opposite.

Coefficient of variation Notes the relative variation. Useful for analyzing if a variation is large or small relative to the data set. Coefficient of variation is calculated by;

$$cv = \frac{\sigma}{\overline{x}}$$

Correlation Coefficient In the analysis sections of the results chapters, a correlation coefficient is used to present the statistical relationship between to variables. The results are made of two categories; positive and negative correlation. Positive correlation occurs when increasing one variable lead to increasing another, and ranges between 0 and 1. Negative correlation occurs when increasing one variable decreases another, and ranges between 0 and -1. If a correlation calculation results in a 0-value, the variables have no relationship. If the number is 1, or -1, the correlation shows that there is a perfect linear relationship.

$$\rho_{x,y} = \frac{cov_{x,y}}{\sigma x \sigma y}$$

Throughput the analysis, the Pearson correlation coefficient is used. Pearson's is the covariance of the variables divided by the product of their standard deviation. Scipy [32] was used for the actual calculations.

Data rate over time *Bandwidth* is a term used to describe the maximum capacity of a network. *Throughput* is a term used to describe the actual data rate over time. *Goodput* is a term used to describe a subset of the throughput consisting of only useful traffic for the application. In this thesis, we present throughput in Mbps with two decimal places.

Because we present throughput numbers, the numbers and download times do not calculate to exact numbers for the payload size – as goodput would have. This is because the results include setup of the connection like TLS, initial handshake, as well as tear down. The throughput numbers are therefore slightly higher than the actual throughput for the payload. This small constant overhead is representative of all the throughput results presented. A concrete example if this is the numbers in single-link table for the Samsung S6 5.1a. In this table, the throughput for Wi-Fi is 79.01, but when dividing the payload by time, and then multiply by eight, the result is 76.74. For LTE the result is 37.75 compared to the 39.28 number in the table.

Download time Are presented in seconds with two decimal places. A complete download, from the user perspective, means that all the data has been downloaded and, therefore, the slowest link is representative for the time it takes to complete a download. Ideally, the links are finished with the least amount of Link Δ.

Link Δ The difference in seconds between the link completion time. This number represents the HoL or last segment problem – section 2.1

CPU numbers Are presented in system-wide percentages with two decimal places.

Memory numbers Are presented in application-specific usages as KB (KiloBytes), with two decimal places.

***Avg* and *now* energy-values** Are presented as Joule. Both smartphones exposed two values for measuring power levels, referred to as *avg*, and *now*. Power is a term used to describe the rate at which work is done. Energy is power integrated over time. The avg and now values are read directly from the Android kernel, which further reads them from the maxim chipset [26], more information about the max chipset can be found in section 3.2. The values represent a level at the time of measure (now), and an average level over the last couple of seconds (avg). The kernel does not expose power levels directly, only volt and ampere. The values used for power is a product of these factors, and are negative directly from the kernel. During the experiment, we measured both of these values, but later found that the *now* value was the most accurate representation of energy usage because the *avg* values did not react to rapid changes. The energy integral calculations used Simpsons rule – section 3.4.

Tail integral For energy-usage "tail-integral" is introduced. The tail-integral numbers represent the integral values of energy usage after the downloads. No head-integral are included because the relatively low sampling rate of 1Hz was not able to record any significant head energy values.

Figures The figures in the results chapters display measurements done with one of the approaches. For each figure, the individual lines represent single measurements. Each figure has ten individual lines.

- **Throughput figures** in chapter 5, display the downloads as individual lines where the x-axis represents time and the y-axis throughput. The color matching straight horizontal line is showing the median of medians number. In addition to this, the average download completion time is displayed in a color matching vertical line. For single-link measurements, only the blue color is used, but for multi-link LTE and Wi-Fi, both blue and orange is visible.

- **CPU and memory figures** in chapter 6, display CPU and memory usage, where the x-axis represent time, and the y-axis shows usage. A low polling frequency of 1Hz was selected during the measurements, due to the associated overhead of doing software measurement – section 3.1.3. Each measurement is represented with a separate color, and the blue horizontal lines represent a median of medians, the same as the other sections.

- **Energy usage figures** in chapter 7, is from the same measurement as previously presented, and the frequency used for the energy usage measurement was also, the same as for CPU and memory, at 1Hz. The x-axis in the energy graphs displays a more extended period than the actual download. The increase of time in the x-axis is to show the head and tail energy that occurs when starting and ending a download – section 2.2.4. The blue vertical lines represent, in order; start of the download, end of download median, end of tail energy median for the now-values, and finally end of tail energy median for the avg-values. The energy-metric are all negative from the source, and because of this, all the graphs have an inverted y-axis. In the accommodating tables, the negative sign in front of the values is removed making the data more accessible.

- **Analytical bar graph** in all the results chapters. Bar graphs are used to visualize the accommodating tables in the analysis sections. In chapter 5, thin gray bars are added in the center of a thicker bar. These gray bars represent individual download time for the Wi-Fi and LTE links. In chapter 6 and 7, thin black bars represents total download time and are used for comparison with the presented result.

Tables In the tables, some symbols and abbreviations are used;

- $\int$: representing integral calculations using Simpsons rule.
- $\uparrow$: representing the slowest link completion-time – section 3.1.1
- Δ : representing difference between the table row header and column header
- % : representing the row value compared to the column value in percentages
- σ or "Std.Dev": representing Standard Deviation.
- C_v : representing Coefficient of variation
- *p50, p90, p05* : representing percentile, e.g., p90 is the 90th percentile. p50 is the same as median.
- *S.L* : Abbreviation for Single-Link
- *H.S* : High-Speed

- *L.S* : Low-Speed
- *T.S* : TechnoWLAN Stationary
- *T.M* : TechnoWLAN Moving
- *A.O* : AlwaysOnline

3.5 Summary

In this chapter, we have presented the research methodology used in this thesis. We started by defining what metrics and parameters to use and then argued that measurements were the most suited evaluation technique to use for the stated problem. Next, we defined a suitable workload being a 250MB file download, the payload, due to the effects of carrier-aggregation.

In the test setup section, we presented the various hardware and software used in this thesis. This included some details into why certain Samsung smartphones were selected over others, because of the specific Power Management Integrated Circuit (PMIC) chipset. The reason why the phones were flashed with the alternative operating system LineageOS, was due to the need for controlling hardware access on an administrative level.

Next, we described the different scenarios for the experiments. These scenarios were selected to produce results from both an isolated environment, as well as a dynamic environment using public Wi-Fi's.

Finally, we presented and explained the numbers and figures used in the result chapters. We argued that the median-of-medians calculation better represented performance compared to the average-of-averages. We also explained why the throughput numbers presented in the result chapters does not calculate to precisely the time it takes to download the payload, due to TCP connection overhead and more.

Chapter 4

Proposed Algorithms and Implementations

In this chapter we present a description of the various approaches later referred to in the result chapters 5, 6, and 7. In the first section, we describe the motivation for the approaches and the associated implementation. Next, some background information is presented like whether to choose static or dynamic load sharing, the sequence of requests, and what the Bandwidth Delay Product is. Further, five approaches are presented. They consist of the default single-link approach, the naive multi-link approach, and the proposed multi-link Algorithm 1, 2, and 3.

Finally, three implementations are presented for supporting the five approaches. The two first implementations cover the single-link approach and the naive approach. The last implementation covers Algorithm 1, 2, and 3. After having presented the implementations, some details surrounding terms and challenges are described.

In this chapter, some algorithms and math are presented. In these figures, certain symbols are used;

n : number of elements in a set, e.g; number of links the algorithm utilizes

x : specific element in a set, e.g $link_x$

$\sum_{i=1}^{n}$: sum of all the elements in a set, e.g total throughput for all the links

BDP : Bandwidth Delay Product : section 4.2.3

4.1 Motivation

Why propose algorithms for load sharing between links? In this chapter three algorithms are proposed, explained, and later implementation is described. The motivation for the proposed algorithms is to answer the first research question; which input parameters are important for multi-link HTTP in a mobile environment. A custom implementation is necessary

to answer this question and also due to the lack of existing applications for testing multi-link HTTP on modern smartphones with a focus on dynamic adaptability in wireless networks. Samsung, on their devices, has a function called *Download Booster* but testing has shown that it has some limitations – section 2.3.1. The proposed algorithms in this chapter are inspired by the research in section 2.2, and are iterations addressing issues observed during testing and development.

4.2 Background

This section covers some background associated with load sharing, including the sequence of requests, and the Bandwidth Delay Product.

4.2.1 Static or Dynamic Load Sharing

When selecting a load-sharing algorithm for multi-link purposes, there are usually two primary ways of solving it, the static or dynamic approach. Static-approaches can work well in a controlled environment, where the links are expected to perform the same, as a local server-to-server setup.

Static-approaches tend to divide the load of the links with an equal share, or deterministic share of the payload based on the expected throughput. The load split can be anything from a static 1/n share to a more tuned share – where n is the number of links. It is difficult to determine the speed of a wireless link beforehand with Wi-Fi networks changing based on the physical location of a smartphone, and some networks, like LTE, also has its own set of challenges like Carrier Aggregation – section 2.3.3. Static-approaches can also result in an amplification of the HoL (Head of Line) problem because it could assign too large shares on slow links, and consequently, result in bad user experience – section 2.1.

Dynamic-approaches are usually more suited for dynamic environments with wireless networks rapidly changing bandwidth and signal strength.

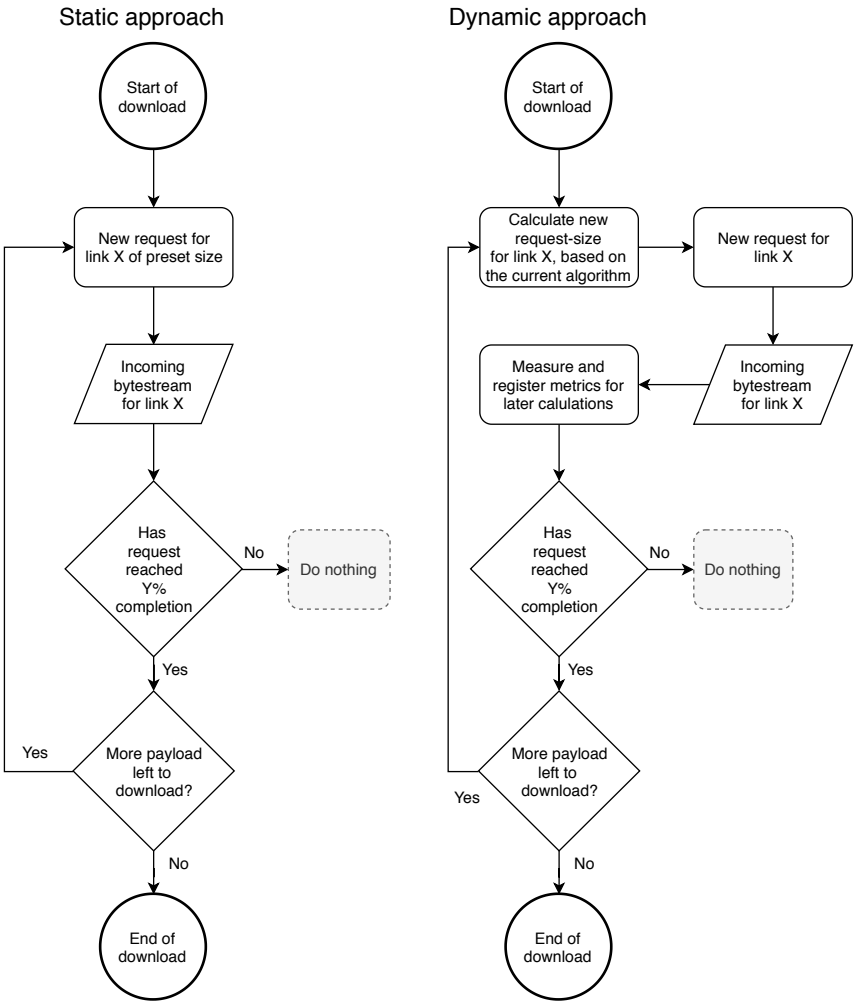


Figure 4.1: Flowchart displaying the static and the dynamic approaches

Figure 4.1 shows that the main difference between the static approach and a dynamic approach is; the dynamic measures and recalculates the size of the requests, where the static does not.

The static approach starts by making a request for a specific size. This static size can be anything from a certain amount of MB, or as the naive approach used in this thesis, a function of the payload size – payload size divided by link count. Dividing the payload size by the link count bypasses the last decision element "More payload left to download", because both links have been assigned half the payload – in a two-link setup.

The dynamic approach works differently, by continuously monitoring and registering input parameters for each link. The continuously measured input parameters are;

- Throughput for each link
- Round Trip Time (RTT) for each link

These data are then used for calculating request sizes, making sure that none of the links are overloaded or starved. Algorithm 1, 2, and 3,

described in section 4.3.4, are all dynamic approaches, and the differences are the amount of data collected in the measurement step, and how the request calculating is done.

4.2.2 The sequence of requests

Dividing a larger payload into smaller Range Requests (section 2.1), immediately introduces a challenge with the sequence of the requests. HTTP pipelining enables concurrent requests; however, selecting when to start a new request is up to the implementation/algorithm. It is not ideal for letting the requests completely download before starting a new one, as this creates a gap in the pipeline. The algorithm must, therefore, have the next request ready and started before the previous is done, with some amount of overlap. By continually measuring the progress of a request, the algorithm can detect and start a new concurrent request before the previous is done. Figure 4.1 displays this in the "Has request reached Y% completion" decision element, where the percent can be set to a suitable overlap based on the request size. This means that each link can have multiple requests in-flight at the same time. The amount of concurrent request is determined by when new requests spawn off the ongoing, and if previous requests have slowed down for some reason.

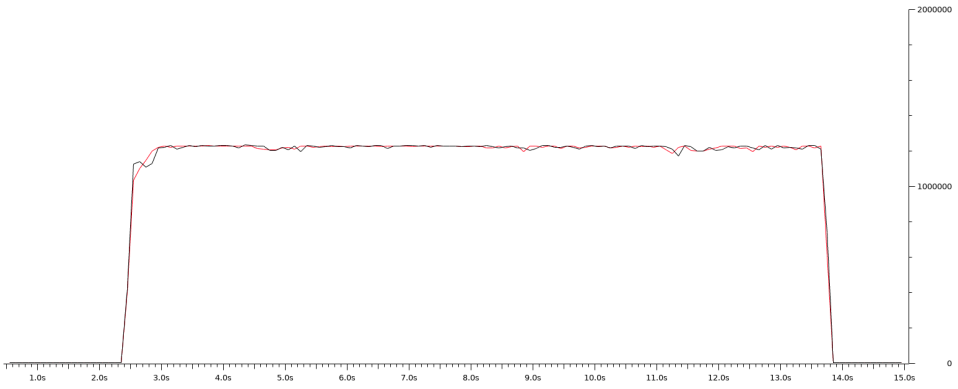


Figure 4.2: Figure visualizing throughput over time for two links using 60% completion-percentage.

This thesis decided to use the static value 60% for all the experiments, as early testing showed that this was a safe value with no significant drops in throughput as visualized in figure 4.2.

4.2.3 The Bandwidth Delay Product

Networks that are offering high bandwidth with very high delay are often called long fat networks (LFN). Examples of LFNs are satellite connections, and in some cases, LTE networks. One problem with LFN-networks is that the client has to wait a long time for an acknowledgment from the server before it can send the next data. During this time, nothing happens, and the connection is not fully utilized. A countermeasure for this is to calculate

the Bandwidth-Delay Product and make sure this is the minimum of data in the pipe.

$$link_x BDP = link_x Bandwidth * link_x RTT$$

This product indicates a minimum request size, but it loses some of the dynamic factors because, for specific networks like LTE, it can be hard to predict bandwidth. The BDP is used for Algorithm 3 presented later in this chapter.

4.3 The Five Approaches

This section presents five approaches for downloading a payload. First, a default single-link approach, and a naive multi-link approach. Next, three multi-link algorithms (1, 2, and 3) are proposed and presented. Each of the algorithms is an increment and improvement over the next one, utilizing more of the available input parameters. The first Algorithm 1 calculates request-sizes with a static upper limit, preventing the requests becoming too large. Algorithm 2 addresses this upper limit, by replacing the static value with a dynamically calculated value. Algorithm 3 improves upon Algorithm 2 by introducing a lower request size limit by utilizing the BDP calculation – section 4.2.3.

The motivation behind the single-link and naive approach is to compare the differences between the algorithms later. The motivation with presenting three algorithms is to test them in various scenarios to prove that the improvements affect results.

4.3.1 Single-Link – The default approach

The default approach is a single-link approach and is a way of downloading a payload over one single link, like Wi-Fi or LTE. In the result chapters, this approach, referred to as *single-link*, is presented as a benchmark for comparing various metrics against the multi-link approaches.

4.3.2 Naive approach

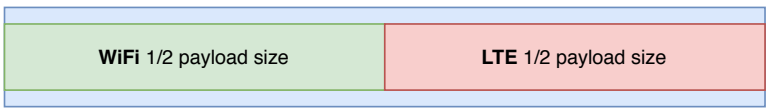


Figure 4.3: Figure displaying how the naive approach can split a payload into two equal loads, one for each link in a two-link setup like Wi-Fi and LTE.

The first multi-link approach is the naive-approach. Without knowing any information about the links, a naive approach is the simplest way of sharing data between links by dividing the payload into equal request sizes - like

a 50:50 share displayed in figure 4.3. This approach has its faults in that it is not able to adjust for weaknesses or differences between the links. If one links throughput is double another, the naive approach is suboptimal because the slower link would finish much later than the fastest. This approach is used for multi-link comparisons in the result chapters 5, 6, and 7.

4.3.3 Algorithm 1 – Static upper limit

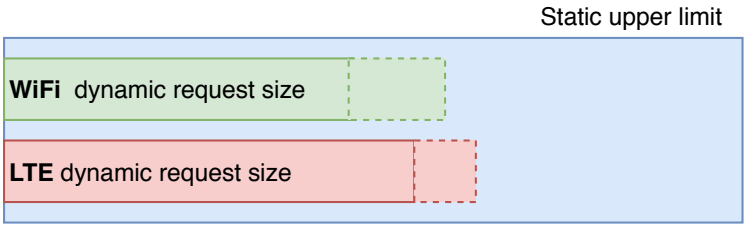


Figure 4.4: Figure displaying blue static upper limits, and green and red dynamically calculated requests for each link. The request-size calculation is done for every new request and is based on the current throughput of both links combined.

Algorithm 1 is the most basic of the algorithms. It starts by measuring a share for each link (between 0 and 1), where the link throughput is the current average throughput for one link, and total throughput is the combined average throughput of all the links and uses this share to calculate requests. The throughput is measured continuously like displayed in algorithm flow figure 4.1, and the current average throughput is calculated based on these measurements. Figure 4.4 displays the dynamic request load sharing, where the static upper limits are preventing the requests from growing too large.

$$link_x Share = \frac{link_x Throughput}{\sum_{i=1}^{n} link_i Throughput}$$

Figure 4.5: Link share calculation, for a specific link-x, are the product of current link throughput ($link_x Throughput$), and the sum throughput for all links

The request-size for a link is then calculated by multiplying the link share with the static upper limit.

$$link_x RequestSize = staticUpperLimit * link_x Share$$

The product of this multiplication is the request-size, in bytes, for the current link. Prevent one link from dominating others; a limiting factor

can be implemented preventing the share from becoming too low or too high. Algorithm 1 shortcomings are the missing ability into the lower end of throughput. It does not necessarily fill the pipeline of the links with high bandwidth and high round-trip time. It also does not allow high performing links to excel with more significant requests.

Algorithm 1 – Example 1

Let us say; link1 has an average throughput of 1600KBps, link2 has an average throughput of 800KBps, and the upper limit is 4000KB.

$$link_1 share = \frac{1600}{1600 + 800} = 0.666$$

$$link_2 share = \frac{800}{1600 + 800} = 0.333$$

$$link_1 RequestSize = 4000 * 0.666 = 2666KB$$
$$link_2 RequestSize = 4000 * 0.333 = 1333KB$$

Figure 4.6: Example calculation of request sizes using the Algorithm 1

4.3.4 Algorithm 2 – Dynamic upper limit

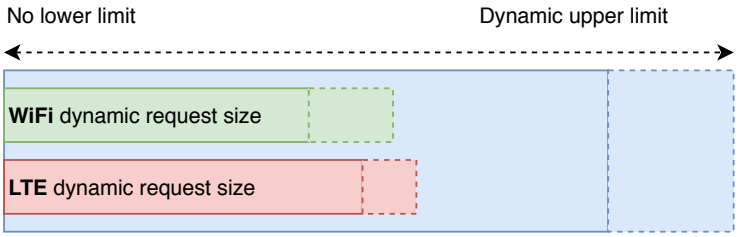


Figure 4.7: Visualizing dynamic blue upper limits and green and red dynamic requests for each link. Both the upper limits and the requests are dynamic because they are recalculated in-flight.

Building on the previous algorithm and addressing its challenges, Algorithm 2 recalculates the upper limit in flight, and are therefore more flexible in situations where one or more links perform well. Figure 4.7 displays this, where the algorithm adjusts the upper limit according to the continuous throughput measurements.

$$dynamicUpperLimit = \frac{payloadSize}{n} * \frac{\sum_{i=1}^{n} link_i Throughput}{\sum_{j=1}^{n} link_j Bandwidth}$$

Figure 4.8: Dynamic upper link calculation. n is the number of links, the fraction divides the sum throughput with the sum bandwidth for all links

The result of this is a dynamic upper limit, based on the combined current average throughput and combined bandwidth of the links. Initially, the bandwidth sum value can be the best guess, but as soon as the download progress has reached a certain point, throughput peaks/max can be used as a replacement. With two links, the total dynamic request size can reach half the payload size, and that is not ideal if the payload size is large. A limiting factor can be used to keep this from happening.

The request size is calculated by;

$$link_x RequestSize = dynamicUpperLimit * \frac{link_x Throughput}{\sum_{i=1}^{n} link_i Throughput}$$

Figure 4.9: Request size product for a specific link x. The sum function in the denominator is calulates to the sum throughput for all links, where n are the number of links

This is similar to Algorithm 1, but instead of a static upper limit, it has a dynamic upper limit.

Algorithm 2 – Example 1

Given the following values;

- $payloadSize$ = 250,000KB, n (numberOfLinks) = 2
- $sum_{j=1}^{n} link_j Bandwidth$ (sum bandwidth) = 60,000KBps
- $link_1$ avg. throughput = 1600KBps
- $link_2$ avg. throughput = 800KBps

, the calculation is;

$$dynamicUpperLimit = \frac{250,000}{2} * \frac{1600 + 800}{60,000} = 5000KB$$

$$link_1 RequestSize = 5000 * \frac{1600}{1600 + 800} = 3333KB$$

$$link_2 RequestSize = 5000 * \frac{800}{1600 + 800} = 1666KB$$

Algorithm 2 – Example 2

Now lets say $link_1$ has increased throughput significantly, and $link_1$ now has a current average throughput of 8000KBps but $link_2$ average throughput is unchanged.

$$dynamicUpperLimit = \frac{250,000}{2} * \frac{8000 + 800}{60,000} = 18333KB$$

$$link_1RequestSize = 18333 * \frac{8000}{8000 + 800} = 16666KB$$

$$link_2RequestSize = 18333 * \frac{800}{8000 + 800} = 1666KB$$

This example shows that the faster link does not influence the slow $link_2$.

4.3.5 Algorithm 3 – Dynamic upper and lower limit

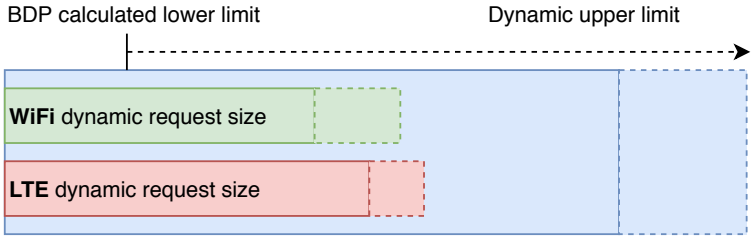


Figure 4.10: Visualizes how Algorithm 3 uses the same dynamic upper limit as Algorithm 2, but introduces a BDP calculation for the lower limit.

The last algorithm is Algorithm 3, the same as Algorithm 2, section 4.3.4, but utilizing the Bandwidth Delay Factor – section 4.2.3. It does this by using the BDP as a lower limit for the request sizes because the bandwidth delay product represents the minimum amount of data to be requested for keeping the pipe filled – visualized in figure 4.10.

Algorithm 3 starts by calculating dynamic upper limits, and dynamic request sizes, like Algorithm 2, then use this result and checks that it is not less than the bandwidth delay product.

$$link_xRequestSize = \begin{cases} link_xBDP & \text{if } link_xRequestSize < link_xBDP \\ link_xRequestSize & \text{otherwise} \end{cases}$$

Example

Given the following values;

- $payloadSize$ = 250,000KB, n (numberOfLinks) = 2
- $sum_{j=1}^{n} link_j Bandwidth$ (sum bandwidth) = 60,000KBps
- $link_1$ avg. throughput = 1600KBps, RTT = 20ms
- $link_2$ avg. throughput = 400KBps, RTT = 40ms

, results in the calculation;

$$dynamicUpperLimit = \frac{250,000}{2} * \frac{1600 + 400}{60,000} = 4166$$

$$link_1 RequestSize = 4166 * \frac{1600}{2000} = 3333KB$$

$$link_1 BDP = 30,000 * 0.02 = 600KB$$

$$link_2 RequestSize = 4166 * \frac{400}{2000} = 833KB$$

$$link_2 BDP = 30,000 * 0.04 = 1200KB$$

Since the calculated request size for $link_2$ is less than the BDP, the request size is adjusted up accordingly.

$$link_2 RequestSize = link_2 BDP$$

When throughput and bandwidth measures the same, or very similar Using the bandwidth delay product as a lower limit ensures the pipe is always filled with a minimum amount of data. However, when the links are at max capacity, the algorithm reduces the request size to 1/n, where n is the number of links. This can create large requests, similar to the naive approach, which makes volatility in wireless networks a challenge. A modification of the algorithm is to check the difference of the calculated request size and use the difference as padding for the link bandwidth delay product.

$$link_x RequestSize = \begin{cases} link_x BDP & \text{if } link_x RequestSize < link_x BDP \\ diffPad() + link_x BDP & \text{otherwise} \end{cases}$$

Where *diffPad()* refers to a simple function, like:

Algorithm Link BDP diff padding

```
1: function BDPDIFFPADDING(requestSize, BDP, staticFactor)
2:     diff ← requestSize − BDP
3:     diffPad ← diff ∗ staticFactor
4:     return diffPad
5: end function
```

Here the multiplying *staticFactor* can be any number between 0 and 1 and acts as amplification for the padding. Ideally, this factor can be dynamic and adjusted based on the link history throughput stability. With an optimistic approach to this, a link which has a history of stable throughput has a higher probability of maintaining stability.

Visually this can be displayed like;

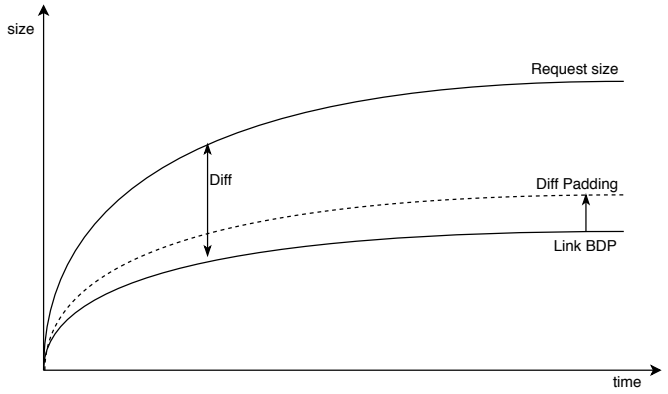


Figure 4.11: Displaying the diff padding between link BDP and request size

Example of padding Let say; $link_1$ has averaged a throughput very close to the bandwidth, and therefore ended up with a large $link_x RequestSize$ of 10000KB. Further, lets say the BDP for $link_x$ is 2000KB and the $staticFactor$ is set to 0.3.

$$diff = link_x RequestSize - link_x BDP = 9000 - 2000 = 7000$$
$$diffPad = diff * staticFactor = 7000 * 0.3 = 2100$$

$$link_x RequestSize = diffPad + BDP = 2000 + 2100 = 4100$$

4.3.6 Algorithms comparisons

Comparing the various algorithms, the use of input parameters are; link count, payload size, throughput, bandwidth, and RTT.

	Link Count	Payload Size	Throughput	Bandwidth	RTT
Default approach	1				
Naive approach	>1	✓			
Algorithm 1	>1	✓	✓		
Algorithm 2	>1	✓	✓	✓	
Algorithm 3	>1	✓	✓	✓	✓

Table 4.1: Comparing input parameters among the different approaches

Table 4.1 shows which algorithms are using what input parameters. The bandwidth delay product can be essential in LTE networks because during

carrier aggregation buildup the measured throughput is not representative of the max throughput – or bandwidth. If not handled correctly, the LTE link can become starved and not utilizing its capacity, or reaching its capacity.

4.4 The Three Implementations

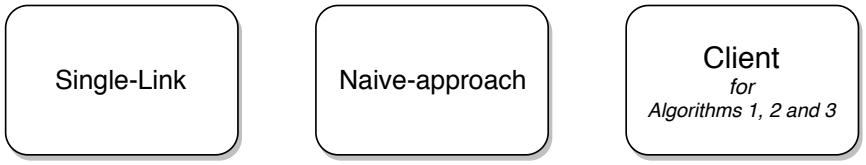


Figure 4.12: The three main implementations used in this thesis

This section covers the basics of how the algorithms, in section 4.3, were implemented, including the single-link application and the naive approach application. Figure 4.12 displays these three implementations. The last implementation, the client application, is described in detail including some challenges like; throughput monitoring, and multiple TCP ports.

4.4.1 Single-Link – A simple application

Comparing the results with an application using only one link, the Curl CLI [9] application is a common choice, and readily available on LineageOS. However, for the fairest comparisons between the three implementations, a small libcurl-driven application using the same libraries as the others were created – the libraries are described in section 3.2.2. This was important to eliminate library version differences and features, as well as possible compiler optimizations with more – which can affect the resources usage and performance. An example of this is the lack of HTTP/2 support with the provided Curl CLI application found on Lineage operating systems [63].

4.4.2 Naive approach – The equal load sharing application

The naive-approach application is very similar to the single-link applications, in that it is a small application doing a straightforward task. The naive approach application extends the single-link application by splitting the payload equally over multiple links and concurrently downloading the requests with range requests.

4.4.3 Client – The configurable applications for all the algorithms

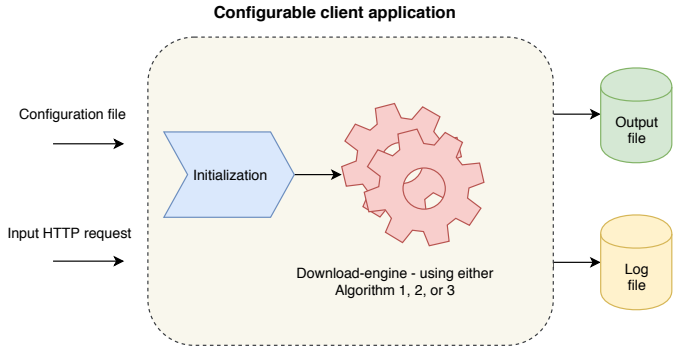


Figure 4.13: Figure visualizing basics of the client application

The configurable client application for all the algorithms is an event-loop driven application probing the throughput metrics during runtime. It works by inputting a configuration file and listening to a socket for URLs. The configuration file specifies which links to be used, and various other static settings. The socket is a traditional UNIX socket. When the client application receives a request for a new download, through the socket, it starts by initializing and configuring the links and connection pools. As soon as this is done, the first requests are loaded up, and the request is sent to the server. Typically the first requests are shared 50/50 between the links, but as soon as the first byte from either request is received from the server, the measuring and logging processes take place. When a new request is due for download, the measurement data are utilized for calculating a new request size. The basis of this process is visualized in figure 4.1.

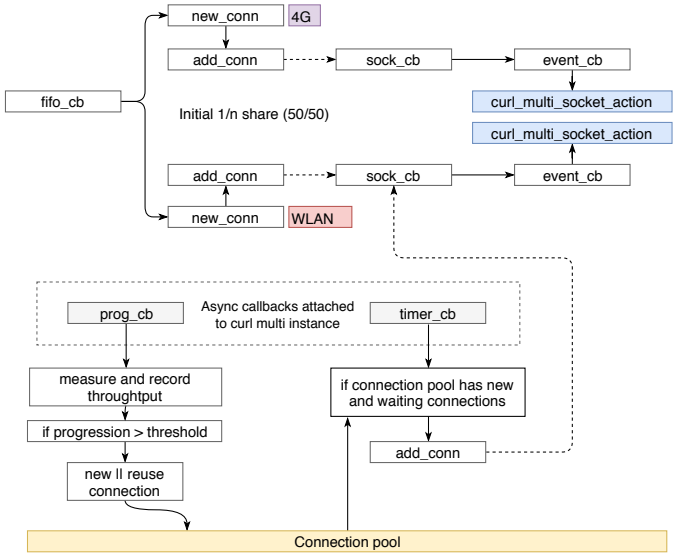


Figure 4.14: Figure visualizing the internal schematics of the client application engine

Due to the concurrent nature of the client application, an event loop library Libev was used – section 3.2.2. Figure 4.14 shows how data flows through the client applications. It starts by listening URLs from the socket called *fifo_cb*. As soon as the socket retrieves a URL, it creates a small request attached to a connection handle to each of the links, called *4G* and *WLAN*. Then, two processes start up and send the handle further to the *curl_multi_socket_action* function, which sends of the HTTP request to the remote server. As soon as the data stream starts to come back from the server, both *prog_cb* (progress call back), and *timer_cb* (timer callback) are processes that are involved in measuring and starting new connections.

When the client was implemented, all the libraries was cross-compiled to the standard Android Armv7 CPU architecture, making it executable directly on Android smartphones.

Further development into this client was done to make sure it could handle all the needed mechanisms to integrate and utilize the algorithms. Implementing a multilink application without wasting resources can be difficult; therefore much work went into profiling and analyzing weak points in the application with tools like Valgrind [36] and Callgrind [4].

4.4.3.1 Libcurl handles

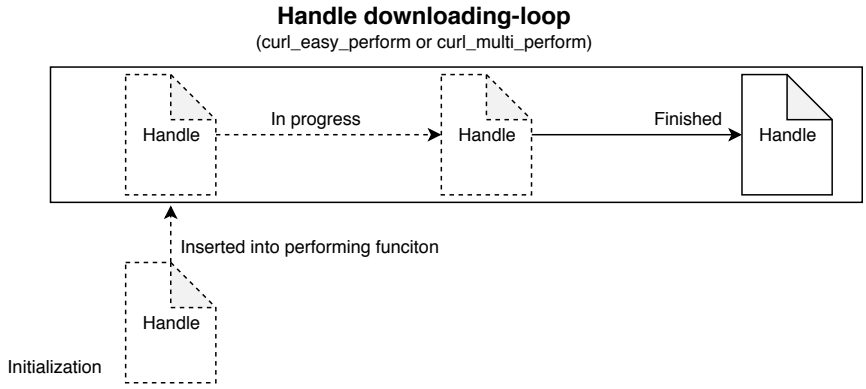


Figure 4.15: Figure visualizing the libcurl handle lifecycle

Libcurl is structured by exposing *handles* via an API. A *handle* represent a request for downloading or uploading data from a client to a server.

Some examples of settings which goes into a handle are;

- Device link (Wi-Fi, LTE...)
- Remote HTTP URL
- Range request (from byte x to y)
- Enable pipelining
- SSL verification
- Progress function for receiving information during download

- Receiving datastream function

Figure 4.15 visualizes how handles are inserted into a handle-downloader, typically 'curl_easy_perform' [1] or 'curl_multi_perform' [2]. When throughput and bandwidth measurement are done during runtime, the limitations of the handle-downloader needs to take into consideration. Functions exist for extracting average throughput for a handle, but doing this when a handle is done does not provide all necessary information. Instead, by continuously monitoring this data during a download of a handle, a local data set can be created providing more detailed information. Inside this data set, there also exist max and min numbers which can be used for bandwidth estimation.

4.4.3.2 Throughput measurement in flight

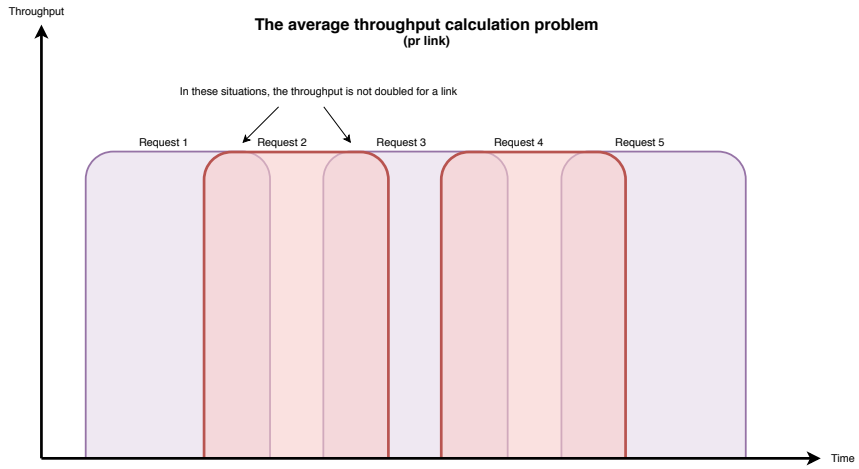


Figure 4.16: Figure visualizing the throughput measurement with concurrent request/handles problem

Section 4.2.2, describes how multiple links and multiple requests can be in flight concurrently. Measuring throughput and bandwidth in such situations can be challenging. First, HTTP belongs to the application layer, therefore, low-level metrics are not readily available during runtime. Next, the limitations of the HTTP library has to be taken into consideration.

When multiple *handles*, described in section 4.4.3.1, connected to a link is in-flight concurrently, it represents a challenge that the average throughput measurements for a handle do not at all times represent the average throughput number for a link. Instead, the sum of all handles in-flight, related to a link, provides this information.

Figure 4.16 visualizes this problem when the handles or requests, overlap. If at the time, the throughput of a link is 100Mbps, the throughput

[1] Curl easy perform – https://curl.haxx.se/libcurl/c/curl_easy_perform.html
[2] Curl multi perform – https://curl.haxx.se/libcurl/c/libcurl-multi.html

share between the handles is not apparent. The effect is most likely to cause the throughput of the existing handle to decrease, as the new overlapping one starts to build up? This results in throughput curves for a handle looking more like a bell curve than straight blocks like in the figure above.

This also means that bandwidth information must be calculated in the same way – by summing all the peaks in all handles in-flight and connected to a link.

4.4.3.3 New TCP connection vulnerability

LTE links can be susceptible to TCP slow buildup times and carrier aggregations challenges. Testing shows that when LTE links start a new TCP connection or adopt another's link TCP connection, the results are that carrier aggregation has to build up anew.

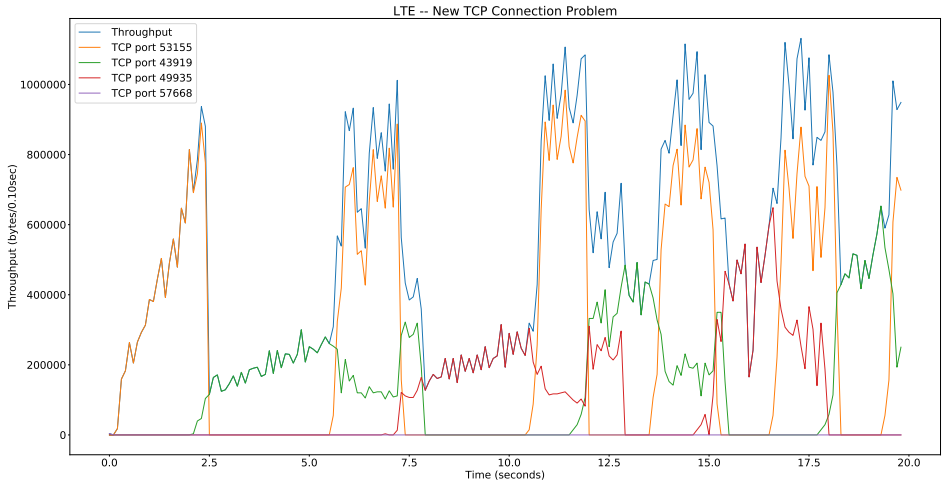


Figure 4.17: Figure displaying LTE with carrier aggregation, and the "new tcp connection problem". Each line, expect the blue, represents a TCP connection and in this example four connection were created for a single LTE link.

Figure 4.17, visualizes a single-link LTE download using range requests. The first range request, displayed with the orange TCP connection 53155, has successfully built up a high throughput over the first 2.5 seconds. The next range request starts at around the 2-second mark because the first request has reached Y% completion. This new range request, green connection 43919, has to build up throughput all over again because it starts a new TCP connection.

The second green connection never reaches the throughput of the original orange connection, but when the orange connection is reassigned a new request around 5.5 seconds, it continues with the same throughput where it left off. This proves that the overall throughput of a download can be significantly affected if multiple TCP connections are used, compared to if the same TCP connection is used throughout. Further experiments

show the Wi-Fi link was not affected to the same degree as the LTE link by this, mostly because it does not experience throughput buildup by carrier aggregation.

Making sure that both LTE and Wi-Fi did not adopt each others TCP connection, from a global pool, and making sure that HTTP pipelining was enabled correctly, corrected this issue.

4.5 Summary

In this chapter, we have presented a motivation for why there is a need for proposed algorithms and custom implementation. The existing solutions for concurrent multi-link today is the Samsung's Download Booster, which in section 2.3.1 was explained and found lacking in some areas. This existing solution was not able to answer which input parameters are needed for multi-link HTTP in mobile environments. Therefore, we presented three proposed algorithms.

Next, we explained the differences between static and dynamic approaches, in the background section. The main takeaway from this was that static approaches were only suitable in isolated and controlled environments like a local server-to-server scenario. The background section also covered why we used 60% overlap in the sequence of request. This value was selected due to early testing validating that it was a save value. We also covered what the Bandwidth Delay Product (BDP) is, and its importance in long fat networks like LTE. The BDP represents a minimum request size to keep the pipe filled.

Further, all the approaches were presented including the default single-link and the naive approach. The algorithms are iteration upon each other using an increasing amount of input parameters. Each algorithm was presented with example calculations, and Algorithm 3 proved to be the most advanced algorithm using all of the input parameters.

Finally, the three implementations were presented enabling the approaches mentioned above, one for the single-link approach, one for the naive-approach, and lastly one for Algorithm 1, 2, and 3. The section also included some challenges and what were done to address them, like how LTE is vulnerable to new TCP connections.

Chapter 5

Results: Throughput

This chapter, and the following result chapters 6 and 7, presents the results from the experiments described in section 3.3. The experiments are configured to compare the multi-link approaches with single-link, as well as against each other. The motivation for this is; answering the research questions by finding out which input parameters are needed for utilizing most the available bandwidth, thereby which approach performs the best in regards to download time. Next, to find out how the devices are affected by measuring the CPU, memory, and energy usage during the experiments.

This chapter presents the aggregated bandwidth performance of the approaches by measuring throughput. This is achieved by using Wi-Fi and LTE alone in a single-link approach, then the various algorithms and the naive approach is tested to compare how efficient it is in respect to download times and load sharing in two main scenarios – the isolated and dynamic. The isolated scenario consists of a high-speed and low-speed Wi-Fi network, where the differences are the available bandwidth to the internet – not the actual wireless network. These two scenarios are isolated because there are no other devices on the Wi-Fi network consuming resources during testing. The dynamic scenarios consisted of two public Wi-Fi's tested in different configurations. The first public Wi-Fi, the TechnoWLAN, was measured with the device both stationary and moving, and the second, the AlwaysOnline, only stationary. These dynamic scenarios were selected to test the approaches in a scenario with unpredictable capacity, bandwidth, and signal strength.

Terminology The terms used in this chapter are;

- *Mbps* : Megabit per second. The default throughput value.
- *Seconds*: The default download time value.
- *Slowest* ↑ : The slowest link download time.
- *S.L* : Abbreviation for Single-Link
- *rmnet0* : Represents the LTE link, used in the figures.
- *wlan0* : Represents the Wi-Fi link, used in the figures.

- Δ : Difference, used for Link Δ
- σ : Standard deviation.

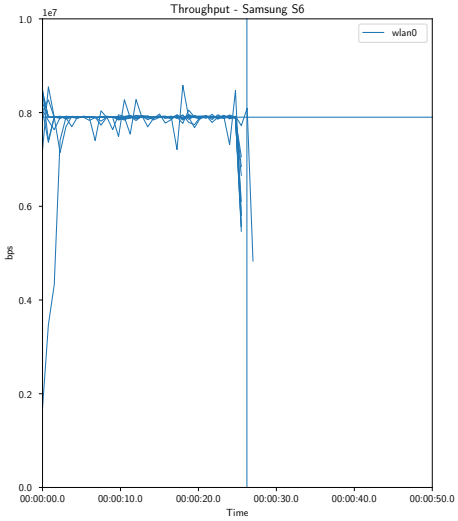
Further description in section 3.4.

5.1 Isolated Scenarios

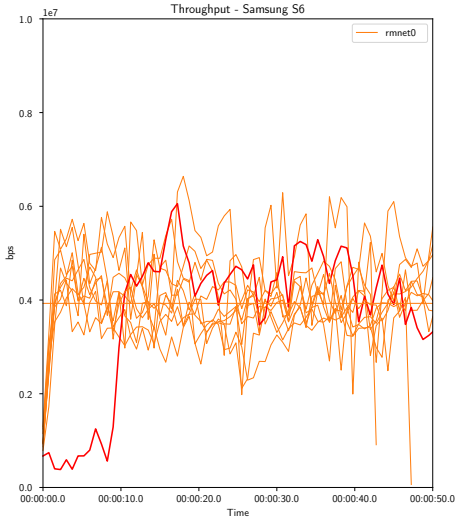
The isolated scenarios are divided into a High-Speed Wi-Fi and a Low-Speed Wi-Fi scenario. The difference between these two scenarios and the later dynamic scenarios are the isolation of the Wi-Fi network – section 3.3.0.1. This means the Wi-Fi network used in the isolated scenario was at full capacity without other devices consuming resources.

5.1.1 High-Speed Wi-Fi

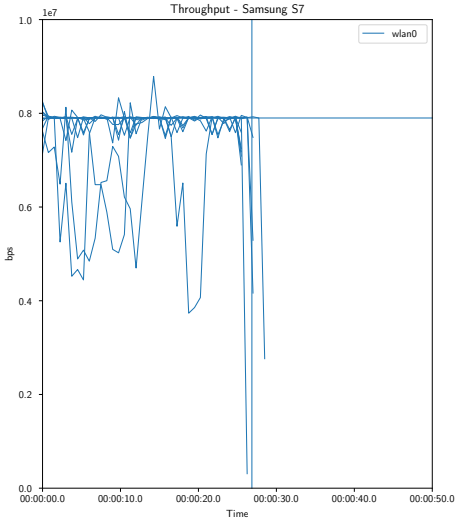
The single-link throughput results presents the device performance for the Wi-Fi and LTE separately. The figures visualize the downloads by plotting throughput over time as individual lines, described in section 3.4. The horizontal line is showing the calculated throughput median of the individual medians, and the average download time is displayed in a vertical line. The figures below 5.1a, 5.1b, 5.1c, and 5.1d visualizes throughput over time using the single-link approach. The figures 5.2a and 5.2b, 5.2c, and 5.2d are presenting multi-link results, and only includes the Samsung S7 device, the Samsung S6 results are included in the tables after.



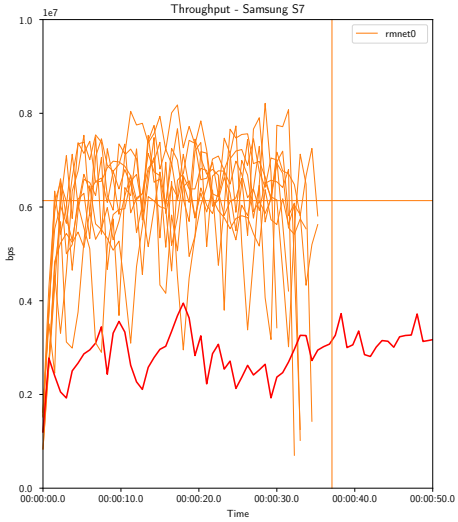
(a) Wi-Fi – Samsung S6



(b) LTE – Samsung S6

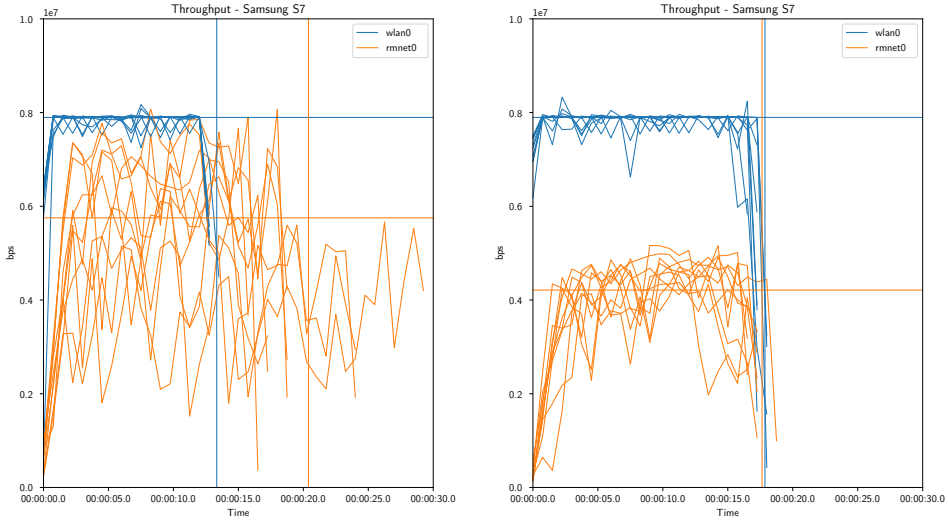


(c) Wi-Fi – Samsung S7



(d) LTE – Samsung S7

Figure 5.1: Single-Link throughput results using Wi-Fi and LTE with the Samsung S6 and the Samsung S7. The individual lines are single measurements, and the straight horizontal lines shows median throughput. The straight vertical line displays average download time



(a) Naive approach

(b) Algorithm 1

(c) Algorithm 2

(d) Algorithm 3

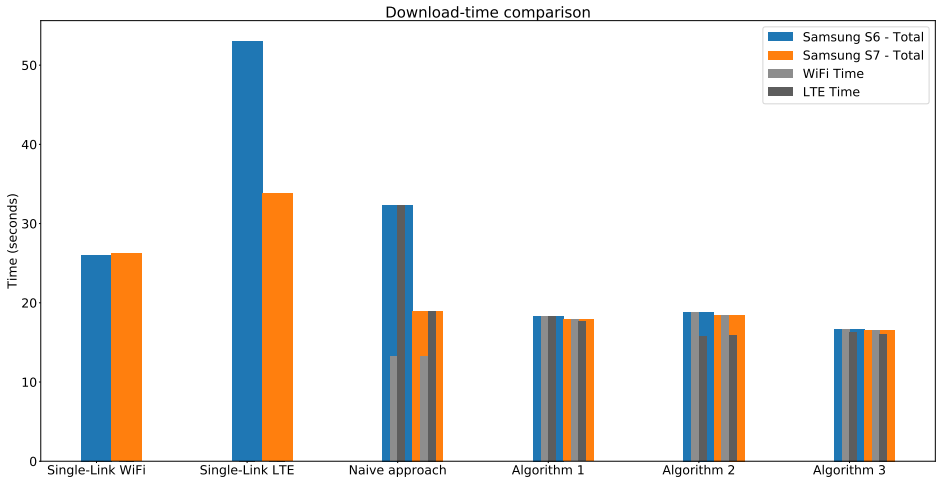
Figure 5.2: Multi-link throughput results from the Samsung S7. The blue lines display individual measurements using the Wi-Fi link, and the orange displays for the LTE link. The horizontal straight lines display median throughput for each link, and the vertical average download time.

	Time (s)			Throughput		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	26.06	–	–	79.01	–	0.02	–
Single-Link LTE	–	52.97	–	–	39.28	–	3.85
Naive approach	13.19	32.36	19.17	79.01	33.79	0.05	7.26
Algorithm 1	18.34	18.35	0.01	79.01	36.29	0.03	9.09
Algorithm 2	18.86	15.77	3.09	78.98	41.36	24.93	8.03
Algorithm 3	*16.68*	16.29	0.39	78.98	49.81	0.05	10.92

(a) Samsung S6

	Time (s)			Throughput		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	26.25	–	–	78.99	–	0.74	–
Single-Link LTE	–	33.79	–	–	61.37	–	11.31
Naive approach	13.27	18.99	5.72	78.96	57.53	0.13	11.55
Algorithm 1	17.88	17.67	0.21	78.90	42.10	0.03	2.73
Algorithm 2	18.44	15.89	2.55	78.90	47.85	0.43	5.93
Algorithm 3	*16.55*	16.05	0.50	78.97	51.60	0.05	4.34

(b) Samsung S7



(c) Samsung S6 and S7

Table 5.1: High-Speed results form the Samsung S6 and the Samsung S7. The download time values are presented in seconds, and both the throughput and standard deviation numbers are presented in Mbps.

Single-Link The data from tables 5.1a, 5.1b, and 5.1c, as well as figures 5.1a, 5.1b, 5.1c, and 5.1d show that a complete download using the single-link Wi-Fi approach, takes on about 26 seconds for both devices. The single-link LTE approach is varying between 52 seconds for the Samsung S6, and 33 seconds for the Samsung S7. The differences in LTE download times can be attributed to fluctuations on the LTE network. The Samsung S7 is per specification more capable of gaining throughput over LTE with a carrier aggregation number of three, compared to the two for the Samsung S6

device – section 2.3.3. However, carrier aggregation cannot be guaranteed in all situations and is dependent on the available channels in the nearby network. The *red* outlier in the LTE figure 5.1d for the Samsung S7 displays this. Another example is the red slow buildup outlier in figure 5.1b, where it takes about 10 seconds before the device gained carrier-aggregation.

Multi-Link The load sharing capability of the naive approach is shown in figure 5.2a for the Samsung S7, and with a Link Δ with over 5 seconds, the approach is not optimal – table 5.1b. The Link Δ represents the last-segment or *Head of Line* problem described in section 2.1. Even worse is the Link Δ for the Samsung S6, with over 19 seconds – table 5.1a. Algorithm 1 improves upon the naive approach by decreasing the time difference down to under 1 second for both devices. The algorithm shares the load significantly better and improving the download time from high twenty seconds to well under twenty seconds. Algorithm 2 fails to share the load between the links, and the result is a link delta between 2 and 3 seconds. Algorithm 3 solves this, which leads not only to a low link delta but also the best download time.

	Time ↑	S.L Wi-Fi	S.L LTE
S.L WiFi	26.06	–	–
S.L LTE	52.97	–	–
N.A	32.36	124.17%	61.09%
Alg.1	18.35	70.41%	34.64%
Alg.2	18.86	72.37%	35.60%
Alg.3	*16.68*	64.00%	31.48%

(a) Samsung S6

	Time ↑	S.L Wi-Fi	S.L LTE
S.L W-Fi	26.25	–	–
S.L LTE	33.79	–	–
N.A	18.99	72.34%	56.20%
Alg.1	17.88	68.11%	52.91%
Alg.2	18.44	70.24%	54.57%
Alg.3	*16.55*	63.04%	48.97%

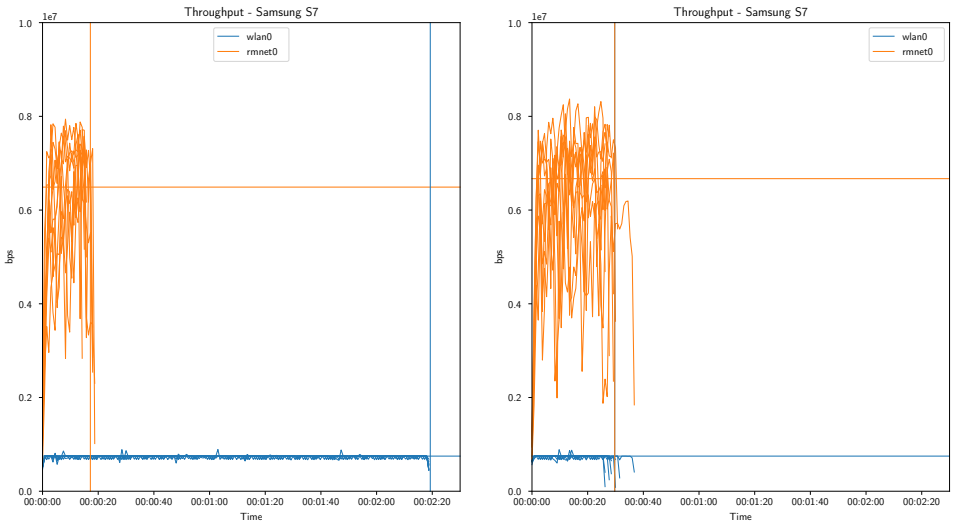
(b) Samsung S7

Table 5.2: Download time comparison, from the High-Speed isolated scenario, in seconds for the Samsung S6 and S7. The S.L Wi-Fi and S.L LTE columns shows what the row-name uses compared to the column-name. A value below 100 means the row-name uses less time compared to the column-name.

Compared The tables 5.2a and 5.2b, are based on calculations from table 5.1a and 5.1b. They show that almost all the multi-link approaches improve upon the single-link approaches quite significantly for both devices. They further show that the multi-link algorithms have a significant influence on overall download times, approximating about half the download time compared to single-link LTE, and about two-thirds compared to the single-link Wi-Fi. The only multi-link approach using more time to download than a single-link is the naive-approach for the Samsung S6. In this case, the naive approach uses 124.17% more time, or 6.3 seconds slower to download the payload compared to single-link Wi-Fi.

5.1.2 Low-Speed Wi-Fi

In the previous high-speed isolated scenario, the internet bandwidth of the Wi-Fi network and the LTE network was comparable with a ratio of about 8:5ths in favor of the Wi-Fi network. Making this difference larger, the internet bandwidth of the Wi-Fi network was restricted by ten times; to 8Mbps – section 3.3.0.1. Single-Link LTE was not measured in this scenario, because of the similarity between the high-speed and low-speed setup. Therefore, in the following comparisons, the LTE results from the High-Speed scenario is used. Only the Samsung S7 was used in this scenario because of how the two devices performed equally in terms of throughput – section 3.3. The figures below 5.3a and 5.3b visualizes throughput over time for the naive approach and Algorithm 3.



(a) Naive approach

(b) Algorithm 3

Figure 5.3: Throughput results from the Low-Speed Wi-Fi scenario using the Samsung S7 device. In these figures the blue throughput Wi-Fi lines are significantly less (and thereby slower) than the orange LTE lines.

	Time (s)			**Throughput**		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	278.18	–	–	7.49	–	0.00	–
H.S S7 Single-Link LTE	–	*33.79*	–	–	*61.37*	–	*11.31*
Naive approach	139.26	17.34	121.92	7.49	64.90	0.00	6.58
Algorithm 1	29.63	29.22	0.41	7.49	65.24	0.00	7.39
Algorithm 2	*29.18*	28.49	0.69	7.49	66.66	0.00	4.62
Algorithm 3	29.01	29.33	0.32	7.49	66.70	0.00	6.44

Table 5.3: Results from the Low-Speed Wi-Fi measurements with the Samsung S7. The High-Speed Single-Link LTE results is inserted making it easier to compare.

In the high-speed isolated scenario, the naive approach had a "Link

Δ" (difference in completion times between the links) of approximately 12 seconds more than Algorithm 1,2 and 3 – table 5.1b. In the low-speed table 5.3 this difference grew significantly, to approximately 120 seconds. This stands as an example of how the naive approach can suffer if the network bandwidth differs significantly. This increase in link delta represents a multiplication of 10 times, which is to be expected due to the fact that internet bandwidth of the Wi-Fi network was divided by the same amount. All the algorithms handled the increased difference between the links and shared the load accordingly with a link delta well below one second. The difference between single-link Wi-Fi and multi-link (excluding the naive approach) in this scenario is an improved download time of approximately 250 seconds.

Compared The numbers in the low-speed table 5.3, shows that when adding a slow Wi-Fi link to a high performing LTE link in a multi-link setup, the effect is not very significant. The results in the table show the download times decreasing from 33.79 seconds for single-link LTE, to around 29.4 seconds for Algorithm 1, 2, and 3. This difference is a decrease in download times of 85%.

LTE throughput Calculating the average LTE throughput, between the multi-link approaches in table 5.3, results in average throughput of 65.87Mbps. In the high-speed table 5.1b, for the Samsung S7, the same calculated LTE throughput average is; 49.77Mbps. This difference makes the low-speed LTE throughput 132.34% faster. The high-speed and low-speed scenarios were not measured on the same day, leading to a possible explanation of network capacity at the time of measurement. Another possible explanation could be that the Samsung S7 device has limited capabilities, but early testing showed numbers above the results presented. However, the single-link LTE results from the high-speed scenario are used in comparison with the low-speed multi-link approaches because of similar LTE throughput values (61.37Mbps versus approximately 66Mbps).

5.1.3 Analysis

For both devices, Algorithm 3 performed the best in the high-speed isolated scenario, and only slightly worse than Algorithm 2 in the low-speed scenario. The tables below are based on the results from table 5.1a, 5.1b and 5.3, and focus on download time and compare the multi-link approaches.

	Time ↑	Naive A.	Alg.1	Alg.2	Alg.3	*Average*
Naive approach	32.36	–	175.80%	171.05%	193.41%	*180.09*
Algorithm 1	18.35	56.71%	–	97.30%	110.01%	*88.01*
Algorithm 2	18.86	58.28%	102.78%	–	113.07%	*91.38*
Algorithm 3	16.68	51.55%	90.90%	88.44%	–	*76.96*
Average		*55.51*	*123.16*	*118.93*	*138.83*	–

(a) High-Speed – Samsung S6

	Time ↑	Naive A.	Alg.1	Alg.2	Alg.3	*Average*
Naive approach	18.99	–	106.21%	102.98%	114.74%	*107.98*
Algorithm 1	17.88	94.15%	–	96.96%	108.04%	*99.72*
Algorithm 2	18.44	97.10%	103.13%	–	111.42%	*103.88*
Algorithm 3	16.55	87.15%	92.56%	89.75%	–	*89.82*
Average		*92.80*	*100.63*	*96.56*	*111.40*	–

(b) High-Speed – Samsung S7

	Time ↑	Naive A.	Alg.1	Alg.2	Alg.3	*Average*
Naive approach	139.26	–	470.00%	477.24%	474.80%	*474.01*
Algorithm 1	29.63	21.28%	–	101.54%	101.02%	*74.61*
Algorithm 2	29.18	20.95%	98.48%	–	99.49%	*72.97*
Algorithm 3	29.33	21.06%	98.99%	100.51%	–	*73.52*
Average		*21.10*	*222.49*	*226.43*	*225.10*	–

(c) Low-Speed – Samsung S7

Table 5.4: Download time comparison for the multi-link approach, in percentages for both devices.

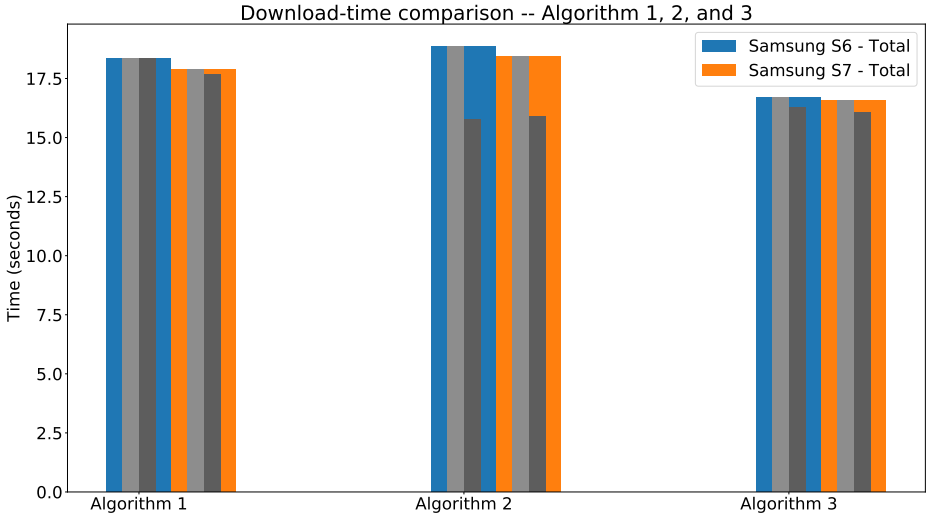


Figure 5.4: Download time comparison, from the high-speed isolated scenario, between Algorithm 1,2 and 3, and for both devices. This figure is the same as 5.1c but with focus on the proposed algorithms only. The thin light-gray and dark-gray columns represent Wi-Fi download time and LTE download time, respectively.

The high-speed tables 5.4a and 5.4b shows that Algorithm 3 outperforms the other multi-link approaches by using on average 76% and 89% of the time to download for the Samsung S6 and S7 respectively. For the Samsung S6, all the algorithms used on average 55.51% of the download time compared with the naive approach. The differences in download time between the algorithms and the naive approach average 92.80% for the Samsung S7.

For the low-speed results in table 5.4c, Algorithm 1, 2, and 3 differ only slightly, with Algorithm 2 performing best on average. Comparing the scenarios, the low-speed scenario gained the most when comparing the naive approach with Algorithms 1, 2, and 3, with the difference being the algorithms used on average 21.1% of the time to download.

Why Figure 5.4 visualizes and compares the algorithms 1, 2, and 3 in the high-speed isolated scenario. Equal-size thin gray columns translate to a better link-share than if the height of the column is far apart. The figure shows that for both devices, Algorithm 2 fails to share the load between Wi-Fi and LTE by approximately two seconds, leaving the LTE link unused by the end of the download. The reason for the poor link-sharing capabilities of Algorithm 2 is probably assigning too large request-sizes to the Wi-Fi link, and not filling the pipeline of the LTE link. Algorithm 2, as described in section 4.3.4, has no calculations for lower request sizes, only upper. Algorithm 3 addresses this issue and outperforms both Algorithm 1 and 2. The reason Algorithm 1 fails to gain high throughput is probably due to the lack of dynamic upper limits. High latency network like LTE needs larger requests to fill the pipeline. This difference makes Algorithm 1 use a couple of more seconds to download compared to Algorithm 3.

In the low-speed scenario, all the algorithms performed almost equal with a difference of under one percent from the average – calculated 73.6% from table 5.4c. Compared to the differences observed in the high-speed scenario, this is almost insignificant. In the low-speed scenario, all the approaches were throttled by the Wi-Fi router providing access to the internet. This effect is visible in table 5.3, where the throughput results are exactly 7.49Mbps and, the standard deviation is 0.00 – for all the results. The static nature of the Wi-Fi network probably suits Algorithm 1 and 2 due to the fact that their calculations are more suited for static environments. Algorithm 3 did therefore not perform badly in this scenario; it was Algorithm 1 and 2 that excelled making the difference smaller.

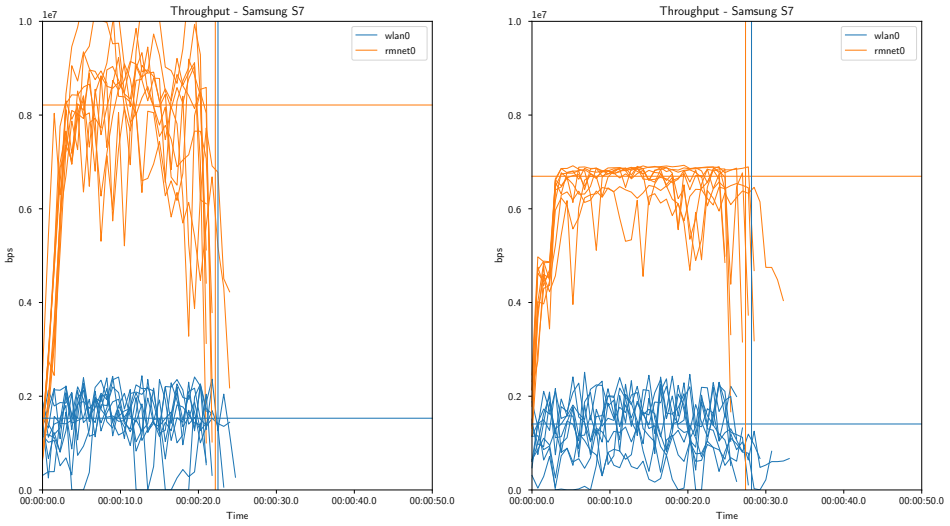
5.2 Dynamic Scenarios

In the previous section, the isolated scenario, the Wi-Fi network was isolated leading to predictable throughput numbers. In this section, more volatile wireless signal qualities and unpredictable throughput numbers are presented in the two Wi-Fi networks; TechnoWLAN and AlwaysOnline – section 3.3.0.2. The Wi-Fi networks are publicly accessible, or guest networks, with unknown capacity or availability. The results from these

scenarios were captured using a randomization setup – section 3.1.4. This randomization script executed the different approaches in an unpredictable order, enabling all the measurements for all the approaches to be executed sequentially. Only the Samsung S7 was used in these scenarios, because of the similar performance of the two devices – section 3.3.

5.2.1 TechnoWLAN

The first dynamic scenario tested was with Technopolis [34] Wi-Fi network. This network is available to anyone visiting the premises of IT Fornebu – section 3.3.0.2.



(a) Stationary – Algorithm 3

(b) Moving – Algorithm 3

Figure 5.5: Throughput results from the TechnoWLAN scenario using the Samsung S7 device. The blue lines represent the Wi-Fi link, and the orange the LTE link.

	Time (s)			Throughput		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	135.05	–	–	15.27	–	2.11	–
Single-Link LTE	–	24.86	–	–	80.14	–	6.90
Naive approach	65.59	14.25	51.34	15.43	77.62	0.77	19.31
Algorithm 1	32.91	32.09	0.82	15.27	49.14	1.48	11.67
Algorithm 2	26.06	21.77	4.29	14.73	76.28	3.35	11.33
Algorithm 3	*22.51*	21.67	0.84	15.30	82.14	1.82	6.50

(a) TechnoWLAN – Stationary

	Time (s)			Throughput		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	131.59	–	–	16.59	–	6.76	
Single-Link LTE	–	33.32	–	–	67.30	–	1.30
Naive approach	71.78	16.54	55.24	14.34	68.13	5.10	8.43
Algorithm 1	29.16	28.41	0.75	14.76	64.96	4.30	11.68
Algorithm 2	29.07	26.50	2.57	13.89	66.77	36.07	2.62
Algorithm 3	*27.29*	26.60	0.69	14.07	66.94	3.78	2.40

(b) TechnoWLAN – Moving

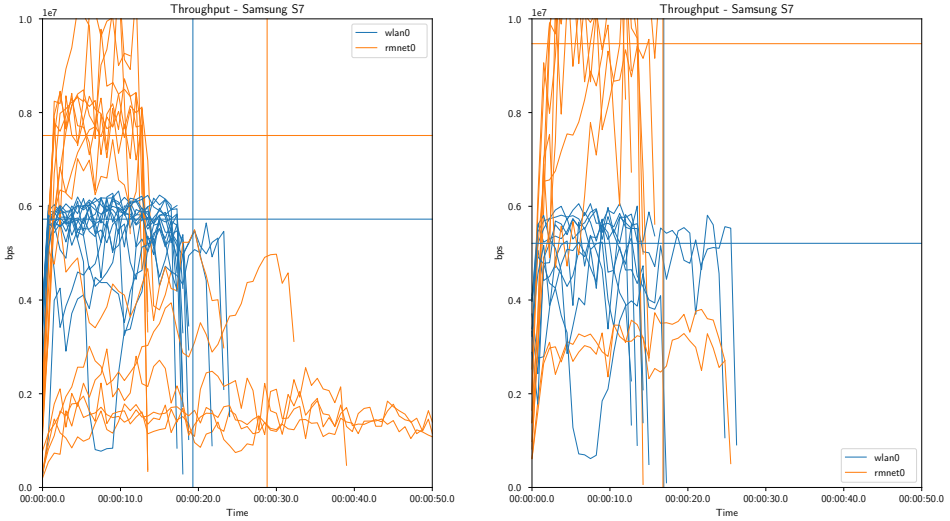
Table 5.5: Throughput results from the TechnoWLAN. The download time values in this table are presented in seconds, and the throughput and standard deviation are presented in Mbps.

Figure 5.5a and 5.5b displays the throughput results of Algorithm 3, and table 5.5a and 5.5b shows the results from the stationary and moving measurements. The Wi-Fi throughput numbers were consistent throughout both the stationary and moving measurements, even considering access point handover in this Wi-Fi network. LTE, on the other hand, differed significantly between stationary and moving measurements, with the stationary results having higher LTE throughput numbers. This is probably due to the construction matter surrounding the hallway and affecting signal quality. However, with the higher LTE throughput numbers in the stationary results, more volatile measurements are observed. This is confirmed by the figures, and in the standard deviation numbers in the tables.

The Wi-Fi throughput makes multi-link in this scenario questionable. For the stationary results, the multi-link download times decrease by 3 seconds compared to single-link LTE. The difference is approximately 6 seconds for the moving results.

5.2.2 AlwaysOnline

In the next dynamic scenario, a Wi-Fi network installed in Celerway's offices was used – section 3.3.0.2. Only the Samsung S7 device was used for these tests.



(a) Naive approach

(b) Algorithm 3

Figure 5.6: Throughput results from the AlwaysOnline scenario using the Samsung S7 device. The blue lines represent the Wi-Fi link, and the orange the LTE link.

	Time (s)			**Throughput**		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
Single-Link Wi-Fi	37.24	–	–	57.28	–	1.13	
Single-Link LTE	–	28.16	–	–	70.44	–	22.66
Naive approach	18.43	13.84	4.59	57.25	75.11	4.80	29.34
Algorithm 1	21.76	21.50	0.26	55.01	41.68	5.73	21.53
Algorithm 2	17.93	13.36	4.57	54.96	96.99	12.71	34.53
Algorithm 3	14.85	*14.94*	0.09	52.10	94.70	6.07	30.85

Table 5.6: Results from the **AlwaysOnline** Wi-Fi access point at Celerways offices. The device used in these measurements was the Samsung S7, and it was placed in a stationary position inside the offices.

The table 5.6 and figures 5.6a and 5.6b displays a situation where the LTE network has many variations. The standard deviation is high in this scenario compared to the TechnoWLAN scenarios. This is because the throughput numbers are higher. Figure 5.6a shows that the LTE link did not acquire full carrier aggregation every time due to the outliers around the 0.2 mark on the y-axis.

The download time difference between single-link LTE and the average download time between the multi-link approaches is about 11 seconds. This makes the multi-link results significantly better.

5.2.3 Analysis

Compiling the results from both the TechnoWLAN and AlwaysOnline scenario, figure 5.7 compares all the download times between the approaches. The tables below focus on download time and compare the multi-link results with each other in a matrix. Each value in the table is a calculation between the row-name and the column-name. Example; table 5.7 shows there is a 50.17% difference between Algorithm 1 and the Naive approach. This means Algorithm 1 used 50.17% of the time to download compared to the naive approach. The averages at the right are average-calculations of the row values.

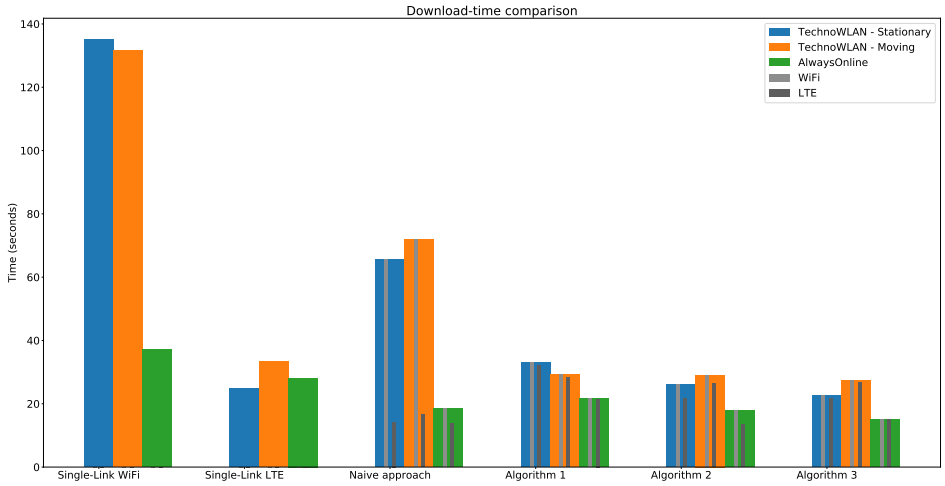


Figure 5.7: Download time comparison between the approaches in the dynamic scenarios. The thin bars in-between visualizes the Wi-Fi download time and LTE download time, respectively.

	Time ↑	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	65.59	–	199.30%	251.68%	291.38%	*247.45*
Algorithm 1	32.91	50.17%	–	126.28%	146.20%	*100.91*
Algorithm 2	26.06	39.73%	79.18%	–	115.77%	*78.02*
Algorithm 3	22.51	34.31%	68.39%	86.37%	–	*63.02*

Table 5.7: Algorithm download time comparison calculated in percentages for the TechnoWLAN Stationary scenario. The percentage values are calculations between the row-name and the column-name, meaning; how long the row-name used to download compared to the column-name.

	Time ↑	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	71.78	–	246.15%	246.92%	263.02%	*252.03*
Algorithm 1	29.16	40.62%	–	100.30%	106.85%	*82.58*
Algorithm 2	29.07	40.49%	99.69%	–	106.52%	*82.23*
Algorithm 3	27.29	38.01%	93.58%	93.87%	–	*75.15*

Table 5.8: Algorithm download time comparison calculated in percentages for the TechnoWLAN Moving scenario.

	Time ↑	**Naive A.h**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	18.42	–	84.65%	102.73%	123.29%	*103.55*
Algorithm 1	21.76	118.13%	–	121.36%	145.64%	*128.37*
Algorithm 2	17.93	97.33%	82.39%	–	120.01%	*99.91*
Algorithm 3	14.94	81.10%	68.65%	83.32%	–	*77.69*

Table 5.9: Algorithm download time comparison calculated in percentages for the AlwaysOnline scenario.

The percent calculations in the tables above are based on the results in the tables 5.5a, 5.5b, and 5.6, and translate to; the amount of download time the row-name uses compared to the column name.

In the stationary TechnoWLAN table 5.7, Algorithm 3 on average uses 63% of the time to download compared to the others. In the moving TechnoWLAN table 5.8, Algorithm 3 uses on average 75% of the time to download, and lastly, in the AlwaysOnline scenario, the Algorithm 3 uses on average 77% of the time to download compared to the other multi-link approaches. Comparing across the scenarios, Algorithm 3 uses on average 51% of the time to download versus the naive approach.

Further, figure 5.7 visualizes the download time differences between the approaches in the dynamic scenario. The figure shows that for both the TechnoWLAN and the AlwaysOnline measurements, the multi-link approaches are almost always better at download times compared to singe-link. The two scenarios represent different challenges, where TechnoWLAN provides relatively consistent LTE throughput, the AlwaysOnline measurements do not.

Why Section 5.1.3 concluded that Algorithm 2 performed poorly when sharing the load between the links in the high-speed scenario, and the same pattern is observed in the figure 5.7, where Algorithm 2 for all scenarios fail to share the load equally and leaving the LTE link unused for several seconds at the end of the download. From the same analysis, in section 5.1.3, it was observed that Algorithm 1 shared the load more equally between the links, but failed to reach the same high throughput values of Algorithm 3. Figure 5.7 visualizes the same pattern, where Algorithm 1 shares the load equally but fails to reach the high throughput numbers of Algorithm 3.

The reason Algorithm 3 outperforms Algorithm 1 and 2 in every scenario, is due to having the most dynamic request-size calculations, suitable

for these scenarios. Therefore, the differences are more polarizing than in the isolated scenario, leading to not only Algorithm 3 outperforming the others, but even more so in dynamic scenarios.

5.3 Analysis

The scenario-analysis done in section 5.1.3 and 5.2.3, showed that Algorithm 3 has the lowest download times, by utilizing all the input parameters. In this section, we look into why Algorithm 2, using many of the same request-size calculations as Algorithm 3, performed so poorly. We also look into the high standard-deviation numbers for the LTE link.

5.3.1 Why is Algorithm 2 performing so poorly?

In almost every scenario presented, Algorithm 2 has the highest Link Δ (link completion time difference), compared to Algorithm 1, and 3. Further analysis also discovers that Algorithm 2 also has the highest σ (standard deviation) for the Wi-Fi link compared across all scenarios. Focusing on all the results from Algorithm 2 in this chapter, we get;

	Time (s)			**Throughput**		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
H.S S6 – Algorithm 2	18.86	15.77	3.09	78.98	41.36	24.93	8.03
H.S S7 – Algorithm 2	18.44	15.89	2.55	78.90	47.85	0.43	5.93
L.S – Algorithm 2	29.18	28.49	0.69	7.49	66.66	0.00	4.62
T.S – Algorithm 2	26.06	21.77	4.29	14.73	76.28	3.35	11.33
T.M – Algorithm 2	29.07	26.50	2.57	13.89	66.77	36.07	2.62
A.O – Algorithm 2	17.93	13.36	4.57	54.96	96.99	12.71	34.53
Average	–	–	*2.96*	–	–	*12.92*	*11.18*

Table 5.10: Every result from Algorithm 2

Table 5.10 shows Algorithm 2 having high link-delta and high standard deviation for both the Wi-Fi and LTE link. The reason for the high Link Δ numbers is probably due to the links not being fully utilized and to the upper limit being decreased. This can introduce large requests to begin assigned to one link, and small to another. In section 4.3.4, Algorithm 2 is described to recalculate the upper limit based on throughput measurement for the links. This recalculation does not necessary always push the upper limit in an increasing direction, but could also decrease the limit if one of the links periodically reports poor throughput numbers. For the LTE link, this effect can have a negative influence on throughput, due to previously discussed carrier-aggregation challenges, and high RTT numbers. The Bandwidth Delay Product solves this and is the main difference between Algorithm 2 and 3.

The reason for the high standard deviation numbers is related to the Link Δ numbers. Links which are not being fully utilized cannot reach maximum throughput and are therefore limited by the request-size. If

the smaller requests-sizes vary significantly, this could result in varying throughput numbers and therefore high standard deviation.

5.3.2 Why is the correlation between high throughput and high standard deviation for the LTE link so strong?

Are the approaches or devices incapable of handling the high throughput numbers of the LTE link?

Using the results from the scenarios in this chapter, and focusing on the *one* result from each scenario with the highest LTE throughput, we get;

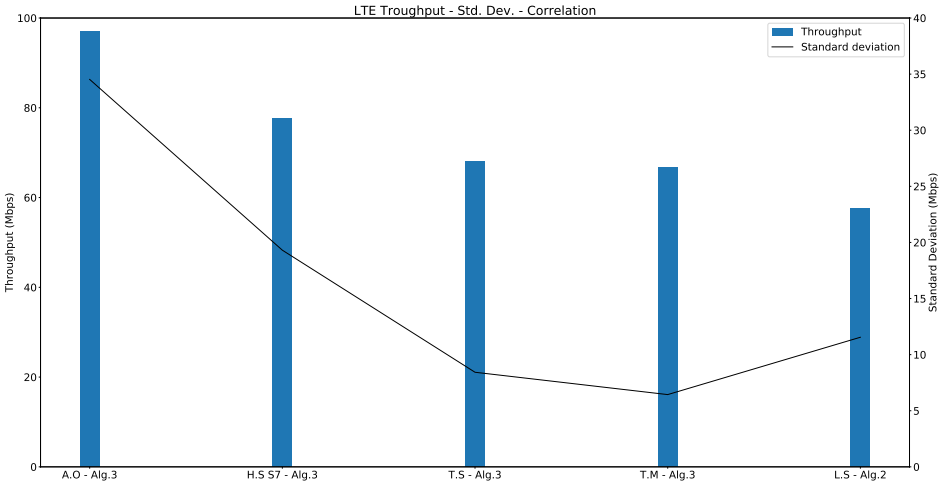


Figure 5.8: Correlation between LTE throughput and standard deviation. The blue bars represent LTE throughput (Mbps), and the black line standard deviation.

	Time (s)			**Throughput**		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
A.O – Algorithm 2	17.93	13.36	4.57	54.96	*96.99*	12.71	34.53
T.S – Naive approach	65.59	14.25	51.34	15.43	*77.62*	0.77	19.31
T.M – Naive approach	71.78	16.54	55.24	14.34	*68.13*	5.10	8.43
L.S – Algorithm 3	29.01	29.33	0.32	7.49	*66.70*	0.00	6.44
H.S S7 – Naive approach	13.27	18.99	5.72	78.96	*57.53*	0.13	11.55
Average					*73.39*		

Table 5.11: The results with highest LTE throughput

Calculating the correlation coefficient for the relationship between LTE throughput and standard deviation, the result is; *0.91*. This strong correlation is visible in figure 5.8, and could be the result of many factors, and some are; the server is incapable of delivering the high throughput, the network between the server and the client is incapable, or the device and approach is incapable.

Device and approach Could this strong relationship between high LTE throughput and standard deviation mean the device is incapable of handling this amount of data? According to the device specification presented in section 3.2.1, the Samsung S7 is equipped with a Cat9 modem, and should be capable of handling 450/50Mbps, meaning; 450Mbps for the downlink and 50Mbps for the uplink. The results from the scenarios are far below 450Mbps, so the question remaining is; is the Samsung S7 capable of handling the throughput, or is there a performance bottleneck in the approaches? The answer to the approach part of the question lies in the occurrence of the naive-approach. This approach is the simplest of all the multi-link approaches, yet are still present in three of the five results presented in table 5.11. This leads to the probability of faulty approaches being relative low, and the suspected reason for this high standard deviation numbers are, most probably due to the network or the server being incapable.

Server or network Table 5.11 shows the average LTE throughput is 73.39Mbps. In the High-Speed results, the Wi-Fi link reached throughput values of 80Mbps, while still having a very low standard deviation. This rules out the server as the suspected reason and leaves only the LTE network left as the suspected reason for the high standard deviation numbers.

5.4 All results

This section presents all the measurement results of this chapter.

	Time (s)			Throughput		σ	
	Wi-Fi	LTE	Link Δ	Wi-Fi	LTE	Wi-Fi	LTE
H.S S6 – Single-Link Wi-Fi	26.06	–	–	79.01	–	0.02	–
H.S S6 – Single-Link LTE	–	52.97	–	–	39.28	–	3.85
H.S S6 – Naive approach	13.19	32.36	19.17	79.01	33.79	0.05	7.26
H.S S6 – Algorithm 1	18.34	18.35	0.01	79.01	36.29	0.03	9.09
H.S S6 – Algorithm 2	18.86	15.77	3.09	78.98	41.36	24.93	8.03
H.S S6 – Algorithm 3	*16.68*	*16.29*	0.39	78.98	49.81	0.05	10.92
H.S S7 – Single-Link Wi-Fi	26.25	–	–	78.99	–	0.74	–
H.S S7 – Single-Link LTE	–	33.79	–	–	61.37	–	11.31
H.S S7 – Naive approach	13.27	18.99	5.72	78.96	57.53	0.13	11.55
H.S S7 – Algorithm 1	17.88	17.67	0.21	78.90	42.10	0.03	2.73
H.S S7 – Algorithm 2	18.44	15.89	2.55	78.90	47.85	0.43	5.93
H.S S7 – Algorithm 3	*16.55*	*16.05*	0.50	78.97	51.60	0.05	4.34
L.S – Single-Link Wi-Fi	278.18	–	–	7.49	–	0.00	–
L.S – Naive approach	139.26	17.34	121.92	7.49	64.90	0.00	6.58
L.S – Algorithm 1	29.63	29.22	0.41	7.49	65.24	0.00	7.39
L.S – Algorithm 2	*29.18*	*28.49*	0.69	7.49	66.66	0.00	4.62
L.S – Algorithm 3	29.01	29.33	0.32	7.49	66.70	0.00	6.44
T.S – Single-Link Wi-Fi	135.05	–	–	15.27	–	2.11	–
T.S – Single-Link LTE	–	24.86	–	–	80.14	–	6.90
T.S – Naive approach	65.59	14.25	51.34	15.43	77.62	0.77	19.31
T.S – Algorithm 1	32.91	32.09	0.82	15.27	49.14	1.48	11.67
T.S – Algorithm 2	26.06	21.77	4.29	14.73	76.28	3.35	11.33
T.S – Algorithm 3	*22.51*	*21.67*	0.84	15.30	82.14	1.82	6.50
T.M – Single-Link Wi-Fi	131.59	–	–	16.59	–	6.76	
T.M – Single-Link LTE	–	33.32	–	–	67.30	–	1.30
T.M – Naive approach	71.78	16.54	55.24	14.34	68.13	5.10	8.43
T.M – Algorithm 1	29.16	28.41	0.75	14.76	64.96	4.30	11.68
T.M – Algorithm 2	29.07	26.50	2.57	13.89	66.77	36.07	2.62
T.M – Algorithm 3	*27.29*	*26.60*	0.69	14.07	66.94	3.78	2.40
A.O – Single-Link Wi-Fi	37.24	–	–	57.28	–	1.13	
A.O – Single-Link LTE	–	28.16	–	–	70.44	–	22.66
A.O – Naive approach	18.43	13.84	4.59	57.25	75.11	4.80	29.34
A.O – Algorithm 1	21.76	21.50	0.26	55.01	41.68	5.73	21.53
A.O – Algorithm 2	17.93	13.36	4.57	54.96	96.99	12.71	34.53
A.O – Algorithm 3	*14.85*	*14.94*	0.09	52.10	94.70	6.07	30.85

Table 5.12: All the throughput results from this chapter. *H.S* = High-Speed, *L.S* = Low-Speed *T.S* = TechnoWLAN Stationary, *T.M* = TechnoWLAN Moving, *A.O* = AlwaysOnline

5.5 Summary

This chapter has presented the throughput results from numerous experiments. It started by presenting the isolated scenario, where the single-link results were compared to the multi-link results and the multi-link against each other. In the isolated high-speed scenario, all the multi-link approaches download the payload in a shorter amount of time compared to single-link, and Algorithm 3 reached the theoretical aggregated maximum in most cases. In the isolated low-speed scenario, all the algorithms performed equally due to the static nature of the Wi-Fi network. Surprisingly,

Algorithm 2 had the worst load sharing capabilities between Algorithm 1, 2, and 3 – but still outperformed Algorithm 1 in the dynamic scenarios when comparing total download times.

In the dynamic scenarios, the approaches were tested in public Wi-Fi's. From these results, Algorithm 3 performed overall the best by reducing the download time significantly due to its adaptability in dynamic scenarios where signal strength and capacity are continuously varying.

Further, we observed that expected LTE throughput varies not only between locations but also from the unpredictable availability of the network. When the LTE throughput is high, the standard deviation is also increased – confirmed with a correlation between the two factors of *0.91*. The reason for the strong correlation was concluded to be an effect of the LTE network, not the server or client. A high standard deviation number means the multi-link approach needs to use the many input parameters for adapting to the rapid changes in the network.

The main takeaway answering the research questions is;

Question 1 – Input parameters Algorithm 3 uses the most input parameters and achieves the best results in every scenario. The reason why Algorithm 3 gained the highest aggregated throughput was the use of the input parameters; payload-size, throughput, bandwidth, and RTT. With all of these input parameters, Algorithm 3 continuously calculated both the upper and lower request-size limits, in addition to the dynamically sized request for each link.

The Bandwidth Delay Product (BDP) acted as a lower limit for the requests-sizes and proved to be essential for gaining the highest aggregated throughput. LTE networks are particularly vulnerable to small request sizes due to being a Long Fat Network – a network with high bandwidth and high delay. Algorithm 2 is using many of the same request-size calculations as Algorithm 3 but is lacking the lower limit calculations, and therefore confirmed the importance by performing comparatively weaker almost every scenario. Thereby concluding; the additional input parameters are essential not only in isolated scenarios but even more so in dynamic scenarios where the fluctuations are higher.

Question 2 – Performance Modern smartphones connected to high-speed wireless networks are able to handle the increased bandwidth from the multi-link approaches. In the best case, they reached throughput numbers enabling a 250MB payload download in under 15 seconds. Thereby concluding with; both the devices used in this thesis were capable of aggregating the links and achieving high throughput results in the numerous experiment presented in this chapter.

Chapter 6

Results: CPU and Memory usage

This chapter presents the CPU and memory results from the experiments. CPU and memory is a resource metric, and is measured to answer the second part of the research question; how the devices perform and how they are affected by multi-link approaches. CPU and memory usage directly affects the device performance, possibly making it slow and unresponsive. Increased CPU-usage could also affect energy usage – which are presented in chapter 7.

In the isolated high-speed scenario, both the Samsung S6 and the Samsung S7 devices are presented, but the later scenarios only include the Samsung S7 due to the similar usage pattern of the two devices – section 3.3.

This chapter only discusses memory usage for the first scenario – the isolated high-speed scenario. For the following scenarios, the memory results are ignored due to the similarity of the results. However, they are included in the figures, and at the end of the chapter, all the memory results are presented in a table for reference.

Terminology The terms used in this chapter are;

- %: The default notation for CPU usage.
- *KB*: The default memory usage value.
- *p50*: 50th percentile the same as *Median*
- *p90*: 90th percentile
- *p05*: 5th percentile
- *Max*: Maximum
- *Time* ↑: Represents the total download time
- $\int$: Integral

- c_v: Coefficient of variation
- Δ: Used for difference between x and y

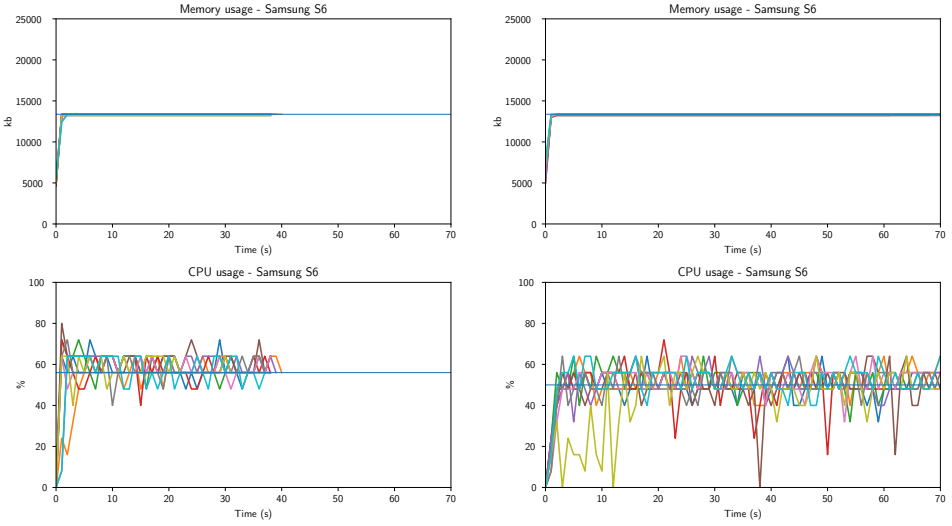
Further description in section 3.4.

6.1 Isolated Scenarios

This section presents the CPU and memory usage results from the isolated scenarios. During measurement of CPU and memory usage, the focus was on initial device conditions, meaning; making sure no other unnecessary applications were running CPU and or memory intensive processes. All the devices presented in this thesis were running the same operating system, with the same version number, and set up as equal as otherwise possible – more information in the testbed section 3.2.

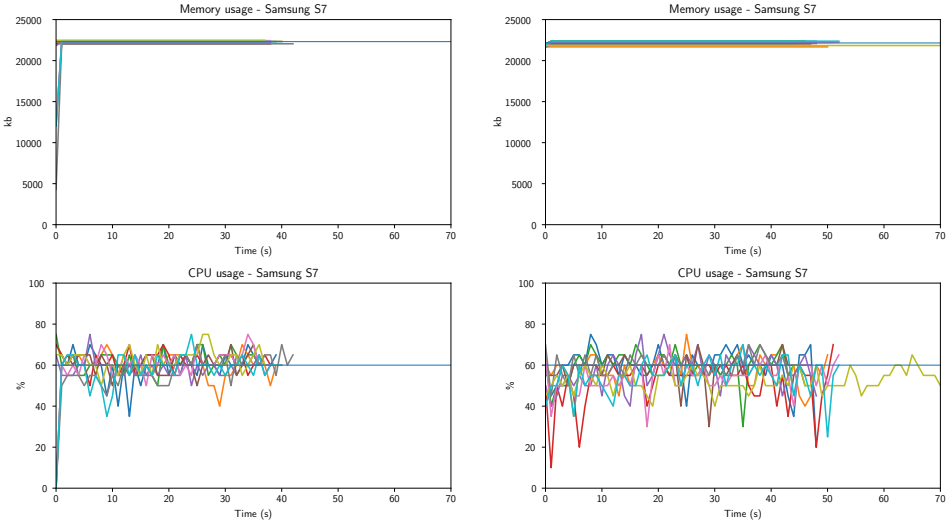
6.1.1 High-Speed Wi-Fi

The results from measuring single-link CPU and memory usage in this scenario stand as benchmarks for the results with the naive approach and algorithms. All the results presented here are from the same measurements presented in chapter 5. The single-link figures 6.1a, 6.1b, 6.1c and 6.1d visualizes the CPU and memory levels of the Samsung S6 and S7 devices during download. The multi-link figures 6.2a, 6.2b, 6.2c and 6.2d visualizes CPU and memory usage over time for the Samsung S7 device is presented. The Samsung S6 device is included as tabular data.



(a) Wi-Fi – Samsung S6

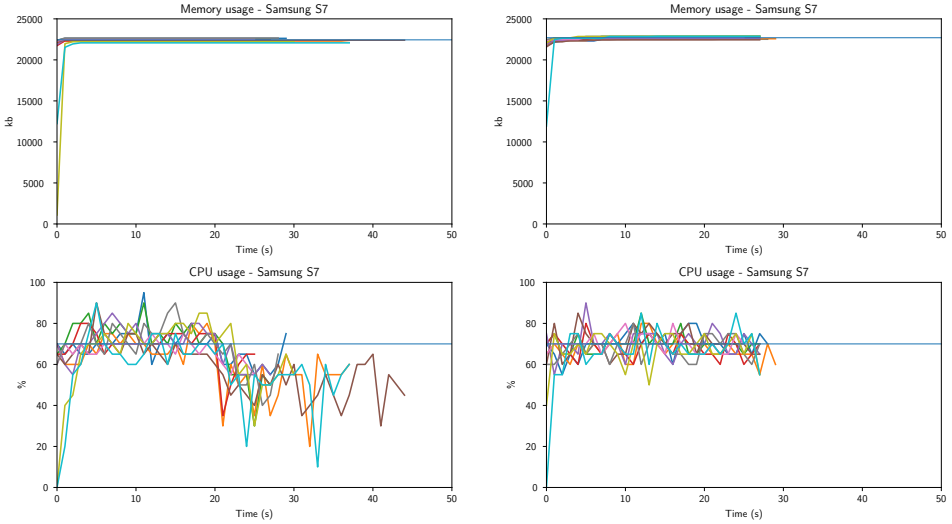
(b) LTE – Samsung S6



(c) Wi-Fi – Samsung S7

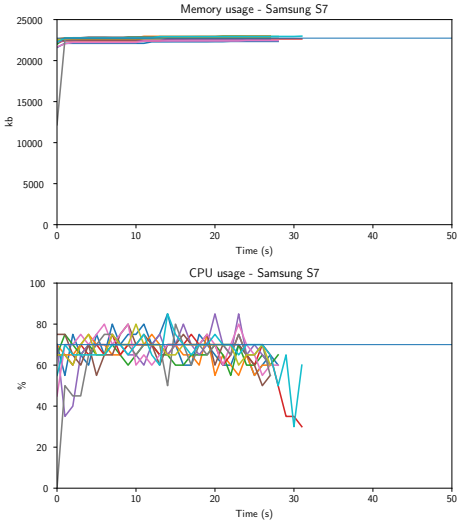
(d) LTE – Samsung S7

Figure 6.1: Single-Link CPU and memory usage for the Samsung S6 and the Samsung S7 device. The lines represent individual measurements – 10 in total. The straight horizontal line visualize the median-of-medians.

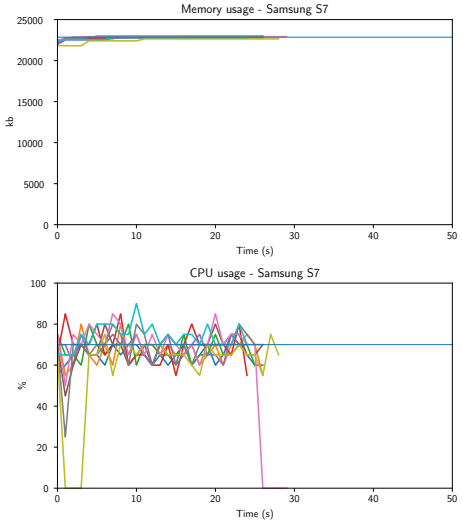


(a) Naive approach

(b) Algorithm 1



(c) Algorithm 2



(d) Algorithm 3

Figure 6.2: Multi-Link CPU and memory usage for the Samsung S7 device. The lines represent individual measurements – 10 in total.

	CPU % Median						
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	26.06	56.00	5.86	2185.33	68.00	64.00	47.60
Single-Link LTE	52.97	50.00	7.71	3739.00	64.00	56.00	40.00
Naive approach	32.36	56.00	11.60	2601.33	76.00	64.00	40.00
Algorithm 1	18.35	64.00	0.00	1741.67	72.00	68.40	51.60
Algorithm 2	18.86	64.00	9.34	1810.33	80.00	72.00	53.80
Algorithm 3	16.68	68.00	6.20	*1689.67*	80.00	73.60	52.00

(a) Samsung S6 – CPU

	CPU % Median						
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	26.25	60.00	2.61	2334.17	72.50	65.00	52.25
Single-Link LTE	33.79	60.00	4.45	2800.83	70.00	65.00	45.00
Naive approach	18.99	70.00	7.47	1948.33	87.50	75.00	49.38
Algorithm 1	17.88	70.00	2.44	1876.04	80.00	75.00	60.00
Algorithm 2	18.44	70.00	3.08	1874.54	80.00	74.75	55.00
Algorithm 3	16.55	70.00	4.93	*1717.96*	80.00	75.00	60.00

(b) Samsung S7 – CPU

	Memory Median		
	KB	c_v	$\int$
S.L Wi-Fi	13366.00	0.51	504550.67
S.L LTE	13370.00	0.54	999743.33
N.A	13532.00	0.06	633393.00
Alg.1	13856.00	0.18	384648.67
Alg.2	13844.00	0.15	383212.00
Alg.3	13878.00	0.08	*349072.00*

(c) Samsung S6 – Memory

	Memory Median		
	KB	c_v	$\int$
S.L Wi-Fi	22259.60	0.70	848489.33
S.L LTE	22152.00	1.09	1078109.33
N.A	22442.00	0.83	645245.33
Alg.1	22700.00	0.73	613168.53
Alg.2	22738.00	1.04	620259.33
Alg.3	22854.00	0.41	*591980.67*

(d) Samsung S7 – Memory

Table 6.1: High-Speed CPU and memory results both devices. The CPU numbers are presented in percentages, and the memory are presented in KB. The download times in this table are presented in seconds and are the total download time.

Single-Link Table 6.1a and 6.1b show that the CPU median (p50) for both devices are about equal for the single-link approaches, with the Samsung S7 having slightly higher median compared. The integral numbers are higher for the single-link LTE measurements due to increased download times. This difference is also present in the memory numbers in table 6.1c and 6.1d for both devices but compared the Samsung S6 uses less memory for a longer period than the Samsung S7.

Multi-Link For the multi-link results, table 6.1b and 6.1a shows that, there are small differences in CPU median between the multi-link approaches. The differences between the naive approach and the algorithms are due to the resource curve the naive approach has when one link is done downloading, and the last link is left. Figure 6.2a for the naive approach shows curve then dips to a lower number, making the median number

lower. However, for the integral numbers, for both devices, the naive approach uses more CPU resources than the algorithms. The same pattern can be observed for memory resources. Algorithm 1, 2 and 3 use fewer CPU resources according to the integral numbers.

		Single-Link Δ		**Single-Link %**	
	$\int$	*Wi-Fi*	*LTE*	*Wi-Fi*	*LTE*
Single-Link Wi-Fi	2185.33	–	–	–	–
Single-Link LTE	3739.00	–	–	–	–
Naive approach	2601.33	-416.00	1137.67	119.03%	69.57%
Algorithm 1	1741.67	433.66	1997.33	79.70%	46.58%
Algorithm 2	1810.33	375.00	1928.67	92.84%	48.41%
Algorithm 3	1689.67	495.66	2049.33	*77.31%*	*45.19%*

(a) CPU usage

		Single-Link Δ		**Single-Link %**	
	$\int$	*Wi-Fi*	*LTE*	*Wi-Fi*	*LTE*
Single-link Wi-Fi	504550.67	–	–	–	–
Single-link LTE	999743.33	–	–	–	–
Naive approach	633393.00	-128842.33	366350.33	125.53%	63.35%
Algorithm 1	384648.67	119902.00	615084.66	76.23%	38.47%
Algorithm 2	383212.00	121338.67	616531.33	75.95%	38.33%
Algorithm 3	349072.00	155478.67	650671.33	*69.18%*	*34.91%*

(b) Memory Usage

Table 6.2: High-Speed CPU and memory usage differences between multi-link downloads and Single-Link downloads for the Samsung S6 device. The "%" columns are the Δ numbers converted to percentages. The memory integral numbers in this table are based on KB.

Compared The Samsung S6 device CPU numbers, in table 6.2a, shows that the naive-approach uses more CPU resources than a single-link Wi-Fi download, but significant less CPU usage compared to single-link LTE. Algorithm 1, 2, and 3 are increasingly more efficient in both CPU and memory usage compared to both single-link Wi-Fi and LTE. On average the algorithms uses about 83% CPU resources compared to single-link Wi-Fi, and only approximately 46% of the CPU resources compared to single-link LTE.

Table 6.2b shows that the algorithms use approximately 73% of the memory resources compared to single-link Wi-Fi, and about 36% compared to single-link LTE. The Samsung S6 device used in these measurements, with the current configuration, utilizes approximately 1531MB of the 2700MB available system memory in idle mode after a reboot. The idle memory usage is about 56.7%, and the impact the algorithms has on the device is negligible with about 13MB during download. This small memory footprint, combined with the fact that any of the approaches use

about 13MB in all the scenarios, is the reason memory usage is not focused on after this section.

		Single-Link Δ		**Single-Link %**	
	$\int$	*Wi-Fi*	*LTE*	*Wi-Fi*	*LTE*
Single-Link Wi-Fi	2334.17	–	–	–	–
Single-Link LTE	2800.83	–	–	–	–
Naive approach	1948.33	385.84	852.50	83.47%	69.56%
Algorithm 1	1876.04	458.13	924.79	80.37%	66.98%
Algorithm 2	1874.54	459.63	926.29	80.30%	66.92%
Algorithm 3	1717.96	616.21	1082.87	*73.60%*	*61.33%*

(a) CPU usage

		Single-Link Δ		**Single-Link %**	
	$\int$	*Wi-Fi*	*LTE*	*Wi-Fi*	*LTE*
Single-link Wi-Fi	848489.33	–	–	–	–
Single-link LTE	1078109.33	–	–	–	–
Naive approach	645245.33	203244.00	432864.00	76.04%	59.85%
Algorithm 1	613168.53	235320.80	464940.80	72.26%	56.87%
Algorithm 2	635897.57	212591.76	442211.76	74.94%	58.98%
Algorithm 3	587527.13	260962.20	490582.20	*69.24%*	*54.49%*

(b) Memory usage

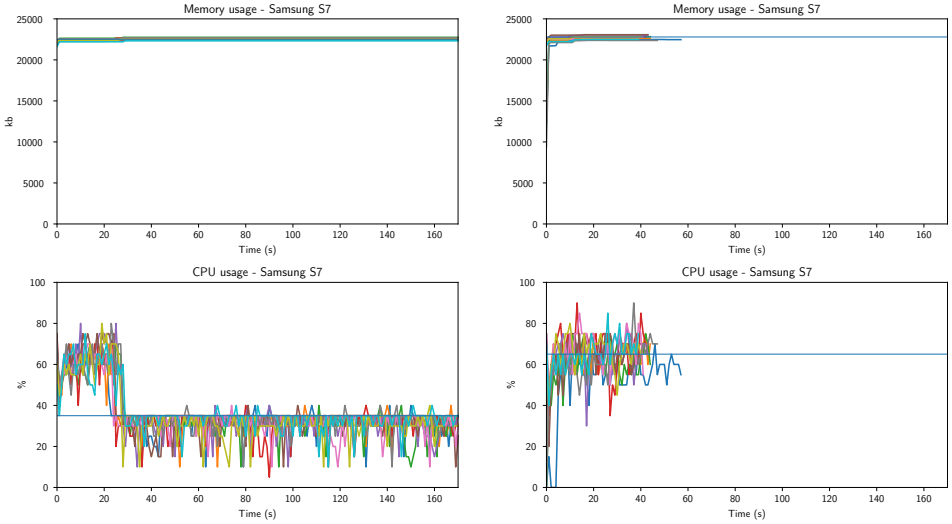
Table 6.3: High-Speed CPU and memory resource differences between multi-link downloads and single-link (S.L) downloads for the Samsung S7 device. The "%" columns are the Δ numbers converted to percentages. The memory integral numbers in this table are based on KB.

The High-Speed Samsung S7 CPU integral numbers, in table 6.3a, display almost the same trend as the Samsung S6 device. For the CPU integral numbers, the algorithms use approximately 77% of the resources compared to single-link Wi-Fi, and 64% compared to the single-link LTE.

Further, the memory usage integral numbers in table 6.3b for the Samsung S7 device display a usage of about 71% compared to the single-link Wi-Fi result, and about 56% compared to single-link LTE. The Samsung S7 device is equipped with 3557MB of system memory and uses about 1803MB after a reboot, which leads 50% memory usage in idle mode. The addition of about 23MB over time is not significant compared to the total available system memory.

6.1.2 Low-Speed Wi-Fi

The results in figure 6.3a and 6.3b are from the same measurements as in chapter 5, and it displays how the naive approach and Algorithm 3 handles situations where the bandwidth from the Wi-Fi link is significantly slower than the LTE link.



(a) Naive approach

(b) Algorithm 3

Figure 6.3: Multi-Link CPU and memory usage form the Low-Speed Wi-Fi scenario. The measurements are done with a Samsung S7 device.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	278.18	32.50	8.11	*12101.67*	67.50	35.00	20.00
H.S Single-Link LTE	33.79	60.00	4.45	2800.83	70.00	65.00	45.00
Naive approach	139.26	35.00	4.58	6902.50	75.00	55.00	20.00
Algorithm 1	29.63	65.00	6.48	2758.12	80.00	74.25	45.00
Algorithm 2	29.18	65.00	5.31	*2727.29*	80.00	72.50	48.50
Algorithm 3	29.33	65.00	7.71	2789.17	80.00	75.00	48.75

Table 6.4: Low-Speed Wi-Fi scenario using the Samsung S7 device. The time columns show the number of seconds it takes to download the payload, where the slowest link time is used. The CPU numbers are presented in percentages.

Table 6.4 shows that when downloading the payload using single-link Wi-Fi with a limited internet bandwidth of 8Mbps, the CPU integral is significantly higher compared to downloading the payload using multi-link.

However, combining a slow Wi-Fi link and a fast LTE link makes the download about 4 seconds faster than downloading using only a single-link LTE – table 6.1b. Algorithm 1, 2, and 3, has CPU integral numbers averaging 2758 compared to 2800.83 for the single-link LTE download; this difference is within a 2% margin and should be considered insignificant. The 4 seconds improvement means Algorithm 1, 2, and 3 use 85% of the download time, in the low-speed scenario, compared to the single-link LTE results from the high-speed scenario. However, the CPU usage is almost the same – 98.47%.

Comparing Algorithm 3 with the single-link Wi-Fi measurement, the differences in download time are 248.85 seconds meaning Algorithm 3 uses 11.78% of the time to download. The differences in CPU integral-numbers means Algorithm 3 uses 23.04% of the resources.

6.1.3 Analysis

The following figure and tables show the difference between the approaches in the high-speed scenario, focusing on the CPU integral numbers. The numbers in the tables are percentage calculations between the row-name and column-name. Example; in table 6.5a Algorithm 3 uses 64.95% of the CPU resources compared to the naive approach.

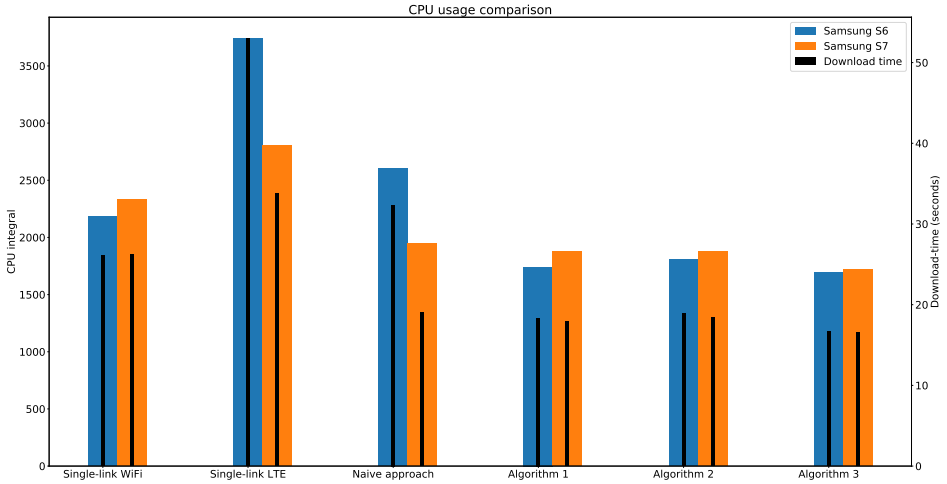


Figure 6.4: CPU and memory comparisons between the approaches and devices in the High-Speed isolated scenario. The thin black bars represents download time in the secondary y-axis.

	CPU $\int$	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	2601.33	–	149.36%	143.69%	153.95%	*149.00*
Algorithm 1	1741.67	66.95%	–	96.20%	103.08%	*88.74*
Algorithm 2	1810.33	69.59%	103.94%	–	107.14%	*93.56*
Algorithm 3	1689.67	64.95%	97.01%	93.33%	–	*85.10*

(a) High-Speed – Samsung S6

	CPU $\int$	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	1948.33	–	109.20%	109.17%	113.41%	*110.59*
Algorithm 1	1876.04	96.28%	–	100.02%	109.20%	*101.83*
Algorithm 2	1875.54	96.26%	99.97%	–	105.39%	*100.54*
Algorithm 3	1717.96	88.17%	91.57%	91.59%	–	*90.44*

(b) High-Speed – Samsung S7

	CPU $\int$	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	6902.50	–	250.26%	253.09%	247.48%	*250.28*
Algorithm 1	2758.12	39.96%	–	101.13%	98.89%	*79.99*
Algorithm 2	2727.29	39.51%	98.88%	–	97.78%	*78.72*
Algorithm 3	2789.17	40.41%	101.13%	102.27%	–	*81.27*

(c) Low-Speed – Samsung S7

Table 6.5: Multi-link approach, CPU integral comparison in percentages for both devices. *H.S* = High-Speed, *L.S* = Low-Speed. Percentage values representing the usage of the row-name compared to the column-name.

In table 6.5a Algorithm 3 uses on average 85.10% of the CPU resources compared to the other multi-link approaches. For the Samsung S7, in table 6.5b, the same average is 90.44%. For the low-speed results in table 6.5c, Algorithm 2 is the most efficient by using 78.72%.

Figure 6.4 shows that for Algorithm 1, 2, and 3, the Samsung S6 device uses slightly fewer CPU resources compared to the Samsung S7 device. Further, the Samsung S6 CPU integral numbers show that the naive approach uses more resources than a single-link Wi-Fi download, and significant less compared to single-link LTE. All the algorithms are increasingly more efficient in CPU usage compared to both single-link Wi-Fi and LTE.

Comparing with throughput Section 5.1.3 showed that for the high-speed scenario, Algorithm 3 outperformed the other multi-link approaches by using on average 76% of the time to download for the Samsung S6 and 89% for the Samsung S7. These differences are again represented in the CPU usage tables 6.5a and 6.5b, however, the differences are not as significant as the throughput differences, with the CPU differences being 85% for the Samsung S6 and 90% for the Samsung S7. This represents a 5% difference compared to the 13% for the throughput.

For the low-speed table in section 5.1.3, Algorithm 2 had a slight advantage over the other, and the same pattern is displayed here.

Why In the high-speed isolated scenario, Algorithm 3 gained the highest throughput for both links, and therefore downloaded the payload in the

least amount of time. Algorithm 3 uses the least amount of CPU resources and confirms the relationship between download time and CPU usage answering why Algorithm 3 uses the least amount of resources.

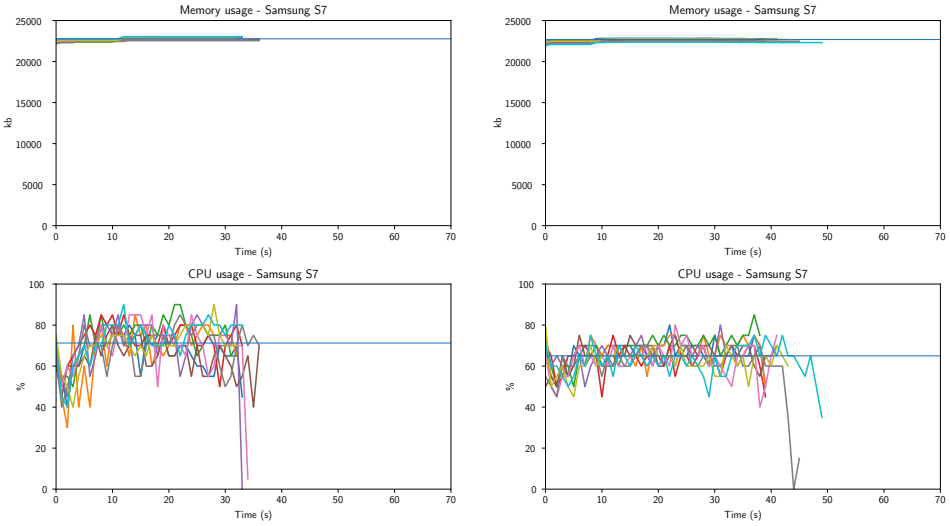
In the low-speed scenario, the differences between Algorithm 1, 2, and 3 is about 1.3% from the average. This is slightly larger than the observed difference below 1% from the average, in 5.1.3. However, the CPU integral numbers are very similar leading to the same conclusion; Algorithm 3 did not perform poorly; it was Algorithm 1 and 2 that excelled in this scenario.

6.2 Dynamic Scenarios

During the testing of throughput in the dynamic scenarios, CPU and memory were measured to give information on how the devices consume resources in multi-link situations including access point switching, loss of signal quality and more.

6.2.1 TechnoWLAN

The TechnoWLAN experiment, as described in more detail in section 3.3.0.2, focused on two scenarios; stationary and moving. The CPU and memory usage numbers in these situations reflect the influence volatile and unpredictable networks has with additional overhead like access point switching and LTE signal strength fluctuations. The figures 6.5a and 6.5b visualizes the CPU and memory levels of Algorithm 3 during the stationary and moving scenario.



(a) Stationary – Algorithm 3

(b) Moving – Algorithm 3

Figure 6.5: CPU and memory usage for the TechnoWLAN scenario using the Samsung S7 device.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	135.05	37.50	21.46	6015.21	65.00	55.00	0.00
Single-Link LTE	24.86	70.00	3.48	2417.92	80.00	75.00	45.25
Naive approach	65.59	45.00	8.11	4016.25	80.00	69.50	0.00
Algorithm 1	32.91	62.50	7.00	2932.71	80.00	74.00	40.00
Algorithm 2	26.06	70.00	9.41	2459.79	82.50	76.75	45.00
Algorithm 3	22.18	71.25	4.66	*2352.08*	85.00	80.00	51.50

Table 6.6: CPU and memory results from the TechnoWLAN Stationary scenario using the Samsung S7. The download time is presented in seconds and shows the slowest of the two links. The CPU numbers are presented in percentages.

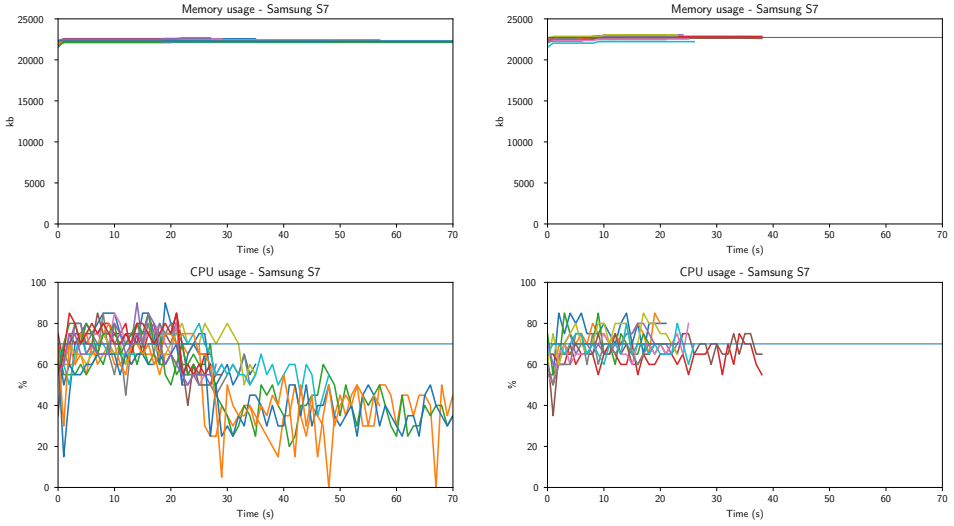
		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	131.59	42.50	47.55	6794.17	62.50	50.00	4.00
Single-Link LTE	33.32	60.00	3.84	2911.67	70.00	65.00	48.62
Naive approach	71.78	45.00	38.85	4185.00	75.00	65.00	2.50
Algorithm 1	29.16	63.75	5.50	2644.17	75.00	70.00	50.00
Algorithm 2	29.07	65.00	5.13	2715.62	75.00	70.00	50.00
Algorithm 3	27.29	65.00	3.66	*2608.54*	77.50	70.00	52.38

Table 6.7: CPU and memory results from the TechnoWLAN Moving scenario using the Samsung S7. The download time is presented in seconds, and shows the slowest of the two links. The CPU numbers are presented in percentages.

The tables 6.6 and 6.7 shows, on average the algorithms (1, 2, and 3) use 27.05 seconds downloading the payload stationary, compared to 28.50 seconds for the moving. This small difference reflects on the CPU integral as well. The approximate average CPU integral for the stationary results is 2581, compared to 2667 for the moving – visible in the x-axis in figure 6.5a and 6.5b. This represents an increase in about 103% for the moving results. From these results, it is difficult to conclude that noise influences CPU usage in a significant way. The biggest factor affecting CPU usage is throughput leading to download time.

6.2.2 AlwaysOnline

The TechnoWLAN scenarios represent relative strong Wi-Fi signal strength in both moving and stationary cases. The AlwaysOnline scenario, however, represents more unpredictable and volatile signal strengths and throughput. The figures 6.6a and 6.6b visualizes CPU and memory levels over time for the Naive approach and Algorithm 3.



(a) Naive approach

(b) Algorithm 3

Figure 6.6: CPU and memory usage for the AlwaysOnline scenario using the Samsung S7 device.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	37.24	55.00	0.00	2873.12	70.00	60.00	48.50
Single-Link LTE	28.16	65.00	18.62	2523.33	80.00	75.00	45.25
Naive approach	18.43	70.00	20.98	2229.58	85.00	77.50	48.75
Algorithm 1	21.76	68.75	5.14	2191.88	82.50	75.00	60.00
Algorithm 2	17.93	65.00	6.85	1738.33	80.00	72.50	53.50
Algorithm 3	14.94	70.00	6.67	*1672.92*	80.00	75.00	60.00

Table 6.8: CPU and memory results from the AlwaysOnline scenario using the Samsung S7 device. The download time are presented in seconds, and only the slowest link is presented. The CPU numbers are presented in percentages and memory in KB

The integral numbers in table 6.8, shows that both the single-link measurements stand out as using more CPU resources due to slower download times. Single-link Wi-Fi in this scenario uses almost double the number of CPU resources as Algorithm 3, according to the integral numbers. For the single-link LTE numbers, the difference is not as big as with the single-link Wi-Fi.

Further, comparing the numbers vertically in the table, it shows that both the p50 (median), and p05 numbers increase when the download time decreases. This could mean a correlation between CPU usage and throughput numbers. In unpredictable scenarios, high throughput numbers are not a guaranty. More so, the integral numbers show that using more CPU resources over a shorter amount of time is preferable to use fewer CPU resources over a more extended amount of time.

6.2.3 Analysis

The following figure and tables compare the CPU integral between the algorithms and shows for each scenario. The single-link results are left out of the tables due to the quadratic nature of the tables.

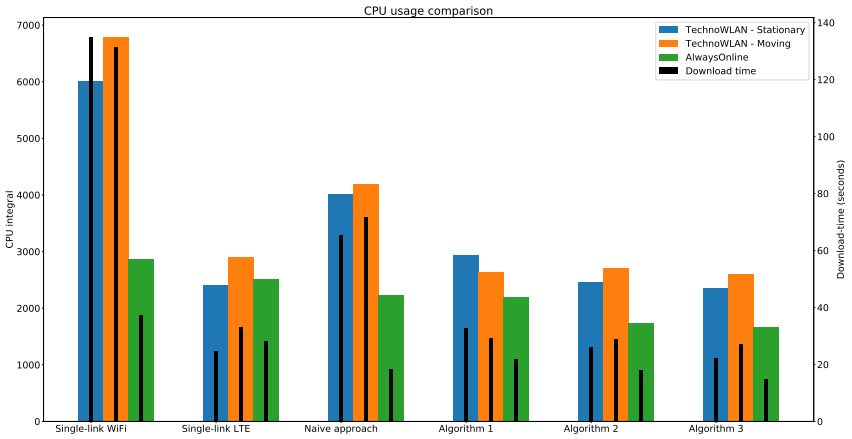


Figure 6.7: CPU usage comparisons between the approaches and devices. The thin black bars represents download time in the secondary y-axis.

	CPU $\int$	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	4016.25	–	136.97%	163.28%	170.75%	*158.33*
Algorithm 1	2932.71	73.02%	–	119.22%	124.66%	*105.63*
Algorithm 2	2459.79	61.24%	83.87%	–	104.58%	*83.23*
Algorithm 3	2352.08	58.56%	80.21%	95.62%	–	*78.13*

Table 6.9: Algorithm CPU-integral comparison in percentages for the TechnoWLAN Stationary scenario

	CPU $\int$	**Naive A.**	**Alg.1**	**Alg.2**	**Alg.3**	*Average*
Naive approach	4185.00	–	158.27%	154.11%	160.43%	*157.60*
Algorithm 1	2644.17	63.18%	–	97.36%	101.37%	*87.30*
Algorithm 2	2715.62	64.88%	102.70%	–	104.10%	*90.56*
Algorithm 3	2608.54	62.33%	98.65%	96.05%	–	*85.68*

Table 6.10: Algorithm CPU-integral comparison in percentages for the TechnoWLAN Moving scenario

	CPU $\int$	Naive A.	Alg.1	Alg.2	Alg.3	*Average*
Naive approach	2229.58	–	101.72%	128.26%	133.27%	*121.08*
Algorithm 1	2191.88	98.30%	–	126.09%	131.02%	*118.47*
Algorithm 2	1738.33	77.96%	79.30%	–	103.91%	*87.06*
Algorithm 3	1672.92	75.03%	76.32%	96.23%	–	*82.53*

Table 6.11: Algorithm CPU-integral comparison in percentages for the AlwaysOnline scenario

The tables 6.9, 6.10, and 6.11, show the compared CPU usage between the multi-link approaches. In the stationary TechnoWLAN table 6.9, Algorithm 3 uses on average 78.13% compared to the others, and the moving 6.10 the average is 85.68%. For the AlwaysOnline scenario, the difference is 82.53%. The biggest difference is between Algorithm 3 and the naive approach, where the algorithm uses on average 65% of the time to download across all scenarios.

Why Section 5.2.3 answered why Algorithm 3 performed best in a dynamic scenario, due to having the most advanced request-size calculations. Figure 6.7 almost mirrors the throughput figure, by having a very similar pattern. The thin black bars, representing download time, confirms this. Algorithm 1 has the least overhead between CPU usage and download-time, Algorithm 2 slightly more, and Algorithm 3 the most. The differences are not significant, and even if Algorithm 3 has the highest overhead between download time and CPU usage, the decreased download time makes the CPU usage overall the least compared to Algorithm 1, and 2. This pattern can be attributed to Algorithm 3 having the most advanced request-size calculations, and thereby needing slightly more CPU cycles during download.

6.3 Analysis

The scenario-analysis, in section 6.2.3 and 6.2.3 concluded that Algorithm 3 uses the least amount of CPU resources, confirming the importance of the input parameters also in regards to CPU usage. CPU and memory are a resource metric, and the main reason we measure the usage is to answer the second research question; how the devices are affected by using HTTP multi-link. This analysis combines all the results from both the isolated and the dynamic scenario and answers; what are the differences in CPU usage between single-link and multi-link, and what are the correlation between CPU usage and throughput.

6.3.1 What are the CPU-usage difference between the best single-link and the best multi-link result?

The tables below present the best single-link results and the best multi-link results for each scenario, in addition to the differences. Only results from the Samsung S7 are presented.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	26.25	60.00	2.61	2334.17	72.50	65.00	52.25
Algorithm 3	16.55	70.00	4.93	*1717.96*	80.00	75.00	60.00
Difference	-9.70	10	2.32	-616.21	7.5	10	7.72
Difference %	63.04	116.66	188.88	73.60	110.34	115.38	114.83

Table 6.12: The best single-link and multi-link results from the High-Speed Isolated Scenario.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link Wi-Fi	278.18	32.50	8.11	12101.67	67.50	35.00	20.00
Algorithm 2	29.18	65.00	5.31	*2727.29*	80.00	72.50	48.50
Difference	-258.00	32.50	-2.80	-9374.38	12.5	37.5	28.5
Difference %	10.16	200	65.47	22.53	118.51	207.14	242.50

Table 6.13: The best single-link and multi-link results from the Low-Speed Isolated Scenario.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link LTE	24.86	70.00	3.48	2417.92	80.00	75.00	45.25
Algorithm 3	22.18	71.25	4.66	*2352.08*	85.00	80.00	51.50
Difference	-2.68	1.25	1.18	-65.84	5.00	5.00	6.25
Difference %	89.21	101.78	133.90	97.27	106.25	106.66	113.81

Table 6.14: The best single-link and multi-link results from the TechnoWLAN Stationary scenario

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link LTE	33.32	60.00	3.84	2911.67	70.00	65.00	48.62
Algorithm 3	27.29	65.00	3.66	*2608.54*	77.50	70.00	52.38
Difference	-6.03	5.00	-0.17	-303.13	7.50	5.00	3.76
Difference %	81.90	108.33	95.31	89.58	110.71	116.66	107.73

Table 6.15: The best single-link and multi-link results from the TechnoWLAN Moving scenario

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
Single-Link LTE	28.16	65.00	18.62	2523.33	80.00	75.00	45.25
Algorithm 3	14.94	70.00	6.67	*1672.92*	80.00	75.00	60.00
Difference	-13.22	5.00	-11.95	-850.41	0.00	0.00	14.75
Difference %	53.05	107.69	35.82	66.29	100.00	100.00	132.59

Table 6.16: The best single-link and multi-link results from the AlwaysOnline scenario

Table 6.12, 6.13, 6.14, 6.15, and 6.16, shows all the scenarios being similar, expect the low-speed scenario which deviates from the pattern. Focusing on the integral numbers, the tables shows that for all scenarios the multi-link Algorithm 3 uses fewer CPU resources than its single-link counterpart. The scenario with the best overall multi-link CPU usage is the *AlwaysOnline* scenario with usages approx 60% compared to the single-link LTE results – excluding the low-speed scenario where the usage is 22.52%. The scenario with the least changes in single-link and multi-link is the stationary TechnoWLAN scenario where the Algorithm 3 results use mid 90% compared to the single-link LTE result.

This leads to a conclusion that; when comparing CPU usage multi-link approaches is preferable to single-link approaches, due to having lower download times and therefore lower usage numbers.

6.3.2 What is the correlation between CPU usage and high throughput?

Using total download time as an indicator of throughput, table 6.8 indicates that AlwaysOnline and Algorithm 3 has the lowest download time, and therefore the must have the highest aggregated throughput. Compiling the best results from each scenario, we get the following;

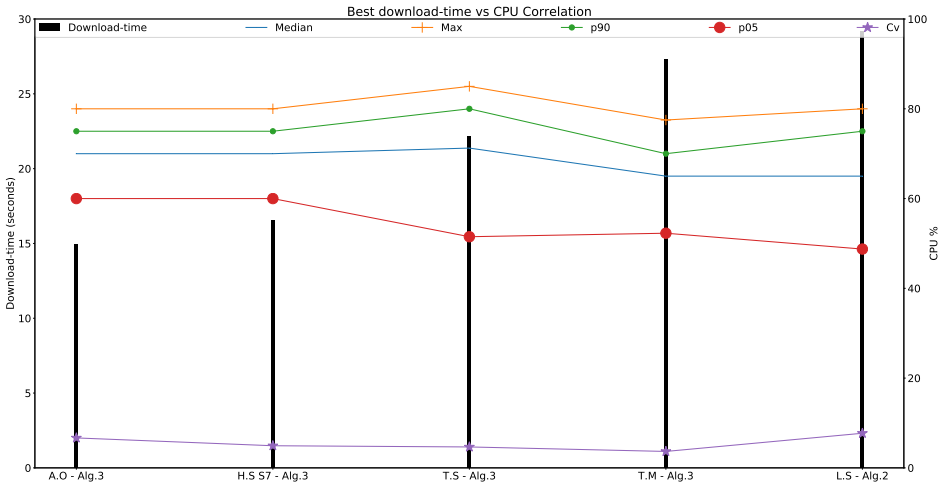


Figure 6.8: Correlation between *best* download time and CPU usage. The thin black bars represents download time. The lines are connected to the secondary y-axis.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
A.O – Algorithm 3	14.94	70.00	6.67	1672.92	80.00	75.00	60.00
H.S S7 – Algorithm 3	16.55	70.00	4.93	1717.96	80.00	75.00	60.00
T.S – Algorithm 3	22.18	71.25	4.66	2352.08	85.00	80.00	51.50
T.M – Algorithm 3	27.29	65.00	3.66	2608.54	77.50	70.00	52.38
L.S – Algorithm 3	29.33	65.00	7.71	2789.17	80.00	75.00	48.75
Correlation	–	*-0.81*	*0.02*	–	*-0.17*	*-0.28*	*-0.93*

Table 6.17: The best results, in regards of download time, from each scenario.

The figure 6.8, and table 6.17 shows that there are a relative strong negative correlation of *-0.81* between CPU-median and download time, and that there is very strong negative correlation of *-0.93* between p05 and download time. When the throughput increases the CPU idle-time, *p05* decreases, and the p50 (median) increases. These calculations indicate that CPU-median and throughput have a relation – the middle and lower part of the usage. However, both the max and p90 number have a very weak correlation with throughput, meaning the CPU is not loaded at maximum. The coefficient of variation (Cv) number does not correlate with throughput, meaning there is no relationship between standard deviation percentages and throughput.

6.4 All Results

This section presents a table with all the results of this chapter.

		CPU % Median					
	Time ↑	p50	c_v	$\int$	Max	p90	p05
H.S S6 – Single-Link Wi-Fi	26.06	56.00	5.86	2185.33	68.00	64.00	47.60
H.S S6 – Single-Link LTE	52.97	50.00	7.71	3739.00	64.00	56.00	40.00
H.S S6 – Naive approach	32.36	56.00	11.60	2601.33	76.00	64.00	40.00
H.S S6 – Algorithm 1	18.35	64.00	0.00	1741.67	72.00	68.40	51.60
H.S S6 – Algorithm 2	18.86	64.00	9.34	1810.33	80.00	72.00	53.80
H.S S6 – Algorithm 3	16.68	68.00	6.20	*1689.67*	80.00	73.60	52.00
H.S S7 – Single-Link Wi-Fi	26.25	60.00	2.61	2334.17	72.50	65.00	52.25
H.S S7 – Single-Link LTE	33.79	60.00	4.45	2800.83	70.00	65.00	45.00
H.S S7 – Naive approach	18.99	70.00	7.47	1948.33	87.50	75.00	49.38
H.S S7 – Algorithm 1	17.88	70.00	2.44	1876.04	80.00	75.00	60.00
H.S S7 – Algorithm 2	18.44	70.00	3.08	1874.54	80.00	74.75	55.00
H.S S7 – Algorithm 3	16.55	70.00	4.93	*1717.96*	80.00	75.00	60.00
L.S – Single-Link Wi-Fi	278.18	32.50	8.11	12101.67	67.50	35.00	20.00
L.S – Naive approach	139.26	35.00	4.58	6902.50	75.00	55.00	20.00
L.S – Algorithm 1	29.63	65.00	6.48	2758.12	80.00	74.25	45.00
L.S – Algorithm 2	29.18	65.00	5.31	*2727.29*	80.00	72.50	48.50
L.S – Algorithm 3	29.33	65.00	7.71	2789.17	80.00	75.00	48.75
T.S – Single-Link Wi-Fi	135.05	37.50	21.46	6015.21	65.00	55.00	0.00
T.S – Single-Link LTE	24.86	70.00	3.48	2417.92	80.00	75.00	45.25
T.S – Naive approach	65.59	45.00	8.11	4016.25	80.00	69.50	0.00
T.S – Algorithm 1	32.91	62.50	7.00	2932.71	80.00	74.00	40.00
T.S – Algorithm 2	26.06	70.00	9.41	2459.79	82.50	76.75	45.00
T.S – Algorithm 3	22.18	71.25	4.66	*2352.08*	85.00	80.00	51.50
T.M – Single-Link Wi-Fi	131.59	42.50	47.55	6794.17	62.50	50.00	4.00
T.M – Single-Link LTE	33.32	60.00	3.84	2911.67	70.00	65.00	48.62
T.M – Naive approach	71.78	45.00	38.85	4185.00	75.00	65.00	2.50
T.M – Algorithm 1	29.16	63.75	5.50	2644.17	75.00	70.00	50.00
T.M – Algorithm 2	29.07	65.00	5.13	2715.62	75.00	70.00	50.00
T.M – Algorithm 3	27.29	65.00	3.66	*2608.54*	77.50	70.00	52.38
A.O – Single-Link Wi-Fi	37.24	55.00	0.00	2873.12	70.00	60.00	48.50
A.O – Single-Link LTE	28.16	65.00	18.62	2523.33	80.00	75.00	45.25
A.O – Naive approach	18.43	70.00	20.98	2229.58	85.00	77.50	48.75
A.O – Algorithm 1	21.76	68.75	5.14	2191.88	82.50	75.00	60.00
A.O – Algorithm 2	17.93	65.00	6.85	1738.33	80.00	72.50	53.50
A.O – Algorithm 3	14.94	70.00	6.67	*1672.92*	80.00	75.00	60.00

Table 6.18: CPU results from all the measurements presented in this chapter. *H.S* = High-Speed, *L.S* = Low-Speed, *T.S* = TechnoWLAN Stationary, *T.M* = TechnoWLAN Moving, *A.O* = AlwaysOnline

		Memory Median		
	Time ↑	KB	c_v	$\int$
H.S S6 – Single-Link Wi-Fi	26.06	13366.00	0.51	504550.67
H.S S6 – Single-Link LTE	52.97	13370.00	0.54	999743.33
H.S S6 – Naive approach	32.36	13532.00	0.06	633393.00
H.S S6 – Algorithm 1	18.35	13856.00	0.18	384648.67
H.S S6 – Algorithm 2	18.86	13844.00	0.15	383212.00
H.S S6 – Algorithm 3	*16.68*	13878.00	0.08	*349072.00*
H.S S7 – Single-Link Wi-Fi	26.25	22259.60	0.70	848489.33
H.S S7 – Single-Link LTE	33.79	22152.00	1.09	1078109.33
H.S S7 – Naive approach	18.99	22442.00	0.83	645245.33
H.S S7 – Algorithm 1	17.88	22700.00	0.73	613168.53
H.S S7 – Algorithm 2	18.44	22738.00	1.04	620259.33
H.S S7 – Algorithm 3	*16.55*	22854.00	0.41	*591980.67*
L.S – Single-Link Wi-Fi	278.18	22176.00	1.05	8671194.00
L.S – Naive approach	139.26	22510.00	0.65	4400334.67
L.S – Algorithm 1	29.63	22760.00	0.80	1004476.83
L.S – Algorithm 2	*29.18*	22832.00	0.76	*971767.67*
L.S – Algorithm 3	29.33	22794.00	1.00	988578.17
T.S – Single-Link Wi-Fi	135.05	22008.00	0.87	4186188.00
T.S – Single-Link LTE	24.86	22004.00	1.52	800471.33
T.S – Naive approach	65.59	22448.00	0.55	2070538.00
T.S – Algorithm 1	32.91	22802.00	1.04	1075317.83
T.S – Algorithm 2	26.06	22836.00	0.91	892175.00
T.S – Algorithm 3	*22.18*	22770.00	0.67	*754807.83*
T.M – Single-Link Wi-Fi	131.59	22228.00	1.13	4135218.00
T.M – Single-Link LTE	33.32	22198.00	1.12	1078302.50
T.M – Naive approach	71.78	22392.00	0.54	2264211.67
T.M – Algorithm 1	29.16	22700.00	0.78	975848.67
T.M – Algorithm 2	29.07	22722.00	0.77	959590.83
T.M – Algorithm 3	*27.29*	22686.00	0.64	*908150.00*
A.O – Single-Link Wi-Fi	37.24	22090.00	1.16	1172761.17
A.O – Single-Link LTE	28.16	22012.00	0.86	888388.00
A.O – Naive approach	18.43	22332.00	0.68	775841.00
A.O – Algorithm 1	21.76	22690.00	1.06	729479.50
A.O – Algorithm 2	17.93	22812.00	1.27	612266.00
A.O – Algorithm 3	*14.94*	22734.00	1.17	*539521.83*

Table 6.19: Memory results from all the measurements presented in this chapter. *H.S* = High-Speed, *L.S* = Low-Speed, *T.S* = TechnoWLAN Stationary, *T.M* = TechnoWLAN Moving, *A.O* = AlwaysOnline

6.5 Summary

This chapter presented the CPU and memory usage results from the same experiments as chapter 5, and had the same isolated and dynamic scenarios structure.

The results from the measurements done in the isolated scenario showed that the two devices consumed a different amount of system memory by running the same experiments. However, memory usage

remained the same for each device in each scenario. It also showed the Samsung S7 consuming slightly more CPU resources. Algorithm 3 proved to consume the least amount of CPU and memory resources across both devices and for both the high-speed and low-speed scenario. Comparing Algorithm 3 to the single-link approaches, the algorithm consumes significantly fewer resources and compared to the naive approach, the algorithm used about 75% of the CPU resources.

In the dynamic scenarios, the same pattern is observed as in the isolated scenario. In these scenarios, Algorithm 3 consumes fewer CPU resources compared to both single-link approaches, and about 65% of the CPU resources compared to the naive approach,

The main takeaway answering the research questions is;

Question 1 – Input parameters The additional input parameters Algorithm 3 uses are; RTT and bandwidth for calculating the lower request-size limits. This proved to not only be important for utilizing the available bandwidth, leading to the lowest download times. It also positively affected CPU and memory resource usage, due to the relationship between median values and download-time producing the usage numbers. Thereby concluding; the additional calculations and memory space Algorithm 3 needs for producing the BDP are insignificant compared to the other multi-link approaches.

Question 2 – Performance The memory median result from the experiments proved to be very similar across all approaches and every scenario. The memory space the approaches reserved during the download were negligible in terms of the available system memory. The CPU median was also relatively similar across all approaches and for every scenario, but a slight increase was observed when the download times decreased. However, compared to the increased throughput which could nearly double, the increase was negligible. Further, none of the devices were at max capacity in terms of CPU percentages, proving that the CPU was affected to an acceptable degree. Thereby concluding; modern smartphones have more than enough CPU and memory reserves to handle multi-link performance, and the increase compared to single-link is minor in terms of the download time improvements.

Chapter 7

Results: Energy usage

This chapter presents the energy usage results from the experiments. Most modern smartphones are powered by a limited energy source – a battery. Therefore, the motivation for monitoring energy usage is; to gather data and analyze how the approaches affect energy usage. In addition to this, this chapter aims to find out if the results from chapter 5 and 6 relates with energy usage, or if there are other factors affecting energy usage.

Unlike the other two result chapters 5 and 6, this chapter starts by introducing a synthetical benchmark giving context to the following energy usage.

The figures in this chapter have a larger x-axis representing the download time compared to the other chapters, this to visualize tail-energy.

Terminology The terms used in this chapter are;

- *Now energy* : representing power over time integrated
- *Peak* : Max value
- *p90* : 90th percentile
- *Time* $\uparrow$: Total download time.
- $\int$: Main and Tail integrals

Further description of the values and the plots in section 3.4.

In the isolated high-speed scenario, both the Samsung S6 and the Samsung S7 devices are presented, but the later scenarios only include the Samsung S7 due to the similar usage pattern of the two devices – section 3.3.

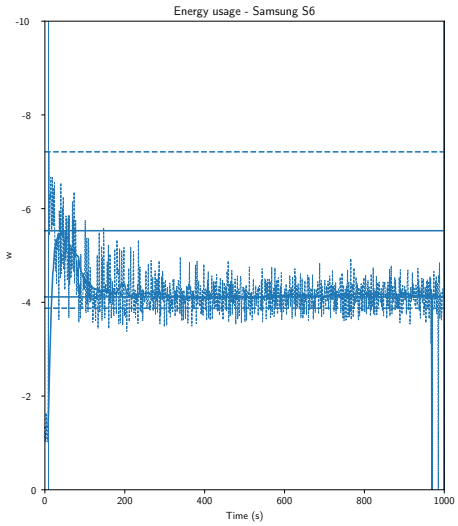
All the figures in this chapter include both avg-energy and now-energy. The tables, however, presents only the now-energy. The reason for this is that the avg-values represents a delayed picture of the actual energy usage.

7.1 Synthetic Benchmark Introduction

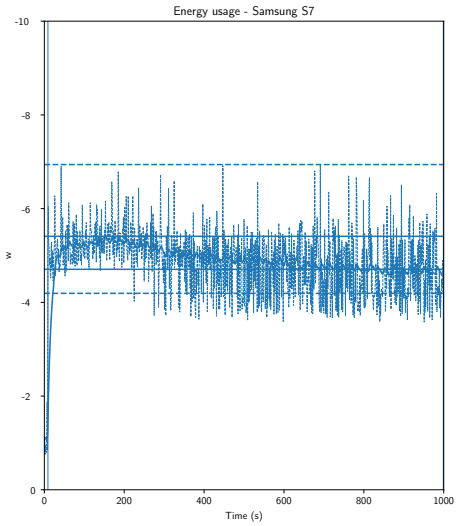
The energy usage numbers presented in this chapter are relative numbers without any full context. The results from the previous chapters are

presented in value of speed or percentages – both absolute. The energy usage presented in this chapter does not describe how the devices are affected, or what they are capable of; therefore, a synthetic test was constructed to measure what energy levels the devices consume at full load and at idle.

For the synthetic max-tests, the devices were configured to use as many system resources as possible, meaning; max screen brightness, max CPU usage on all cores, GPU intensive rendering task, and downloading using the Wi-Fi link. For the idle values, the data was collected from the experiments.



(a) Samsung S6



(b) Samsung S7

	Avg energy		**Now energy**		**Batt. L. (%)**	
	Peak	Median	Peak	Median	Start	End
Samsung S6	5.53	4.15	7.21	4.16	80	61
Samsung S7	5.41	4.89	6.94	4.95	91	73

(c) Energy usage results

Figure 7.1: Energy usage during the synthetic tests for the Samsung S6 and S7 device. The blue lines in the figures display individual measurements. The dotted are "now-energy" and the solid are "avg-energy". The straight lines visualize the median-of-medians.

In figure 7.1a and 7.1b the result from the synthetic tests is displayed, with avg and now-values ranging over 1000 seconds.

Table 7.1c shows peak values, and median values, as well as a battery level (as Batt. L.) not shown in the figures. The battery levels are not perfect because they rely on the battery state of the devices. Both devices lost approximately 20% during these 16minutes, proving that this stress test highly affects the devices. With an energy usage at this level, the devices would have completely drained the battery in approximately 80minutes. The tables show that, in general, the Samsung S7 has a higher energy

potential than the Samsung S6 by 1.28 times. The Samsung S7 is equipped with a 3000mAh battery, compared to the 2550mAh battery in the Samsung S6[1]. This difference calculates to the Samsung S7 having 1.17 times bigger battery and can explain the reason why the device has higher energy potential.

From the extensive measurements done in this chapter, the average idle power level for both devices is calculated to an approximate of *0.7*, creating the median energy scale of 0.7 to 4.16 for the Samsung S6, and 0.7 to 4.95 for the Samsung S7. The similar peak energy scale is 0.0 to 7.21 for the Samsung S6, and 0.0 to 6.94 for the Samsung S7.

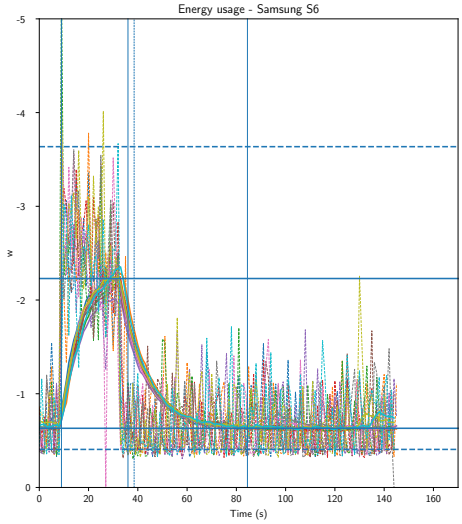
7.2 Isolated Scenarios

The isolated scenarios are presented to gain knowledge into how the devices perform in relative optimal situations. As with the previous result chapters, the focus on energy usage in this isolated scenario is to make sure the devices were in relatively similar conditions.

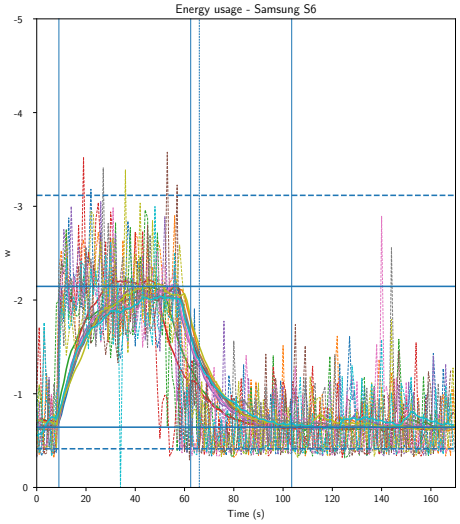
7.2.1 High-Speed Wi-Fi

The energy usage number in the single link measurements stands as a benchmark of what is to be expected as a minimum from the devices. The y-axis in the figures has a limit of -5 for all figures, even if the peak is higher than -5 – section 3.4. The single-link figures 7.2a, 7.2b, 7.2c, and 7.2d visualizes energy usage during single-link download for both devices. Only the Samsung S7 is included in multi-link the figures 7.3a, 7.3b, 7.3c, and 7.3d, however the Samsung S6 multi-link results are shown in the tables after.

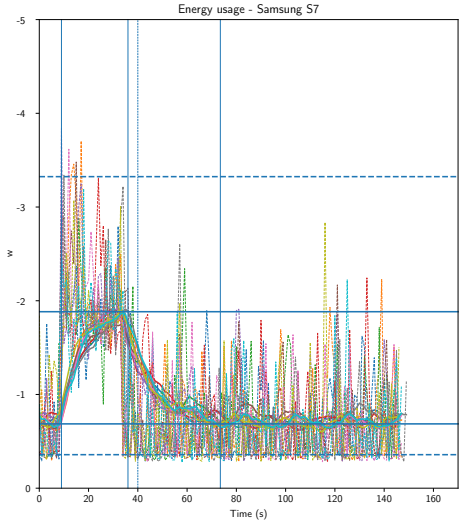
[1] According to www.gsmarena.com



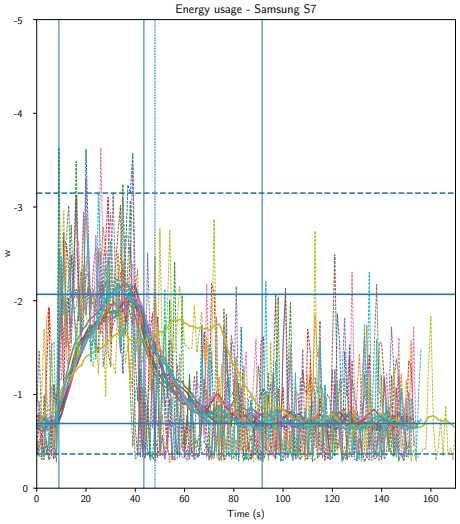
(a) Wi-Fi – Samsung S6



(b) LTE – Samsung S6

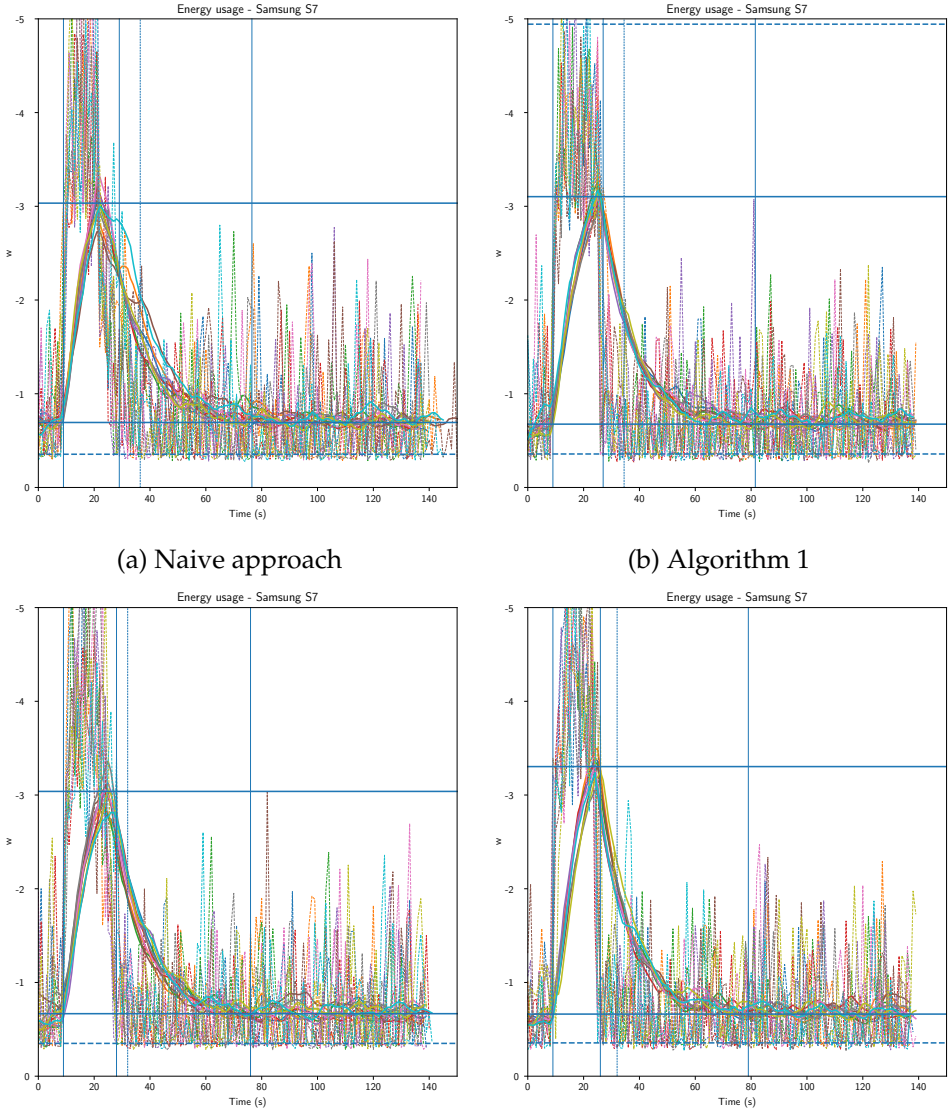


(c) Wi-Fi – Samsung S7



(d) LTE – Samsung S7

Figure 7.2: High-Speed Single-Link energy usage for the Samsung S6 and the Samsung S7 device. The lines represent individual measurements – 10 in total. The dotted lines represent "now-energy", and the solid "avg-energy". The straight horizontal lines are median-of-medians – section 3.4.



(a) Naive approach

(b) Algorithm 1

(c) Algorithm 2

(d) Algorithm 3

Figure 7.3: High-Speed Multi-Link energy usage for the Samsung S7 device. The colorful lines represent individual measurements – 10 in total. The dotted lines represent "now-energy", and the solid "avg-energy". The straight horizontal are median-of-medians.

	Now-energy Medians				
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	26.06	3.64	2.55	58.68	0.98
Single-Link LTE	52.97	3.12	2.38	103.10	4.08
Naive approach	32.36	5.27	4.17	89.16	5.86
Algorithm 1	18.35	6.09	4.69	77.67	2.58
Algorithm 2	18.86	6.67	4.60	77.07	1.05
Algorithm 3	16.68	6.52	5.06	75.53	4.51

(a) Now-energy – Samsung S6

	Now-energy Medians				
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	26.25	3.32	2.18	50.20	1.69
Single-Link LTE	33.79	3.15	2.30	66.83	2.43
Naive approach	18.99	5.60	3.49	58.28	9.05
Algorithm 1	17.88	4.94	3.54	59.82	4.94
Algorithm 2	18.44	5.41	3.64	62.29	2.96
Algorithm 3	16.55	5.36	3.94	62.13	2.87

(b) Now-energy – Samsung S7

Table 7.1: High-Speed energy usage results from the Samsung S6 and S7. The time values in this table are presented in seconds, and only the slowest link time is displayed.

Single-Link Comparing the single-link approaches for the Samsung S6 in table 7.1a, it shows that that the single-link LTE approach uses more energy than the single-link Wi-Fi approach, according to the integral numbers. Not only because of the slower download time of the LTE link but also due to the fact that the peak numbers are slightly lower. The same trend can be observed with the Samsung S7 in table 7.1b. Comparing across the devices, the Samsung S7 uses less energy overall for the single-link approaches.

Multi-Link The numbers in table 7.1b and 7.1a shows a relatively similar peak energy usage between the different algorithms. The small variations in these number are likely accredited to the throughput results. The larges difference in these numbers is visible when comparing the naive approach to the various algorithms integral numbers.

The naive approach peaks at a lower value for the Samsung S6, but because it overall uses more time to download the payload the energy usage is higher than compared to the algorithms. For the Samsung S7 the opposite is true; the naive approach peaks at a higher value, but the energy usage is lower. The reason for these opposites lies in the download time numbers, where the difference between the naive approach and the algorithms for the Samsung S7 is smaller than compared to the Samsung S6.

Compared Comparing the high-speed multi-link approaches in table 7.1b and 7.1a, with the single-link approaches. The tables below focus on the main-integral ($\int$) values presented in these tables, the tail-integral is not included due to the small values for the now-values.

	Now energy		
	Main $\int$	% Wi-Fi	% LTE
S.L Wi-Fi	58.68	–	–
S.L LTE	103.10	–	–
Naive A.	89.16	151.94%	86.48%
Alg.1	77.67	132.36%	75.33%
Alg.2	77.07	131.34%	74.75%
Alg.3	75.53	128.72%	*73.26%*

(a) Samsung S6

	Now energy		
	Main $\int$	% Wi-Fi	% LTE
S.L Wi-Fi	50.20	–	–
S.LLTE	66.83	–	–
Naive A.	58.28	116.10%	87.21%
Alg.1	59.82	119.16%	89.51%
Alg.2	62.29	124.08%	93.21%
Alg.3	62.13	123.76%	*92.97%*

(b) Samsung S7

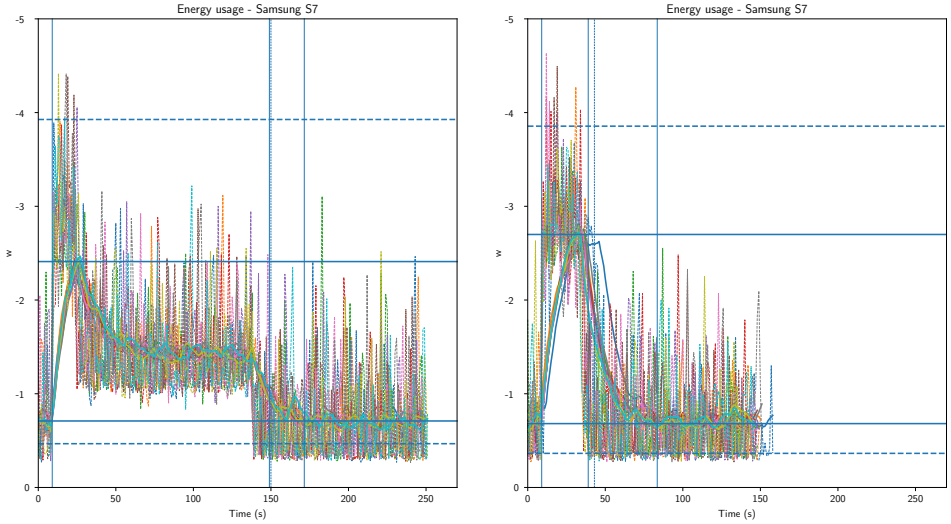
Table 7.2: High-Speed comparison of multi-link energy usage and single-link energy usage in the isolated scenario, for the Samsung S6 and Samsung S7 device.

The Samsung S6 table 7.2a compares the amount of energy, main $\int$, the device uses for the multi-link versus single-link approaches. The calculations show how multi-link can be energy efficient compared to the single-link LTE approach, but compared to the single-link Wi-Fi approach in this scenario there is a slight overhead. The single-link LTE approach used almost 53 seconds to download the payload, while Algorithm 3 used about 17 seconds. This difference means Algorithm 3 used 31% of the time to download the payload while using 73.26% of the energy compared.

The Samsung S7 table 7.2b compares the same measurements as table 7.2a. Where the Samsung S6 device shows an advantage for multi-link over single-link LTE, the Samsung S7 uses slightly more compared with the Samsung S6. The difference between the single-link LTE approach and Algorithm 3 is; Algorithm 3 uses 63% of the time to download, and 92.97% of the energy.

7.2.2 Low-Speed Wi-Fi

In section 6.1.2, the results from the low-speed scenario concluded with a small advantage compared to single-link LTE in regards to CPU and memory usage. Is the same true for energy usage? Figure 7.4a and 7.4b visualizes energy usage for the naive approach and Algorithm 3 in the isolated low-speed scenario. In figure 7.4a the decrease in energy usage after the first Wi-Fi link is done is very visible.



(a) Naive approach (b) Algorithm 3

Figure 7.4: Multi-Link energy usage for the Samsung S7 device in the Low-Speed Wi-Fi, 8Mbps, scenario.

		Now-energy Medians			
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	278.18	3.01	2.12	*375.08*	0.00
Single-Link LTE	33.79	3.15	2.30	66.83	2.43
Naive approach	139.26	3.93	2.61	213.66	0.00
Algorithm 1	29.63	4.08	3.09	78.36	3.34
Algorithm 2	29.18	4.11	3.02	78.77	4.30
Algorithm 3	*29.33*	3.86	2.96	77.48	2.21

Table 7.3: Energy usage results from the Low-Speed Wi-Fi scenario using the Samsung S7 device. The time values presented are from the slowest of the two links and are given in second.

Table 7.3 shows that downloading the payload over a slow single-link Wi-Fi link consumes a significant amount of energy (375.08), due to the low internet bandwidth of this network. The numbers from single-link Wi-Fi range 7.47 times higher compared to the same approach in the high-speed scenario. The naive approach improves upon this, even with the slow Wi-Fi link downloading half the payload.

Comparing the algorithms results in table 7.3, with the single-link LTE the algorithms average at *78.20*, while the LTE is *66.83*. This 4 second, or 85%, the difference represents an increased energy usage of 117.01% by adding a slow Wi-Fi link to a fast LTE link in a multi-link setup – compared to using only the LTE link.

7.2.3 Analysis

Comparing the approaches for both the high-speed and low-speed scenario, we get;

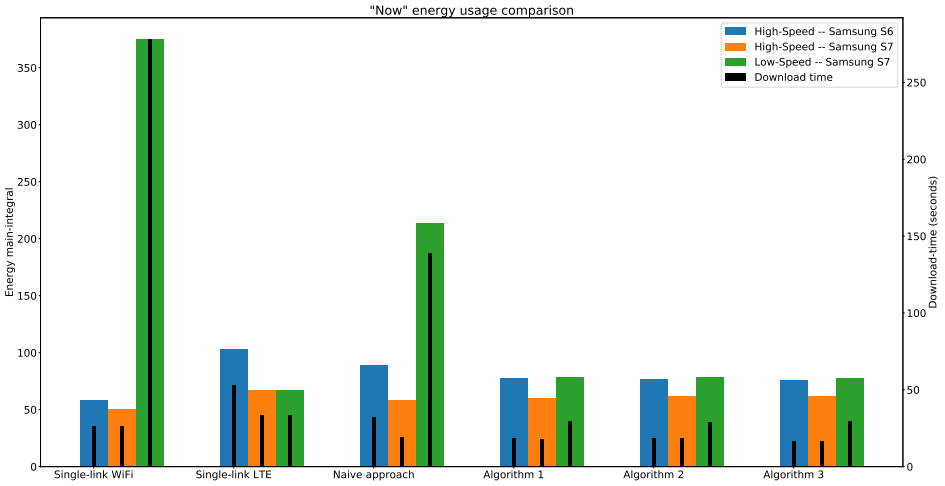


Figure 7.5: Main $\int$, energy usage, comparisons between the approaches and devices for both the high-speed and isolated low-speed scenario. The thin black bars represent download time in the secondary y-axis

Why Figure 7.5 displays the energy usage for all the approaches in both the isolated high-speed and low-speed scenario. The figure shows that the Samsung S7 device uses less energy in the same amount of time compared to the Samsung S6. The figure also shows that even if Algorithm 3 has the lowest download times, compared to Algorithm 1, and 2, the energy-usage differences are not significant. The reason for this is the impact high throughput have on energy-usage. Higher throughput results in higher energy peaks, which further increases the energy potential and could lead to more significant energy integrals. The integral calculations are quadratic, meaning; if one of the axis's increases, the volume have a high probability of increasing as well. A simple example; for each level one axis increases the other needs to decrease by the same amount if the energy usage is to stay the same.

In table 7.1b, for the Samsung S7 device, the peak increased from 4.94 to 5.39 between Algorithm 1, 2, and 3. At the same time, the download time decreased from 17.88 to 16.55. Converting these values to percentages we get; the peak increased by 109% and the download time decreased by; 92%. The relationship between these values is almost entirely quadratic.

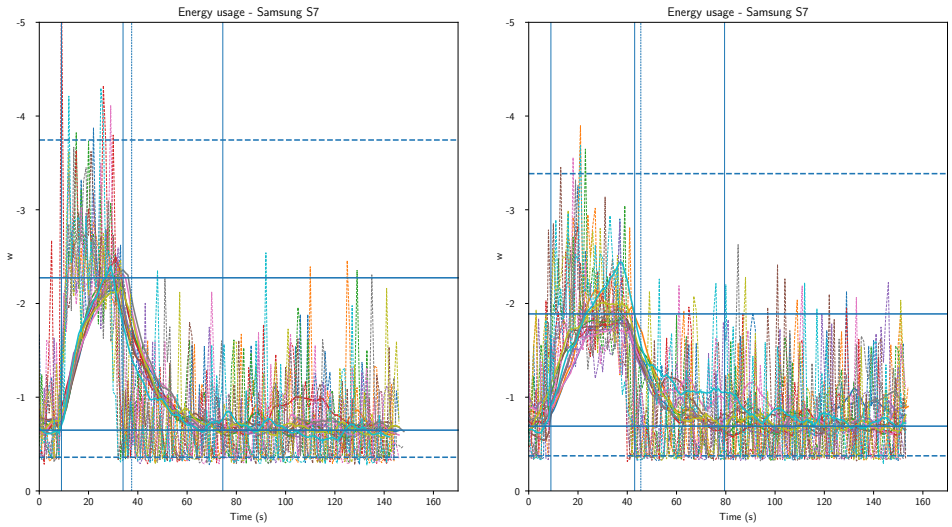
In table 7.1b, for the Samsung S6 device, the same calculation leads to; 107% increase in energy usage between Algorithm 1, 2, and 3, and at the same time a 90% decrease in download time. This indicates that throughput and peak energy correlates.

7.3 Dynamic Scenarios

Energy usage in the dynamic scenarios displays how the device performs in situations where signal strength or other factors play a more significant role than the isolated scenarios. In the dynamic scenarios, only the Samsung S7 device was used.

7.3.1 TechnoWLAN

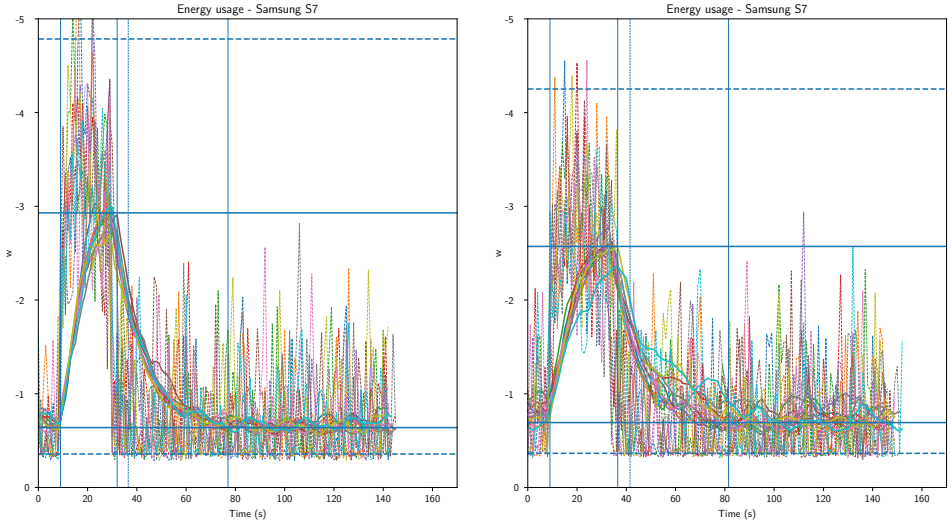
The energy usage shown for the TechnoWLAN experiment displays how the Samsung S7 device, using both single-link, the naive approach, and all the algorithms perform outside an isolated scenario. The TechnoWLAN measurements introduce varying signal strength factors, as well as access point switching. The figure 7.6a, 7.6b, 7.7a and 7.7b visualizes the energy usage for single-link LTE and Algorithm 3 in the stationary and moving TechnoWLAN scenarios.



(a) Stationary – Single LTE

(b) Moving – Single LTE

Figure 7.6: Single-Link energy usage for the Samsung S7 device in the TechnoWLAN scenario.



(a) Stationary – Algorithm 3

(b) Moving – Algorithm 3

Figure 7.7: Multi-Link energy usage for the Samsung S7 device in the TechnoWLAN scenario.

		Now-energy Medians			
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	135.05	2.89	1.93	171.47	0.53
Single-Link LTE	24.86	3.74	2.58	56.47	1.92
Naive approach	65.59	4.55	2.82	111.63	4.05
Algorithm 1	32.91	4.32	3.04	82.82	4.79
Algorithm 2	26.06	4.48	3.29	70.53	5.00
Algorithm 3	*22.18*	4.79	3.39	67.63	2.13

(a) TechnoWLAN Stationary

		Now-energy Medians			
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	131.59	3.02	2.17	185.58	2.80
Single-Link LTE	33.32	3.39	2.15	62.99	0.73
Naive approach	71.78	3.77	2.63	120.80	5.64
Algorithm 1	29.16	3.95	3.03	76.19	3.32
Algorithm 2	29.07	4.28	2.95	73.26	3.32
Algorithm 3	*27.29*	4.25	2.97	68.94	3.00

(b) TechnoWLAN Moving

Table 7.4: Energy usage results form the TechnoWLAN scenario using the Samsung S7 device. The time values are presented in seconds of the slowest link, and the now-energy numbers are a product of volt and ampere.

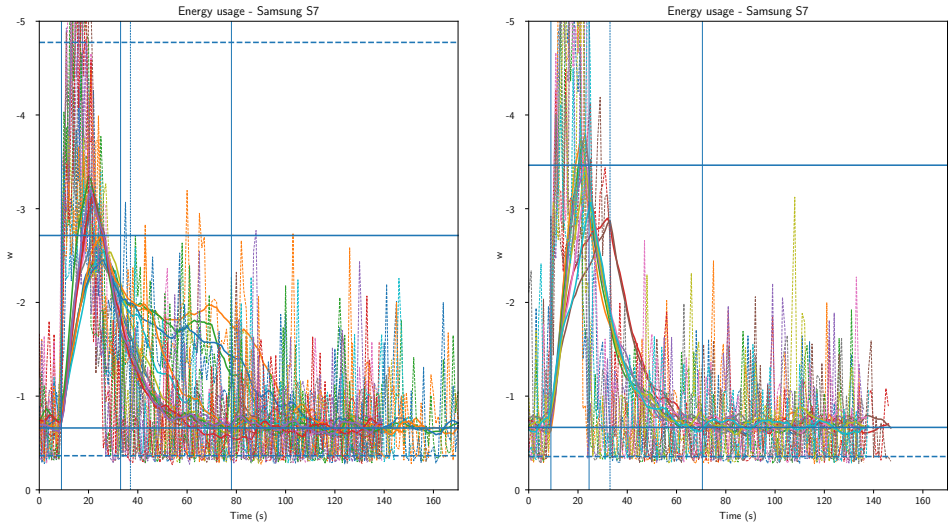
Figures 7.6a and 7.6b shows the energy-usage of single-link LTE in moving and stationary position. Figures 7.7a and 7.7b shows how the two situations, stationary and moving, differ in energy usage using Algorithm

3. The tables 7.4a and 7.4b presents how much energy single-link Wi-Fi uses compared to the rest – for both stationary and moving. In this scenario, with a focus on energy usage, the naive approach is the second worst way of downloading the payload after single-link Wi-Fi.

Further, the tables show that the single-link LTE numbers compared to the algorithms is closer. This is also present in download times, with Algorithm 3 being the fastest of the algorithms. More so, the higher peak values of the algorithms make the slower single-link LTE link more energy efficient over time. This represents an energy usage difference between Algorithm 3 and single-link LTE of; 119.76% increase for the stationary scenario, and 109.44% increase for the moving scenario. Compared with download times, Algorithm 3 uses 89.21% of the time to download in the stationary scenario, and 81.90% in the moving.

7.3.2 AlwaysOnline

The AlwaysOnline measurements differ from the TechnoWLAN measurements, in the way that it represents the scenario of a smaller business or private household. This removes the access point switching, but still, have to deal with volatile LTE throughput and capacity of this scenario.



(a) Naive approach

(b) Algorithm 3

Figure 7.8: Multi-Link energy usage for the Samsung S7 device in the AlwaysOnline scenario.

		Now-energy Medians			
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
Single-Link Wi-Fi	37.24	3.26	2.25	67.70	1.08
Single-Link LTE	28.16	3.24	2.25	53.15	3.15
Naive approach	18.43	4.77	3.11	55.37	7.05
Algorithm 1	21.76	5.59	3.32	64.37	6.74
Algorithm 2	17.93	6.09	3.99	60.93	2.71
Algorithm 3	*14.94*	6.08	4.14	55.75	9.69

Table 7.5: Energy usage results from the AlwaysOnline scenario using the Samsung S7 device.

Table 7.5 shows that like the TechnoWLAN results, downloading with a relatively slow Wi-Fi link, compared with a high performing LTE link, the Wi-Fi link uses more energy.

The table presents a difference in over 13 seconds between the single-link LTE measurements and Algorithm 3 measurements. This difference means Algorithm 3 uses 53.05% of the time to download compared to single-link LTE. However, even with the difference in download times between the two, the high peak values of Algorithm 3 makes the algorithm uses more energy comparing the main integral numbers. This difference amounts to a 104.89% increase in energy usage for Algorithm 3, compared to the single-link LTE link.

Comparing with the relativity slow Wi-Fi link of Algorithm 3, it shows that the algorithm uses less energy than the slow single-link Wi-Fi measurements. This difference, between single-link Wi-Fi and Algorithm 3, means the algorithm uses 82.34% of the energy, and about 40% of the time download the payload.

7.3.3 Analysis

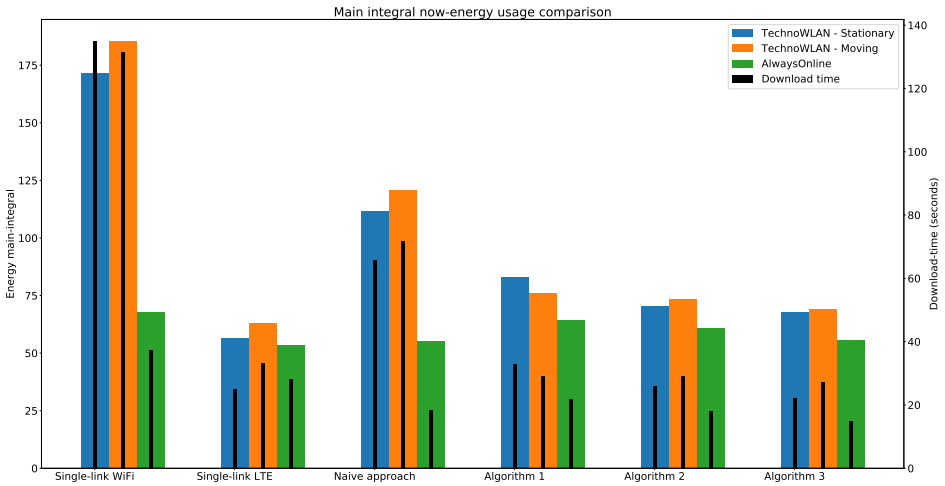


Figure 7.9: Energy usage comparisons between the approaches and devices. Each bar group shows the various approaches, and the black center bars represent download time on the secondary y-axis to the right.

In the high-speed isolated scenario 7.2, the difference between the algorithms and the naive approach were almost insignificant for the Samsung S7. Compared to the results in section 7.2.3, the energy differences in the dynamic scenarios are more significant.

Why Figure 7.9 shows that there is a strong relationship between energy usage and download time for the TechnoWLAN moving scenario. In the isolated scenario, Algorithm 1 did not consume significantly more energy, but in the dynamic scenario where Algorithm 1 has a noticeable slower download time, the differences in energy usage are more visible. This strengthens the relationship between time and energy usage. It also shows that even if Algorithm 2 has a higher download time than Algorithm 3 it has almost the same energy footprint. The isolated scenario concluded that Algorithm 3 used about as much energy as Algorithm 2, even if the download times differed. In the dynamic scenario, the same pattern is observed. Algorithm 3 has the highest throughput but still consumes almost as much energy as Algorithm 2.

7.4 Analysis

Like CPU and memory, energy usage is a resource metric. The motivation for monitoring this metric is to analyze how the devices are affected with limited battery resources – thereby answering the second research question. The scenario-analysis in section 7.2.3 and 7.3.3 proved that Algorithm 1, 2, and 3 used about the same amount of energy due to the increase energy

peaks and decreased download times. This analysis uses all the results in this chapter, and answer the following questions;

7.4.1 Is there a correlation between peak energy and throughput?

Using the total download time as an indicator of high throughput, the tables in this chapter shows that the best download time is the AlwaysOnline scenario with Algorithm 3 using only 14.94 seconds to download the payload – table 7.5. The next best result is the isolated high-speed scenario with the Samsung S7 and Algorithm 3 using 16.55 seconds – table 7.1b. Compiling all the multi-link "best results" from the Samsung S7 into a sorted table, we get;

		Now energy	
	Time ↑	Peak	p90
A.O – Algorithm 3	*14.94*	6.08	4.14
H.S S7 – Algorithm 3	16.55	5.36	3.94
T.S – Algorithm 3	*27.29*	4.25	2.97
T.M – Algorithm 3	*22.18*	4.79	3.39
L.S – Algorithm 2	29.18	4.11	3.02
Correlation	–	*-0.97*	*-0.82*

Table 7.6: Best download times compared between all the scenarios.

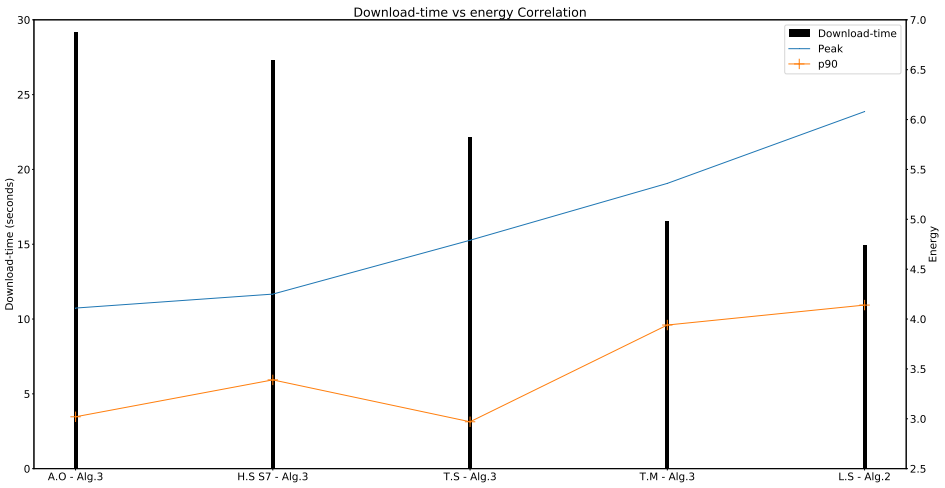


Figure 7.10: Download time vs. peak energy correlation. The black bars represent download time in the left y-axis, and the plots represent peak energy in the right y-axis. Please note that the peak energy level starts a 2.5 not 0. This is done to better visualize the progressive increase in peak energy.

The values in table 7.6, shows a correlation of; *-0.97* for the peak values, and -0.82 for the p90 values – section 3.4. This, almost, perfect negative

correlation between download time and peak energy, and strong negative correlation for the p90 values is visible in figure 7.6, and concludes that the energy potential plays a significant role in download times.

Not only is the correlation strong, but also seems to be a progressive increase in peak energy when the download time decreases. Example; the download time difference between the "AlwaysOnline Algorithm 3", and "High-Speed Algorithm 3" is 1.61 seconds, while the difference in peak energy is 0.72. Converted to percentages results in; 90.27% download time and 113.43% peak energy. The same calculations between the "High-Speed Algorithm 3" and "TechnoWLAN Stationary Algorithm 3", results in; 60.64% download time and 126.11% peak energy increase.

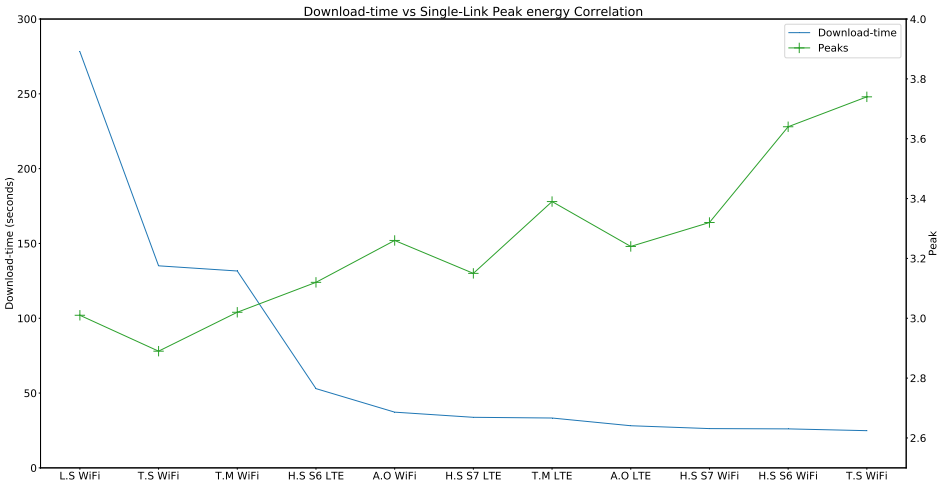
Section 7.2.3 speculated that there was a relationship between peak energy and download time. This correlation strengthens the speculation. The previously observed relationship between CPU and throughput means that the progressively increased energy usage probably is caused by the communication hardware, not CPU.

7.4.2 What would the single-link approaches consume with the same throughput?

In the "best results" section 7.4.1, the correlation between peak energy and download time were proven to be strong. The scenarios used in this thesis did not include a Wi-Fi network with the same bandwidth of the aggregated total bandwidth of the multi-link approaches. The LTE networks used did have high bandwidth in some cases but are difficult to predict. With the observed progressive peak energy levels when the download times decrease, how would the single-link approaches perform with the same bandwidth?

In the high-speed isolated scenario, the results from the single-link Wi-Fi and LTE download time for the Samsung S7 is; 26.25 seconds ad 33.79 seconds. This means the single-link Wi-Fi uses 77.7% of the time to download compared to the LTE link. The energy peaks were; 3.32 for Wi-Fi and 3.15 for LTE. Converted to percentages results in a; 105% increase in peak energy. For the Samsung S6, the same calculations are; 26.06 seconds for the Wi-Fi and 52.97 seconds for the LTE. This means the Wi-Fi used; 49.2% of the download time compared to LTE. The peaks are 3.64 for the Wi-Fi and 3.12 for the LTE and calculate to a 116.7% increase in peak energy. This proves that the energy peaks are affected by download time, also for single-link.

Using the results from all the single-link measurements, we get;



(a) Single-Link download times and peak values. The first blue plot represents download time in the left y-axis, and the green plot represents peak energy in the secondary right y-axis. Please note that the secondary y-axis starts at 2.5.

Scenario	Download time	Peak
L.S – S7 – Wi-Fi	278.18	3.01
T.S – S7 – Wi-Fi	135.05	2.89
T.M – S7 – Wi-Fi	131.59	3.02
H.S – S6 – LTE	52.97	3.12
A.O – S7 – Wi-Fi	37.24	3.26
H.S – S7 – LTE	33.79	3.15
T.M – S7 – LTE	33.32	3.39
A.O – S7 – LTE	28.16	3.24
H.S – S7 – Wi-Fi	26.25	3.32
H.S – S6 – Wi-Fi	26.06	3.64
T.S – S7 – LTE	24.86	3.74

(b) Single-Link download times and peak values

Table 7.7: Single-Link download times and peak values

Figure 7.7a visualizes the relation between the download times and peak values. A correlation calculation confirms this and results in a negative correlation of; -0.62. The reason for this comparatively low correlation number is the increased peak energy at the right part of figure7.7a. The figure visualizes, in the right part, that a slight decrease in download times affect the peak energies. Using the four last values in table 7.7b shows a decrease of 3.3 seconds and an increase in peak energy of; 0.5. Assuming that for every second the download times decreases, the peak energy increases by 0.15 (0.5/3.3), the calculated value for a single-link download of 14.86 seconds would be 5.26 and comparing with Algorithm 3 from the AlwaysOnline scenario with a download time of 14.94 and peak energy of 6.08. This puts Algorithm 3 115.58% above the theoretical single-link result with a similar download time.

Comparing single-link and multi-link results with similar download time None of the single-link results match the download times of the best multi-link results. However, comparing the most similar, in regards to download time, single-link and multi-link results, we get;

		Now-energy Medians	
	Time ↑	Peak	p90
H.S S7 – Single-Link LTE	33.79	3.15	2.30
T.M – Algorithm 1	32.91	4.32	3.04
H.S S7 – Single-Link Wi-Fi	26.25	3.32	2.18
T.M – Algorithm 2	26.06	4.48	3.29

Table 7.8: Two examples of single-link and multi-link results with similar download times. The first group averages around 33 seconds, and the second around 26 seconds.

Table 7.8 shows two examples of similar single-link and multi-link download times. The results is compared based on download times only, not focusing on which scenario the results are from. These results disprove the assumption that the energy peaks would be similar if the single-link results would match that of the multi-link download times. The two groups in the table show an average download time difference of 0.53 seconds (101.69%), with an energy-peak increase of 1.16 (135.23%), and an average main integral increase of 19.83 (142.17%). This difference in energy consumption indicates that there is something else going on. Are the two scenarios comparable?

7.4.3 What are the energy usage differences between the best single-link and the best multi-link approaches?

Answering this question needs the best single-link results, and the best multi-link algorithm results from each scenario. Therefore, the selected single-link can vary from Wi-Fi and LTE in the calculations, because the different scenarios performed differently. The reason behind this selection is the assumption that a user would use the link with the highest throughput for single-link download, being Wi-Fi or LTE.

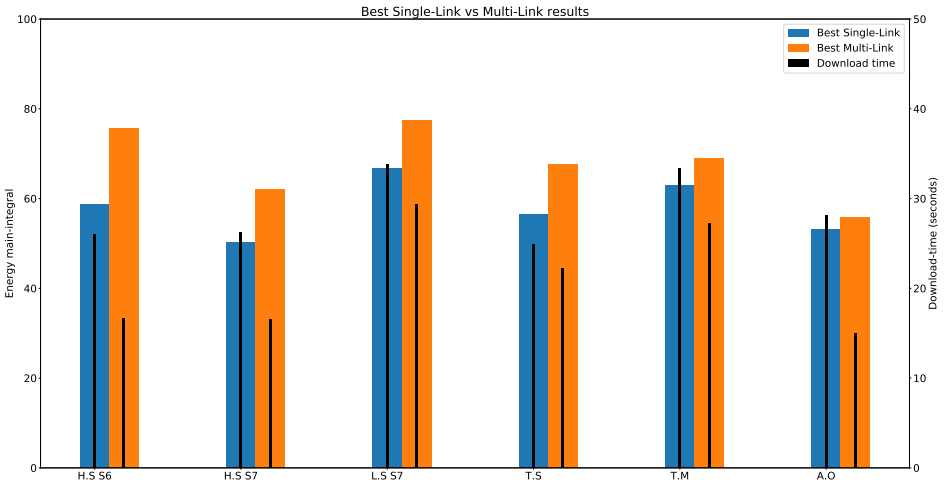


Figure 7.11: The best single-link and multi-link approach from each scenario with the lowest download times. The bars show main integral energy usage, and the thin center black bars represent download time in the secondary y-axis.

Figure 7.11 shows that for each scenario, the best single-link approach appear to consumes less energy than the best multi-link approach, even if the download times are lower for the multi-link approach. The scenario with the least difference in energy usage is the AlwaysOnline (A.O) where the difference in is 104.89%, and the second least is the TechnoWLAN Moving with 109.44%. The scenario with the most significant difference between the two is the High-Speed isolated scenario using the Samsung S6 devices, followed by the same scenario using the Samsung S7 device. However, do these difference matter?

Rate The scenario with the least difference in energy usage, is also the scenario with the most significant difference in download times. In the AlwaysOnline scenario, Algorithm 3 uses 53.05% of the time to download, and have an increased energy peak of 187.65%. Compared to the second scenario with similar energy usage results, the TechnoWLAN Moving scenario, Algorithm 3 used 81.90% of the time to download compared to single-link LTE, and have an increased energy peak of 131.17%. Using these numbers for all the scenarios and calculating the rate between them, we get;

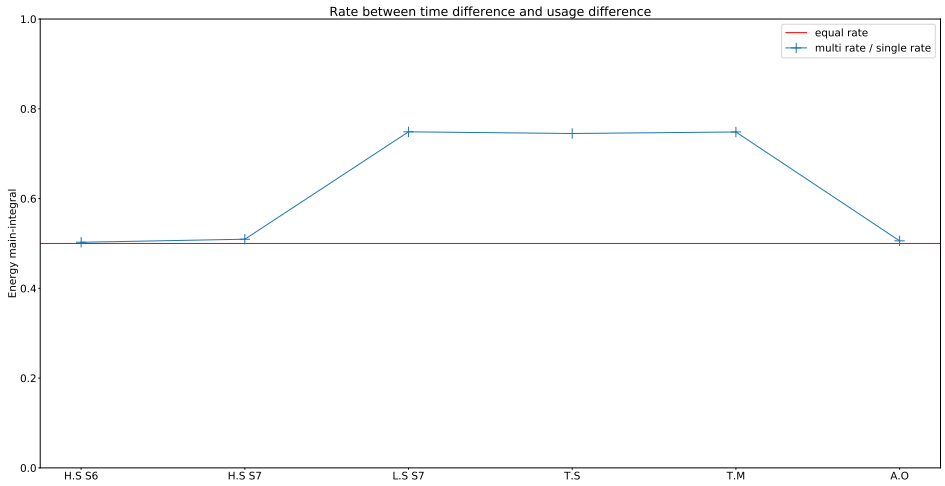


Figure 7.12

Figure 7.12 shows the rate between time and energy usage, and between the best single-link and multi-link. The calculations are as follows;

$$\frac{\frac{singlelink_{downloadtime}}{singlelink_{energyusage}}}{\frac{multilink_{downloadtime}}{multilink_{energusage}}}$$

Meaning; a value close to 0.5 means the multi-link approach have used about the same amount of energy, and about half the time to download. A value above 0.5 means the multi-link approach have used more energy compared to the improved download time, and a number below 0.5 would mean the multi-link approach have used less energy and improved download time. Figure 7.12 reveals a pattern separating the scenarios into two groups;

The first group is the results close to 0.5, consisting of the High-Speed isolated scenario for both devices, and the AlwaysOnline scenario. The scenarios in this first group have improved the download time and consume about the same amount of energy accordingly.

The second group is the results well above 0.5, consisting of the Low-Speed isolated scenario and the two TechnoWLAN dynamic scenarios. The scenarios in this second group, show values of about 0.74 for all of them, meaning the improved download time of the multi-link approach was not worth it in comparison with the increased energy usage.

The most apparent pattern explaining these groups is the fact that all the networks in the second group have a significant difference in bandwidth capacity.

In the first group, the average throughput for Algorithm 3 is; 70.01Mbps and 65.37 for the Wi-Fi link and the LTE link respectively. The average throughput in the second group for Algorithm 3 are; 12.28Mbps and 71.92Mbps for the Wi-Fi link and the LTE link, respectively. This calculates to the first group having a difference in link throughput rate of 0.17, and

the second group 1.07. Therefore, multi-link approaches in scenarios with equal link-bandwidth perform better in regards to energy usage. However, why do these link-bandwidths differences matter?

Why Why is the second group unable to perform in regards to energy usage? The bandwidth differences should not have a significant influence? Are Algorithm 3 unable to utilize the different link-bandwidth, or is there something else going on?

An example confirming Algorithm 3 can utilize the aggregated bandwidths in a second-group scenario; in the isolated low-speed scenario, the aggregated throughput was 74.19Mbps (7.49Mbps + 66.70Mbps) for Algorithm 3. This aggregated throughput means the algorithm should have downloaded the payload in about 26.96seconds. The time Algorithm 3 used for the download were 29.01seconds. Accounting for overhead in the TCP connection, this difference is not significant enough to explain the reason why. The Link Δ for Algorithm 3 in this scenario, was 0.32 seconds and is also not the reason why the energy usage is so high. Concluding that, in fact, Algorithm 3 is able to perform in at least one of the second-group scenarios.

Ruling out the effectiveness of the algorithm, a possible explanation for this could be the relatively high energy peak of the isolated low-speed single-link Wi-Fi measurement. A fundamental theory of physics says;

$$W = \vec{F} * \vec{s}$$

Figure 7.13: Work and energy law

Using this theory, and say that W (Work) is the energy used or consumed, $\vec{F}$ (Force) is the energy peaks, and $\vec{s}$ (distance) is the download time. Therefore, if the energy usage is to stay the same, the energy peaks and the download times need to relate to each other. However, what happens if they do not, and the energy peaks do not go below a certain level?

In the isolated low-speed scenario, the bandwidth was restricted to 7.49Mbps. Still, the measured energy peak was 3.01. By comparison, the energy peak in the AlwaysOnline single-link Wi-Fi measurement was 3.26, with a throughput of 57.28Mbps. These two scenarios cannot consume the same amount of energy with the theory above. The small difference in peak-energy compared with a vast difference in throughput, seem to indicate that using the Wi-Fi link includes a certain amount of minimum energy regardless of what the throughput is. Figure 7.7b display this trend, and concludes that this could be the reason why measurements in scenarios with low bandwidth Wi-Fi networks use more energy than what is expected. Therefore, adding a slow network link means the link have to perform above a certain level to overcome this base energy level.

Minimum throughput To answer what minimum throughput a link have to provide in order not to degrade performance in regards to energy, we present single-link measurements and the energy peaks. All the single-link measurements are included, both Wi-Fi and LTE for all scenarios.

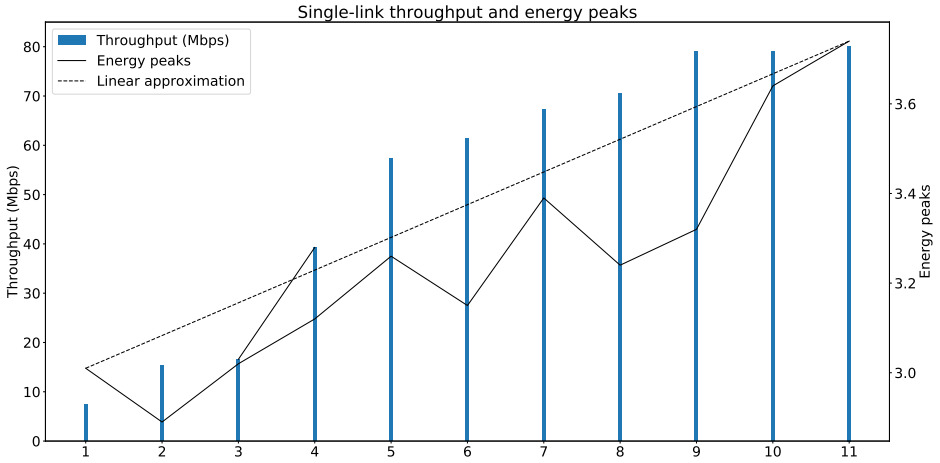


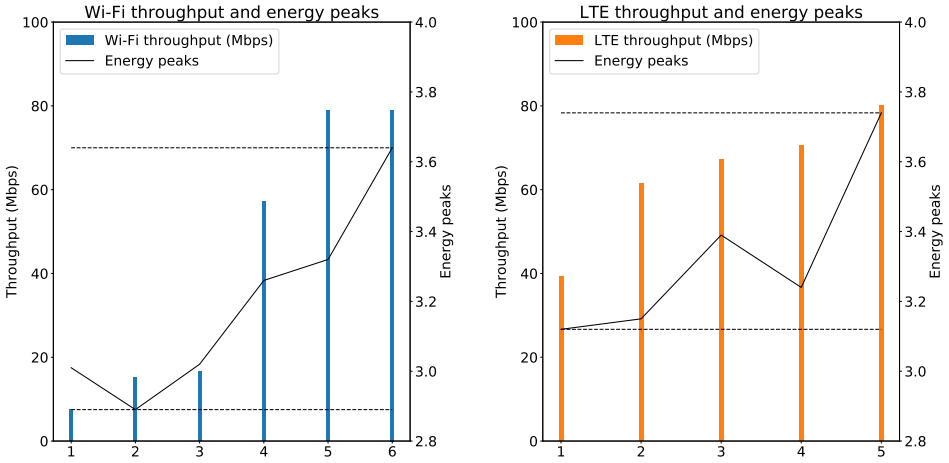
Figure 7.14: Single-link throughput values compared to the energy peaks. The blue columns are throughput values, and the black lines are peaks. The dotted black line are an linear approximation between the lowest value 3.01 and the highest value 3.74.

The two straight black lines in figure 7.14 shows an estimation of what the recommended link bandwidth needs to be. The three leftmost blue columns are the results from the second-group 7.4.3, so we place them below the linear approximation. The first blue column above the linear approximation is from the first group and is the single-link LTE measurement from the isolated high-speed scenario for the Samsung S6. The solid black line between blue column 3 and 4 is crossing the linear approximation at about 30Mbps.

Therefore, the suggested minimum bandwidth for any link in a multi-link setup on these devices needs to surpass 30Mbps bandwidth.

7.4.4 Which link consume the least amount of energy?

Answering the question of which link consumes the least energy, all the single-link measurements are used in comparison with the energy peaks.



(a) Wi-Fi (b) LTE

Figure 7.15: Single-Link Wi-Fi and LTE compared with the energy peaks.

Figure 7.15a and 7.15b show the different single-link measurements from all the scenarios. The measurements are labeled with numbers because the scenario is not important for this analysis. When the two links produce about the same throughput (79.01 and 80.14Mbps) at the right of the figure, the energy peaks differ by 3.64 and 3.74 for the Wi-Fi link and the LTE link, respectively. This indicates that high throughput LTE links will consume more energy over time because of the higher energy peak.

None of the scenarios used in this thesis included an LTE network with less than 39.28Mbps; therefore, the second data point of comparison is at around the 60Mbps mark. At this mark, the Wi-Fi link had an energy peak of 3.26, and the LTE link 3.15, indicating that the Wi-Fi link will consume more energy.

Averaging the throughput and energy peaks for the Wi-Fi link results in a throughput of 42.43Mbps and an energy peak of 3.19. The LTE link had an average throughput of 63.70Mbps and an average energy peak of 3.32. Converting these average throughput numbers into download times results in; the Wi-Fi link had an average download time of 105.80 seconds and LTE 34.62 seconds.

Inserting these values into figure 7.13, we get;

$$\text{Wi-Fi avg. energy} : 3.19 * 105.72 = 337.24$$

$$\text{LTE avg. energy} : 3.32 * 34.62 = 114.93$$

The calculations above indicate that the LTE link is far more effective in terms of energy usage. However, section 7.4.3 concluded that the links must reach a throughput above 30Mbps in order to be energy efficient, and because all the data points from the LTE measurements are above this threshold, the comparison is uneven. Removing the data points from the Wi-Fi measurements below 30Mbps we get an average energy peak of

3.40 and an average download time of 29.85 seconds. Also removing three data points, with the highest download times, from the LTE measurements, resulting in;

$$\text{Wi-Fi avg. energy} : 3.40 * 29.85 = 101.49$$

$$\text{LTE avg. energy} : 3.45 * 28.78 = 99.29$$

With the new calculation for the Wi-Fi and LTE link, the conclusion is that the links consume about the same amount of energy at the upper end of the tested throughput values in this thesis.

7.4.5 How do the energy usage affect the devices?

This analysis answers how the specific devices, the Samsung S6, and S7, are affected by the various algorithms and approaches by calculating how much energy the device was used compared to what it is able ut use in the same amount of time – the "Max $\int$" column.

With the synthetic benchmark numbers presented in section 7.1, a table can be constructed describing the overall impact in percentages the different single-link and, multi-link configurations have on the devices. The calculations use the results from the isolated scenario. For simplicity, the max-integral (Max. $\int$) calculations in this comparison are calculated by using the median values from the table 7.1c and multiplying by the time given.

The "Max $\int$" calculations in Samsung S6 table are a product of time and the *4.16* now-value found in table 7.1c. The same calculations for the Samsung S7 uses the value *4.95* found in that table 7.1c.

		Now energy		
	Time ↑	Main $\int$	Max $\int$	Utilization
Single-link Wi-Fi	*26.06*	58.68	108.40	54.13%
Single-link LTE	*52.97*	103.10	220.35	46.78%
Naive approach	*32.36*	89.16	134.61	66.23%
Algorithm 1	*18.35*	77.67	76.33	101.75%
Algorithm 2	*18.86*	77.07	78.45	98.24%
Algorithm 3	*16.68*	75.53	69.38	108.86%

(a) Samsung S6

		Now energy		
	Time ↑	Cur $\int$	Main $\int$	Utilization
Single-link Wi-Fi	*26.25*	50.20	129.93	38.63%
Single-link LTE	*33.79*	66.83	167.26	39.95%
Naive approach	*18.99*	58.28	94.00	62.00%
Algorithm 1	*17.88*	59.82	88.50	67.59%
Algorithm 2	*18.44*	62.29	91.27	68.24%
Algorithm 3	*16.55*	62.13	81.92	75.84%

(b) Samsung S7

Table 7.9: Energy usage comparisons between the High-Speed results from the isolated scenario and the synthetic max-usage test from table 7.1c for the Samsung S6 and Samsung S7 device.

Table 7.9a and 7.9b shows the amount of energy the various approaches used as "Main $\int$" compared to what the device could have used in the same amount of time if it maxed out all resources – referred to as "Max $\int$". The "Util.%" is an abbreviation for "utilization-in-percent" and are calculations based on what the Main $\int$ is compared to the Max $\int$.

Samsung S6 The table 7.9a for the Samsung S6, shows that the difference between the utilization percentages for single-link downloads are 7.35% even if the time difference is about double. Both single-link approaches utilize about 50% of the energy usage capabilities. The utilization percentages for Algorithm 1, 2, and 3 are about equal and reaching close to and over 100%. These numbers mean that the energy-impact on the Samsung S6 for the algorithms are very significant.

Samsung S7 The utilization percentages of the Samsung S7, in table 7.9b, for single-link are almost equal, but compared to the same percentages for the Samsung S6, they are lower. The table shows that the utilization percentages, for the multi-link approaches, are well below 100%, averaging at around 70% for the algorithms. Further, the algorithms utilization numbers average at about 30% higher than the single-link values. Concluding that the Samsung S7 device is more suited for multi-link applications compared to the Samsung S6 device, which uses most of its potential at the same task.

7.4.6 Why does the Samsung S7 consume less energy compared to the Samsung S6 device?

Table 7.1a for the Samsung S6, and table 7.1b for the Samsung S7 shows that there is a difference in peak now-energy. Focusing in on the best performing Algorithm 3, the difference in total-download time is 0.13 seconds. This minor difference in download time indicates that the devices gained approximately the same throughput for both links. The throughput tables 5.1a and 5.1b shows that the difference in LTE throughput between the devices, for the Algorithm 3 is; 1.79Mbps in favor of the Samsung S7 device, and for the Wi-Fi link; 0.01Mbps in favor of the Samsung S6 device. Therefore, it is safe to conclude that the compared capabilities between the devices are similar.

However, peak now-energy is not. The tables above show a difference in peak now-energy of 1.16 between the devices using Algorithm 3. This difference amounts to the Samsung S6 having a 121.64% higher peak now-energy. Comparing further, between Algorithm 1, 2, and 3, the average shows the Samsung S6 having a 122.88% higher peak now-energy.

The differences in energy usage, in this particular case, must be more than an environmental anomaly. According to Samsung, the S7 device uses an "Exynos 8 Octa (8890)" [12] multi-feature chipset which integrates the LTE modem. The S6 device, on the other hand, uses an "Exynos 7 Octa (7420)" [11] chipset. Both these chipsets, according to Samsung, boost performance and efficiency from its predecessors, with descriptions like; "unmatched performance and power efficiency". The 8890 chipsets found in the Samsung S7 device claiming 10% improved power efficiency.

Therefore, it is probable that the Samsung S7 devices use less energy compared to the Samsung S6 because of these hardware chip-sets.

7.5 All Results

This section presents a table with all the measurements results from this chapter.

		Now-energy Medians			
	Time ↑	Peak	p90	Main $\int$	Tail $\int$
H.S S6 – Single-Link Wi-Fi	26.06	3.64	2.55	58.68	0.98
H.S S6 – Single-Link LTE	52.97	3.12	2.38	103.10	4.08
H.S S6 – Naive approach	32.36	5.27	4.17	89.16	5.86
H.S S6 – Algorithm 1	18.35	6.09	4.69	77.67	2.58
H.S S6 – Algorithm 2	18.86	6.67	4.60	77.07	1.05
H.S S6 – Algorithm 3	16.68	6.52	5.06	75.53	4.51
H.S S7 – Single-Link Wi-Fi	26.25	3.32	2.18	50.20	1.69
H.S S7 – Single-Link LTE	33.79	3.15	2.30	66.83	2.43
H.S S7 – Naive approach	18.99	5.60	3.49	58.28	9.05
H.S S7 – Algorithm 1	17.88	4.94	3.54	59.82	4.94
H.S S7 – Algorithm 2	18.44	5.41	3.64	62.29	2.96
H.S S7 – Algorithm 3	16.55	5.36	3.94	62.13	2.87
L.S – Single-Link Wi-Fi	278.18	3.01	2.12	375.08	0.00
L.S – Naive approach	139.26	3.93	2.61	213.66	0.00
L.S – Algorithm 1	29.63	4.08	3.09	78.36	3.34
L.S – Algorithm 2	29.18	4.11	3.02	78.77	4.30
L.S – Algorithm 3	*29.33*	3.86	2.96	77.48	2.21
T.S – Single-Link Wi-Fi	135.05	2.89	1.93	171.47	0.53
T.S – Single-Link LTE	24.86	3.74	2.58	56.47	1.92
T.S – Naive approach	65.59	4.55	2.82	111.63	4.05
T.S – Algorithm 1	32.91	4.32	3.04	82.82	4.79
T.S – Algorithm 2	26.06	4.48	3.29	70.53	5.00
T.S – Algorithm 3	*22.18*	4.79	3.39	67.63	2.13
T.M – Single-Link Wi-Fi	131.59	3.02	2.17	185.58	2.80
T.M – Single-Link LTE	33.32	3.39	2.15	62.99	0.73
T.M – Naive approach	71.78	3.77	2.63	120.80	5.64
T.M – Algorithm 1	29.16	3.95	3.03	76.19	3.32
T.M – Algorithm 2	29.07	4.28	2.95	73.26	3.32
T.M – Algorithm 3	*27.29*	4.25	2.97	68.94	3.00
A.O – Single-Link Wi-Fi	37.24	3.26	2.25	67.70	1.08
A.O – Single-Link LTE	28.16	3.24	2.25	53.15	3.15
A.O – Naive approach	18.43	4.77	3.11	55.37	7.05
A.O – Algorithm 1	21.76	5.59	3.32	64.37	6.74
A.O – Algorithm 2	17.93	6.09	3.99	60.93	2.71
A.O – Algorithm 3	*14.94*	6.08	4.14	55.75	9.69

Table 7.10: Now-energy usage results from all the measurements in this chapter. *H.S* = High-Speed, *L.S* = Low-Speed, *T.S* = TechnoWLAN Stationary, *T.M* = TechnoWLAN Moving, *A.O* = AlwaysOnline

7.6 Summary

This chapter presented the energy usage results from the same experiments as presented in chapter 5 and 6.

In the isolated high-speed scenario, Algorithm 3 consumed 75.53% of energy compared to the naive approach for the Samsung S6, and 62.13% for the Samsung S7. Compared to the most effective single-link

download, being the Wi-Fi in that scenario, Algorithm 3 consumed 128.72% of the energy for the Samsung S6 and 123.76% for the Samsung S7. The same comparison between the single-link LTE download and Algorithm 3, revealed that the algorithm consumed 73.25% for the Samsung S6, and 92.96% for the Samsung S7.

In the dynamic scenarios, Algorithm 3 consumed on average 72.33% compared to the naive approach. This difference proves that Algorithm 3 gains more in dynamic scenarios than the naive approach.

The main takeaway answering the research questions is;

Question 1 – Input parameters For the first time, it is observed that in some cases using single-link is slightly more effective in terms of energy usage compared to Algorithm 3. Thereby concluding; the additional input parameters which improved download time for Algorithm 3 is also an essential factor for energy usage, as the other result chapters 5 and 6 has concluded.

Question 2 – Performance The final analysis in this chapter concluded with a relatively strong negative correlation between peak energy and download time. This finding means that any multi-link approach increasing the throughput of a single-link alternative have the potential to consume more energy, regardless if it uses more CPU resources or not. When observing the progressive increase in the peak energy values as the download time decreases, the assumption is that a theoretical single-link download with the same download time, like that of the best multi-link approaches, would consume approximately the same amount of energy by having similar energy peaks. Extrapolating the single-link results, and comparing with an actual multi-link result the difference is that the multi-link result has about 115% higher energy peak.

Next, the best single-link and multi-link were compared in regards to energy usage. This analysis revealed that multi-link approaches in scenarios with two high bandwidth networks were able to compete with the energy usage of a comparable single-link approach. However, in the scenarios with a significant difference in network bandwidth between the links, the multi-link approaches were not able to match the energy usage compared to the best single-link approach. The reason for this is that low bandwidth networks still have a relatively high energy-peak, which means the link would have to reach a throughput above a certain level. The calculated minimum throughput was about 30Mbps. Further findings concluded that the Wi-Fi link and the LTE link consume about the same amount of energy in at the upper throughput values measured in this thesis.

Finally, comparing the progressive increase in peak energy, too the gradual increase in CPU usage when download time decreases, it would be safe to assume that the increased energy usage is not due to the CPU load. Both the smartphones used in this thesis, are relatively simple devices containing multi-purpose integrated circuits.

Thereby concluding; modern smartphones are capable of handling the improved performance of multi-link, and the suspected reason for the energy-usage is the communications hardware like LTE modem and Wi-Fi cards. This means that single-link approaches with equal bandwidth would consume approximately the same.

Chapter 8

Analysis and Discussion

In this chapter, a final analysis is presented. This analysis answer questions across the three previous result chapters. Next, we present the remaining questions, limitations, and challenges, in the discussion section.

8.1 Analysis

This section drills further down into the results and analysis presented in the isolated high-speed scenario from the previous three result chapters 5, 6, and 7. The motivation is to tie both of the research questions with one central analysis; If Algorithm 3 gains the highest aggregated throughput and therefore uses the least amount of time to download the payload, and if download time is a critical factor for both CPU and energy usage, what are the total impact on the devices? When comparing CPU and energy usage, a comparable value needs to be used. In section 7.4.5 we analyzed how the energy usage affected the devices by comparing the isolated high-speed results with the energy potential from the synthetic tests. This analysis leads to energy usages in percentages. With these energy percentages, and the CPU median numbers from section 6.1.1, as well as the download times from section 5.1.1, we get;

	Samsung S6	**Samsung S7**
Download time vs CPU	-0.91	-0.92
Download time vs Energy	-0.83	-0.91
CPU vs Energy	0.97	0.95

Table 8.1: Utilization correlation results between download time and CPU, for download time and energy, and for CPU and energy

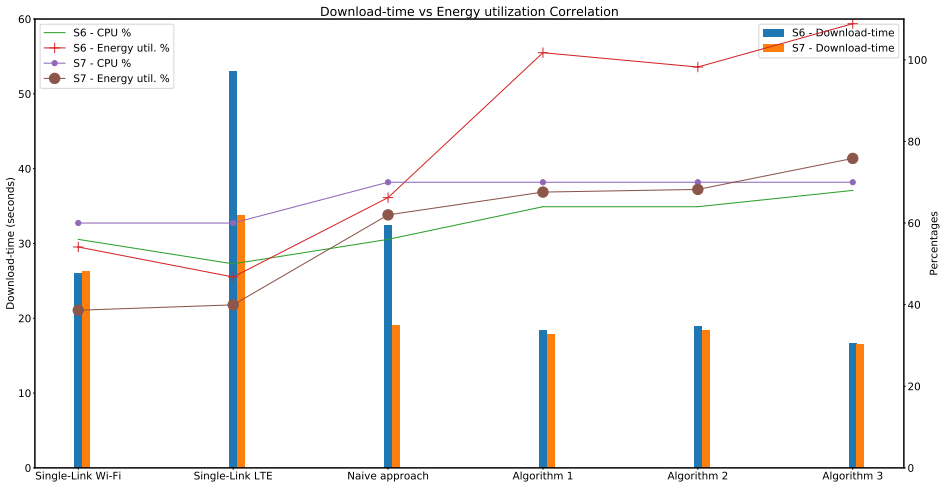


Figure 8.1: CPU and energy utilization for the Samsung S6 and the Samsung S7 device, from the isolated high-speed scenario.

Figure 8.1 shows the utilization of CPU and energy in percentages, combined with the download time representing aggregated throughput. The columns in the figure represent download time connected in the left y-axis, and the plots represent CPU and energy connected to the secondary right y-axis.

Samsung S6 For the Samsung S6 device, the figure shows that the multi-link approaches with low download time profoundly influence energy usage – the red line. For Algorithm 1, 2, and 3, the amount of energy consumed is close to the device potential, reaching well into the high 90th's. Table 8.1 shows a strong negative correlation between energy and download time, -0.83, and confirms that the device is consuming more energy when the download time is decreasing. This could mean the aggregated bandwidth in the isolated high-speed scenario is close to the device capabilities in terms of energy usage.

The strong negative CPU correlation, -0.91, means the device uses more CPU resources increasingly when the download time decreases. However, the figure shows that the increased CPU usage is at a slower rate compared to energy usage. This means the device has more CPU reserves, and also strengthen the assumption that the reason for the energy usage is not CPU time, but energy from the networking hardware.

Samsung S7 For the Samsung S7 device, figure 8.1 shows that energy usage is increasing when the download times are decreasing – the brown line. Table 8.1 shows that the strong negative correlation between both download time and CPU of -0.92, in addition to the strong negative correlation between download time and energy usage, -0.91, the device is increasingly affected when the download time is decreasing – the same as with the Samsung S6 device. However, where the Samsung S6 device utilizes above 90% of the energy potential for Algorithm 1, 2, and 3, the

Samsung S7 has an increasing energy usage of around 70%. This steady increase in energy usage compared to the Samsung S6 is the reason the negative correlation is stronger for the Samsung S7. This also means the Samsung S7 is more capable of higher aggregated bandwidth compared to the Samsung S6.

Similarity Both devices show a strong positive correlation between CPU and Energy usage. Even if the energy usage and CPU resources are different, the effect is similar. This is the main reason the Samsung S6 device were excluded from many of the experiments.

8.2 Discussion

In this section, some reaming questions are discussed, with topics like "Do newer hardware use less energy?", Moreover, "Why do native curl use less CPU and memory?". Next, some limitations and challenges this thesis have encountered are discussed, and finally, we propose some extra features not implemented in any of the proposed algorithms.

8.2.1 Remaining questions

This section discusses some of the questions both directly and indirectly related to the result chapters.

What about newer smartphones? The findings in the result chapters, point in the direction that the Samsung S7 device is more suited for multi-link applications than its predecessor the Samsung S6 device. When the throughput numbers reached the maximum for the scenarios used in this thesis, the Samsung S6 utilized much of the available energy resources.

The newer Samsung S8 device, not tested in this thesis, uses an "Exynos 9 Series (8895)" [13] chipset which Samsung claims improves energy efficiency by 40%. Little information is to be found on these chipsets, only promotional websites, so further details are hard to analyze. However, one takeaway is Samsung's focus on energy efficiency.

In the test-setup section 3.2, many smartphones were excluded due to the compatibility with the software-measurements requirements. Other manufacturers of Android smartphones are probably working on energy usage improvements as well, and their later generations could be an interesting study to look into how capable they are at multi-link applications.

Why do native curl use less CPU and memory? The initial testing of single-link performance for both LTE and Wi-Fi used simple bash scripts. These bash scripts used the precompiled Curl CLI application that is included in LineageOS distributions. All of these tests revealed that the memory usage of Curl CLI outperformed the simple libcurl applications by almost tenfold. Where the libcurl single-link applications used about 20MB of memory, the Curl CLI application used approximately 3-4MB

at the same task. For CPU usage, the differences were less significant but still noticeable. Given the fact that Curl CLI also uses libcurl for its downloading features, one could questions why the differences are this significant.

One of the contributing factors could be compiler optimizations for the Curl CLI application, which was not a focus in this thesis. With this assumption, it could be possible to make similar improvements for the client software running the algorithms, which could further lead to improved CPU and memory usage.

What sampling rate to use for software monitoring? This thesis uses software monitoring for CPU and memory usage, as well as energy usage. This leads to some interesting findings and limitation in regards to which hardware is possible to use. Having a testbed consisting of 100% software monitoring also have the advantage of testing during physical device movement in dynamic scenarios, without having external hardware connected to the device. The disadvantages of software measurements meant that increasing the sampling rate was not possible without affecting the results, in particular, CPU measurements. Therefore, a sampling rate of 1Hz for both energy measurements, as well as CPU and memory measurements were selected. This sampling rate is the same as the default sampling rate for monitoring software like *top* [35] and *htop* [15].

The low sampling rate meant some limitations in the now-energy values for energy measurements. The now-energy values had a higher degree of volatility, compared to CPU and memory fluctuations. It and could, therefore, have benefited by a higher sampling rate. A higher sampling rate for the now-energy values would not have lead to entirely different results, but could possibly make the margins larger or smaller for the integral calculations.

8.2.2 Thesis Limitations

This section describes the limitations found in this thesis surrounding hardware, scenarios, and payloads.

Hardware One of the main goals in this thesis was to measure multi-link performance on newer smartphones with more capable Wi-Fi and LTE networking hardware. This ended up with smartphones that during this period were 2-3 years old. Newer hardware like the Samsung S8 described above, and the Samsung S9 device both supports LineageOS and have compatible max-chipsets. Investigating other manufacturers of Android smartphones could also have lead to some interesting results, as a more varied device pool could have been beneficial to the experiments and the results.

Scenarios and payloads The experiments selected in this thesis reflected challenges and limitations. One of these limitations was the selected file

download workload for the scenarios. The payload, consisting of 250MB file for all measurements, was selected because the time it takes to gain carrier aggregation, and the time it takes for energy peaks to build up. This was important because we wanted to polarize the results, making it possible to analyze correlations between energy peaks and download time. Smaller payloads could, for a typical smartphone user, represent more normal network traffic.

Next, the selected scenarios were out of access and practicality. The isolated scenarios were selected because it was easily accessible and the Wi-Fi network was configurable. The dynamic scenarios did not match the Wi-Fi/LTE bandwidth ratio that of the isolated scenario, and it was challenging to compare the two scenarios without extrapolating the results.

Constant bandwidth values Algorithm 3, proposed in this thesis, was unable to estimate bandwidth for the BDP continuously – section 4.2.3. During the validation stages, experiments of continuously estimating bandwidth resulted in inconsistent and unusable results, particularly for the LTE networks. Software like [48] exists for the sole purpose of estimating bandwidth, but relying on external software was not practical for the test setup. The experiments instead used a constant value for the bandwidth factor of the BDP calculations. This value was set after some initial testing in each scenario trying to find the bandwidth of the network. The scope of this thesis did not permit the extensive work that was necessary for solving this issue dynamically.

8.2.3 Challenges

This thesis experienced some challenges along the way. In this section, the most central of these challenges are described.

Screen on during testing, due to deep sleep The first iteration of the test-bed included a monitoring workstation connected wirelessly to the smartphone during the experiments. This setup was both unstable and introduced an overhead in overall resource usage. The solution for making the test-bed fully autonomous was to hook the tests to a running init-system on the LineageOS system. However, the new test-bed revealed some inconsistent measurements due to CPU starvation. The Android operating system deploys a lot of power-saving features, with one of them turning the device into a deep sleep when the screen is off. There exists APIs for circumventing these mechanisms, but most of these APIs are tailormade for the applications environment running on the Java JRE. We addressed this issue by forcing the screen to be on during experiments, making the energy usage affected by some amount. The additional energy usage should be considered a small constant overhead for all the results.

8.2.4 Extra features

Some extra features were not implemented, this included slow link fail-over, and link banning.

Slow link fail-over Due to the volatility of wireless networks, link connectivity and stability, cannot be guaranteed. The probability of a link losing connectivity or becoming unusable slow is apparent and something to take into consideration. In practice, this means that the algorithm must monitor all requests assigned to a link and, and if the throughput dips below a specific limit, the algorithm would have to release the request form the link and let another available and better-performing link complete it. Another related problem is if more than one link becomes slow, or even worse; all the links become slow – what should happen? In this situation, not much can be done other than try to get slow links to complete their requests.

To deal with the first scenario, where one link becomes slow the following algorithm is proposed:

Algorithm Static – slow link algorithm

```
1: function LINKCHECKSTATIC(link_x)                    ▷ Inputs a link in flight
2:     staticThrougputLimit ← 100                       ▷ Predefined KBps
3:     link_xThroughput ← GetLinkThrouhghputAverage(link_x)
4:     if link_xThrougput < staticThrougputLimit then
5:         request ← ReleaseRequestFormLink(link_x)
6:         RemoveLinkFromLinkPool(link_x)
7:         HandoverRequstToLink(request, link_y)
8:     end if
9: end function
```

The static limit, in this case, 100KBps, does not check if the other links are equally, or almost equally slow. Meaning, if $link_x$ has an average throughput of 90KBps, and $link_y$ has an average of 110KBps, it is not favorable to release the request from $link_x$. Therefore a more dynamic algorithm is proposed:

Algorithm Dynamic – slow link algorithm

```
1:  function LINKCHECKDYNAMIC(link_x)                    ▷ Inputs a link in flight
2:      slowLinkRate ← 0.2                               ▷ Predefined slow link rate
3:      totLinkThroughput ← GetTotalLinkThroughputAverage()
4:      linkCount ← GetLinkCount()
5:      dynamicLimit ← (totLinkThroughput / linkCount) * slowLinkRate
6:      link_xThroughput ← GetLinkThrouhghputAverage(link_x)
7:      if link_xThroughput < dynamicLimit then
8:          request ← ReleaseRequestFormLink(link_x)
9:          RemoveLinkFromLinkPool(link_x)
10:         HandoverRequstToLink(request, link_y)
11:     end if
12: end function
```

The dynamic link check algorithm solves the combined throughput problem. It does this by comparing the total throughput average against the current link average and multiplying this by a rate, but by doing so, it also introduces static factors.

Request continuation The algorithms above do not describe in detail what would happen to a request which is released from a link. There exist two approaches to this; the first would be a naive approach, not to be confused with the *naive approach* in section 4.3.2, where all the data downloaded from the request is thrown away, and the link adopting the request starts all over. The second approach is more complex, where the releasing link stores the bytes downloaded, and the adopting link starts from this point. Meaning, if $link_x$ has downloaded 40% of a request, and the becomes unable to download the rest, $link_y$ could start from the 40% cursor, and only download the rest of the request, or download a new full request size from this cursor. The second approach has, however, to take into consideration not overreaching when creating a new request from the failed links cursor. It also has to be very precise in regards to continue exactly where the other link cut off. If this fails, the whole payload will probably become corrupt.

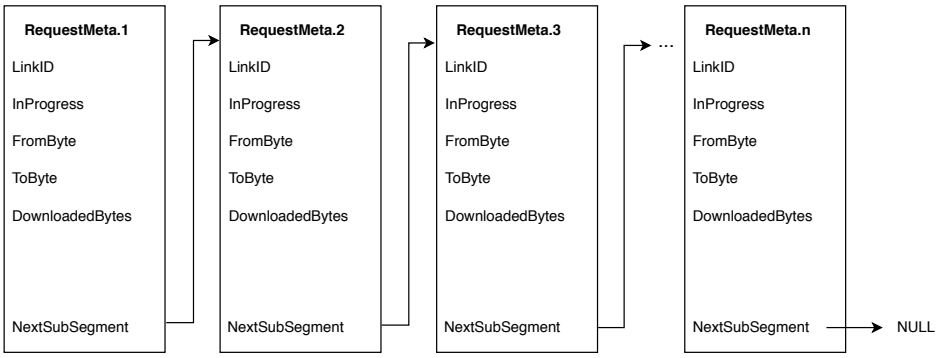


Figure 8.2: Visual representation of support meta-data structure

Continue where the last link left off Supporting the "continue where failed link left off" functionality, a support structure is needed showing in figure 8.2. The support structure needs to keep track of each request, what link is assigned, what byte range, and the progress. With this in place, available links can scan the chain of requests for failed ones, and pick up where the failed link left off.

Algorithm Continue where failed link was canceled

```
1:  function FINDFAILEDREQUESTS
2:      node == root
3:      while node.next ≠ null do
4:          incomplete ← node.tobyte ≠ node.downloadedbytes
5:          if ¬node.inprogress & incomplete then
6:              return node.downloadedbytes − node.startbyte
7:          end if
8:          node ← node.next
9:      end while
10:     return null
11: end function
```

The algorithm above, shows a simple function that finds the first node in a linked list which meets the criteria of not being done and is not in progress.

Link banning According to the proposed algorithms above, when a link reaches a certain threshold, it should no longer be used. The problem with this algorithm is when should the banned link be reactivated. A naive and straightforward approach to this would be to reactivate it at a later time and monitor the results. Another approach could be to monitor low-level hardware API's and make decisions based on these results – for example wireless signal strength.

8.3 Summary

In this chapter, we started by presenting an analysis with a focus on comparing the data from the previous result chapters. This meant drilling further down into the numbers and comparing download times against CPU and energy usage. This analysis showed that the Samsung S6 device was severely affected in terms of energy usage by the high bandwidth numbers in some of the scenarios where the download times reach well below 20 seconds. In comparison, the Samsung S7 device proved to handle these low download times better by only utilizing about 70% of its energy potential. Both devices showed that the most significant influence was energy usage, not CPU usage.

Next, we presented a discussion with some questions surrounding energy efficiency and smartphone evolution. The Samsung S7 device used in this thesis improves upon its predecessor the Samsung S6 in regards to energy usage. Samsung claims that the newer model *S8* improves upon the S7 by 40%. Seeing how the Samsung S6 and S7 differed in energy potential, it would have been interesting comparing even newer Samsung devices like the S8.

In the limitations and challenges sections, some issues with the test setup were discussed. These limitations were the selection of smartphones, the scenarios, and the constant bandwidth values.

Finally, we proposed extra features for the multi-link approaches. None of these extra features would increase performance but could be useful outside the test scenarios.

Chapter 9

Conclusion

9.1 Summary

Increased smartphone usage demands a high-quality internet connection to optimize the user experience, and many of the popular applications and services require a lot of bandwidth. In this thesis, we proposed algorithms for multi-link HTTP in a mobile environment utilizing an increasing degree of input parameters. The motivation for this was to study and analyze what input parameters was necessary for optimum performance in terms of how throughput and resource usage affect modern Android smartphones.

We selected time and resources as metrics for the performance analysis. These metrics included monitoring throughput, CPU and memory usages, in addition to energy usages. After having implemented and verified a working software enabling the proposed algorithms to run directly on an Android smartphone, we designed a test-bed to be used in several scenarios for experimenting. The scenarios consisted of isolated scenarios utilizing a high-speed and low-speed Wi-Fi internet connection. The other dynamic scenarios consisted of two public Wi-Fi's. The reason for these scenarios was to test the approaches and devices in two very different situations. The isolated scenario represented a controlled environment, and the dynamic scenarios were meant to represent scenarios for a typical smartphone user.

We presented the results from the experiments in the three result chapters, each representing the metrics; throughput, CPU and memory, and energy usage. In each of these chapters, the results from all the scenarios were presented and analyzed individually.

First, the throughput chapter concluded with the approaches to multi-link working as expected by utilizing the aggregated bandwidth. Between the Algorithm 1, 2, and 3, the third algorithm utilized the aggregated bandwidth the best on both devices in almost every scenario. Algorithm 3 managed to aggregate the links and share the load at a rate very close to the theoretical maximum of what was expected. It also proved adaptability in dynamic scenarios where the differences between the approaches were more significant, compared to the isolated scenarios. The one scenario where Algorithm 3 did not provide the lowest download times were the

isolated low-speed scenario, but the differences between Algorithm 1, 2, and 3 were negligible. Algorithm 1, and 2 excelled in this scenario due to the static nature of the limited Wi-Fi link.

Next, the CPU and memory chapter also concluded with Algorithm 3 using the least amount of CPU and memory resources over time. In this chapter, we focused on the integral number for the resource metrics. When doing so, we noticed that the memory values were very similar across both the multi-link approaches and single-link approaches, and for all the scenarios. Because of this, we decided to focus mainly on CPU resources. In that, we found a strong correlation between download time and CPU median for all the scenarios, but, even if the correlation was very strong, the amount of increased CPU median was not. Meaning, when the download time decreased, the CPU median did not increase in at the same rate. We concluded that the devices were more than capable of handling higher throughput numbers from a CPU median perspective. In terms of memory consumption, the effect the multi-link approaches had on the devices were negligible compared to the available system memory resources.

Finally, the energy chapter concluded with Algorithm 3 using about the same amount of energy as the best single-link approach when comparing the decrease in download time with the increase in peak energy – but only for some scenarios. When comparing the best single-link and multi-link approaches all the results could be divided into two groups. The first group consisted of multi-link results using slightly more energy compared to the best single-link approach, and in the second group, the multi-link approaches surpassed the energy usage of the best single-link approach. The results showed that the groups were divided by the bandwidth of the Wi-Fi link, and this lead to the discovery of the base-level energy peaks. The energy peaks for links with low bandwidth does not go below a certain level, making links with about 8Mbps and 40Mbps bandwidth gain the same energy peaks. This concluded that multi-link setup using one high and one low-bandwidth link, as in the case of the second group scenarios, could in some cases use more energy than by using a single high-bandwidth link only. Only when two relatively high-bandwidth links were combined, as in the case of the first group, we saw an energy usage approximating that of the single-link. The correlation between download time, in this case, throughput, and energy peaks were concluded to also be strong – as in the case with CPU usage. However, the energy peaks increased more progressively when the download times decreased. In particular when the download times went from about 25 to 16 seconds. Based on this, we extrapolated that if one of the single-link approaches had reached the same download times as Algorithm 3, the energy peaks would have matched that of the multi-link approaches, making the energy-usage the same. A linear extrapolation of theoretical single-link showed that a matching multi-link result had a 115% higher energy peak. The progressive increase in energy peaks could mean the devices were reaching close to maximum capacity, in regards to energy usage.

In the last analysis, this assumption was confirmed. In this analysis, the numbers from all the previous chapters were compared and proved

that the Samsung S6 device was very close to its energy potential when the download times decreased to a minimum, as in the case of Algorithm 3 in some scenarios. This could mean the aggregated bandwidth of some test scenarios surpassed, or was very close, to what the device is capable of. The Samsung S7, on the other hand, seemed to handle the high throughput better.

9.2 Main contributions

In section 1.2 the two following research question were stated;

What input parameters are most important for multi-link HTTP approaches in a mobile environment consisting of smartphones and wireless networks? Three multi-link algorithms using an increasing amount of input parameters were proposed. Algorithm 1 utilized payload-size and throughput as input parameters, Algorithm 2 added bandwidth, and Algorithm 3 further added RTT. By utilizing all of the available input parameters, Algorithm 3 proved to be the most effective in terms of download time affecting resource usage over time. The additional input parameters of Algorithm 3, were used to calculate the optimal lower and upper limit for the individual request sizes. This proved to be important in dynamic scenarios, and in particular for the LTE network, compared to Algorithm 1 and 2. Thereby concluding; in mobile environments with smartphones connected to Wi-Fi networks and LTE networks, the input parameters; payload-size, throughput, bandwidth, and RTT are all important to utilize all of the available aggregated bandwidth.

With the correct input parameters, what multi-link performance gains can we expect in modern smartphones, and how are the device and its limited resources affected? Both devices presented in this thesis managed to aggregate the available bandwidth using the Wi-Fi and LTE link concurrently. This meant a total throughput of approximately 130Mbps at times when both links were fully utilized. The CPU and memory results showed that neither of the devices was performing at the maximum at peak throughput, but a slight increase in CPU median values was observed when the download times started to decrease. The memory usage of multi-link approaches was negligible compared to the available device memory.

However, the energy results showed that the Samsung S6 device consumed almost as much energy as the synthetic stress test when downloading with Algorithm 3 in the high-speed scenario. This seems to indicate that the device was severely affected by the high throughput, and might not be suitable for the speeds the multi-link approaches presented. On the other hand, the Samsung S7 device proved to handle the high throughput numbers better in terms of energy usage. The differences between these devices do not necessarily mean the Samsung S6 device

was incapable of handling multi-link, but instead, the device would have handled scenarios with lower aggregated bandwidth better.

Thereby concluding; modern smartphones can achieve significant performance gains in terms of throughput by utilizing multi-link HTTP. Further, when utilizing most of the available bandwidth, none of the devices presented a problem in terms of CPU and memory usage. However, with the increased throughput, both devices displayed a noticeable increase in energy peaks, but not necessary due to multi-link.

In addition to the research questions above we contributed with;

Propose approaches for multi-link HTTP, using an incremental degree of input parameters. We proposed three algorithms for dynamic request-size calculations. Algorithm 1 relied on a static upper limit for the request-sizes for both links. This had the disadvantage of not being able to adjust for the individual link specifications, like LTE needing larger request-sizes to perform optimally. Algorithm 2 addressed this issue by introducing dynamic upper limits for the requests. This additional calculation proved to perform slightly better than Algorithm 1, but it also increased the Link Δ. Finally, Algorithm 3 addressed the issue of Algorithm 2 by introducing a lower limit for the sizes of the request, making it capable of dynamically calculate both upper and lower limits for the requests sizes. The extra calculations using all the available input parameters proved to outperform Algorithm 1, and 2.

Propose implementations prepared for proxy service functionality, to be used for all HTTP traffic on a smartphone. In chapter 4, a client application supporting Algorithm 1, 2 and 3 were described. This client application was able to run directly on Android-based smartphones, and with some extra work, could stand as a basis for a possible proxy service. The experiments and profiling of this application proved that there were no significant resource usage overhead.

Software based energy monitoring We argued that a software-based measurement technique would be beneficial, and described a Power Management Integrated Circuit (PMIC) chip-set that makes software-based energy monitoring possible – with a high degree of precision. Further investigation showed that not all PMIC chips have the same degree of precision and that the trend was pointing in the direction that newer devices have higher precision chip-sets. The data from these chip-sets were the foundation for the energy result chapter.

Extra features In chapter 8, we proposed extra features for the approaches and implementations. They were theorized out of observations done during the course of this thesis. The first feature describes slow link fail-over, in situations where one of the links loses connectivity like a Wi-Fi network out of range. It also describes algorithms to recover from such situations by request continuation and continue where the

failed link was canceled. Finally, we discuss link banning, and when to reactivate a slow link.

9.3 Future work

This section presents some suggestions for future work.

Proxy server implementations The validation stage of the algorithms and implementations was more challenging than first estimated, forcing the elimination of some goals. One of these goals was to create a fully working proxy server running on the device, capable of receiving any incoming HTTP request from all the applications installed on a smartphone. Android-based smartphones have built-in support for utilizing proxy servers, and enabling this could open up for a further investigation into how the devices perform and how the resources usages are affected when all, or most of the HTTP traffic utilizes multi-link.

Fine tuning RTT measurements The HTTP library used for the implementations in this thesis did not provide continuous monitoring RTT numbers after the initial TCP connection was created. This limited the dynamic calculation of the Bandwidth Delay Product. Continuously monitoring this value could improve the performance of the algorithms in situations where the RTT value fluctuates during a download.

What specifically are consuming energy? The software measurement for energy usage was based on specific PMIC chipset found in the smartphones – section 3.2.1. These chipsets expose the current voltage and amperage-levels, providing a power level which calculates to energy usage over time. However, this only shows the energy usage between the battery and the rest of the device. There was no obvious way of telling what specific hardware components were responsible for the energy usage. Further investigations into this topic could expose ways of doing software measurement on specific hardware components, which could lead to some interesting results. Detailed energy usage information could be passed directly into the algorithms as input parameters making them energy-aware.

Bibliography

[1] About Wi-Fi Assist. https://support.apple.com/en-us/HT205296.

[2] Android – optimize for Doze and App Standby. https://developer.android.com/training/monitoring-device-state/doze-standby.

[3] Android – security. https://source.android.com/security.

[4] Callgrind manual. http://valgrind.org/docs/manual/cl-manual.html.

[5] Celerway http boosting. https://www.simula.no/sites/default/files/celerway_http_boosting.pdf.

[6] Celerway – Resilient networking. https://www.celerway.com/.

[7] Common Open Research Emulator (CORE) | Networks and Communication Systems Branch. https://www.nrl.navy.mil/itd/ncs/products/core.

[8] Connection management in HTTP/1.x. https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x.

[9] curl - History. https://curl.haxx.se/docs/history.html.

[10] Docker - Build, Ship, and Run Any App, Anywhere. https://www.docker.com/.

[11] Exynos 7 Octa 7420 Processor: Specs, Features | Samsung Exynos. //www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-7-octa-7420/.

[12] Exynos 8 Octa 8890 Processor: Specs, Features | Samsung Exynos. //www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-8-octa-8890/.

[13] Exynos 9 Series 8895 Processor: Specs, Features | Samsung Exynos. //www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-8895/.

[14] H2o - the optimized HTTP/2 server. https://h2o.examp1e.net/.

[15] htop - an interactive process viewer for Unix. https://hisham.hm/htop/.

[16] HTTP/1.1: Header Field Definitions. https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35.

[17] HTTP/1.1: Status Code Definitions. https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[18] I2c - What's That? /http://www.i2c-bus.org/i2c-bus/.

[19] IEEE 802.3ad DRAFT - SUPPLEMENT TO IEEE 802.3: LINK AGGREGATION | Engineering360. https://standards.globalspec.com/std/1958412/ieee-802-3ad-draft.

[20] Introduction to HTTP/2 | Web Fundamentals. https://developers.google.com/web/fundamentals/performance/http2/.

[21] libcurl - available alternatives. https://curl.haxx.se/libcurl/competitors.html.

[22] libev. software.schmorp.de/pkg/libev.html.

[23] libev - benchmarking libevent against libev. http://libev.schmorp.de/bench.html.

[24] Making HTTP Pipelining Usable on the Open Web. https://tools.ietf.org/id/draft-nottingham-http-pipeline-00.html.

[25] Matplotlib: Python plotting — Matplotlib 3.0.3 documentation. https://matplotlib.org/.

[26] Maxim Integrated - Analog, linear, & mixed-signal devices. https://www.maximintegrated.com/en.html.

[27] networking:netem [Linux Foundation Wiki]. https://wiki.linuxfoundation.org/networking/netem.

[28] Nghttp2: HTTP/2 C Library - nghttp2.org. https://nghttp2.org/.

[29] Openssl. https://www.openssl.org/.

[30] PowerAPI. http://powerapi.org/.

[31] Python Data Analysis Library — pandas: Python Data Analysis Library. https://pandas.pydata.org/.

[32] SciPy.org — SciPy.org. https://www.scipy.org/.

[33] SPDY: An experimental protocol for a faster web - The Chromium Projects. https://www.chromium.org/spdy/spdy-whitepaper.

[34] Technopolis. https://www.technopolis.no/en/.

[35] top(1): tasks - Linux man page. https://linux.die.net/man/1/top.

[36] Valgrind Home. http://valgrind.org/.

[37] What is the Download Booster and how do I enable it on my Samsung Galaxy Alpha? | Samsung Support UK. https://www.samsung.com/uk/support/mobile-devices/what-is-the-download-booster-and-how-do-i-enable-it-on-my-samsung-galaxy-alpha/.

[38] zlib Home Site. http://zlib.net/.

[39] Head-of-line blocking. https://en.wikipedia.org/w/index.php?title=Head-of-line_blocking&oldid=835665507, Apr. 2018. Page Version ID: 835665507.

[40] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (2009), ACM, pp. 280–293.

[41] BAN, D., IN, J., AND WOO, H. The resource optimization of heterogeneous network interfaces in wireless mobile devices. In *2014 IEEE International Conference on Communications (ICC)* (June 2014), pp. 1235–1241.

[42] BELLALTA, B. IEEE 802.11ax: High-efficiency WLANS. *IEEE Wireless Communications 23*, 1 (Feb. 2016), 38–46.

[43] BOURDON, A., NOUREDDINE, A., ROUVOY, R., AND SEINTURIER, L. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News 2013*, 92 (2013).

[44] BUI, D. H., LEE, K., OH, S., SHIN, I., SHIN, H., WOO, H., AND BAN, D. GreenBag: Energy-Efficient Bandwidth Aggregation for Real-Time Streaming in Heterogeneous Mobile Wireless Networks. In *2013 IEEE 34th Real-Time Systems Symposium* (Dec. 2013), pp. 57–67.

[45] CARROLL, A., HEISER, G., ET AL. An analysis of power consumption in a smartphone. In *USENIX annual technical conference* (2010), vol. 14, Boston, MA, pp. 21–21.

[46] COLMANT, M., FELBER, P., ROUVOY, R., AND SEINTURIER, L. Wattskit: Software-defined power monitoring of distributed systems. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2017), IEEE Press, pp. 514–523.

[47] CORADETTI, T., SKLOWER, K., CARR, D., MCGREGOR, G., AND LLOYD, B. The PPP Multilink Protocol (MP). https://tools.ietf.org/html/rfc1990.

[48] EVENSEN, K. A bandwidth estimation tool designed for mobile broadband networks: kristrev/bandwidth-estimator. https://github.com/kristrev/bandwidth-estimator, June 2018. original-date: 2012-12-22T22:46:51Z.

[49] EVENSEN, K., KASPAR, D., GRIWODZ, C., HALVORSEN, P. A., HANSEN, A., AND ENGELSTAD, P. Improving the Performance of Quality-adaptive Video Streaming over Multiple Heterogeneous Access Networks. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (New York, NY, USA, 2011), MMSys '11, ACM, pp. 57–68.

[50] EVENSEN, K. R. Aggregating the Bandwidth of Multiple Network Interfaces to Increase the Performance of Networked Applications.

[51] EVGENY KHOROV, ANDREY LYAKHOV, A. K. A. G. A survey on IEEE 802.11ah: An enabling networking technology for smart cities - ScienceDirect.

[52] GO, Y., KWON, O. C., AND SONG, H. An Energy-Efficient HTTP Adaptive Video Streaming With Networking Cost Constraint Over Heterogeneous Wireless Networks. *IEEE Transactions on Multimedia 17*, 9 (Sept. 2015), 1646–1657.

[53] GONG, M. X., HART, B., AND MAO, S. Advanced Wireless LAN Technologies: IEEE 802.11ac and Beyond. *GetMobile: Mobile Comp. and Comm. 18*, 4 (Jan. 2015), 48–52.

[54] HANDLEY, M., RAICIU, C., FORD, A., IYENGAR, J., AND BARRE, S. Architectural Guidelines for Multipath TCP Development. https://tools.ietf.org/html/rfc6182.

[55] HANSEN, T. GSM. http://snl.no/GSM, Oct. 2017.

[56] HOPPS, C. E., AND THALER, D. Multipath Issues in Unicast and Multicast Next-Hop Selection. https://tools.ietf.org/html/rfc2991.

[57] JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[58] JOHANSEN, D., JOHANSEN, H. A., AARFLOT, T., HURLEY, J., KVALNES, \., GURRIN, C., ZAV, S., OLSTAD, B., AABERG, E., ENDESTAD, T., RIISER, H., GRIWIDZ, C., AND HALVORSEN, P. A. DAVVI: A Prototype for the Next Generation Multimedia Entertainment Platform. In *Proceedings of the 17th ACM International Conference on Multimedia* (New York, NY, USA, 2009), MM '09, ACM, pp. 989–990.

[59] JOHNSEN, R., ANDERSEN, P. B., AND STETTE, G. GPRS. http://snl.no/GPRS, Jan. 2018.

[60] LAFON, Y., FIELDING, R., AND RESCHKE, J. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. https://tools.ietf.org/html/rfc7233#section-3.1.

[61] LEACH, P. J., BERNERS-LEE, T., MOGUL, J. C., MASINTER, L., FIELDING, R. T., AND GETTYS, J. Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616.

[62] LI, M., LUKYANENKO, A., OU, Z., YLÄ-JÄÄSKI, A., TARKOMA, S., COUDRON, M., AND SECCI, S. Multipath Transmission for the Internet: A Survey. *IEEE Communications Surveys Tutorials 18*, 4 (2016), 2887–2925.

[63] LINEAGEOS. LineageOS – LineageOS Android Distribution. https://www.lineageos.org/.

[64] LUDIN, S. *Learning HTTP/2 : a practical guide for beginners*. O'Reilly, Beijing Boston, 2017.

[65] MOON, S., YOO, J., AND KIM, S. Optimal Multi-Interface Selection for Mobile Video Streaming in Efficient Battery Consumption and Data Usage, 2016.

[66] SERRANO, P., GARCIA-SAAVEDRA, A., BIANCHI, G., BANCHS, A., AND AZCORRA, A. Per-frame Energy Consumption in 802.11 Devices and Its Implication on Modeling and Design. *IEEE/ACM Trans. Netw. 23*, 4 (Aug. 2015), 1243–1256.

[67] STETTE, G. UMTS. http://snl.no/UMTS, Jan. 2018.

[68] TAKIGUCHI, T., HIDAKA, A., MASUI, H., SUGIZAKI, Y., MIZUNO, O., AND ASATANI, K. A new application-level link aggregation and its implementation on Android terminals. *Wireless Communications and Mobile Computing 12*, 18 (Dec. 2012), 1664–1671.

[69] TANG, Z., WANG, Z., LI, P., GUO, S., LIAO, X., AND JIN, H. An Application Layer Protocol for Energy-Efficient Bandwidth Aggregation with Guaranteed Quality-of-Experience. *IEEE Transactions on Parallel and Distributed Systems 26*, 6 (June 2015), 1538–1546.

[70] TAYLOR, T., SCHWARZBAUER, H. J., KALLA, M., ZHANG, L., MORNEAULT, K., RYTINA, I., STEWART, R. R., SHARP, C., AND XIE, Q. Stream Control Transmission Protocol. https://tools.ietf.org/html/rfc2960.

[71] WANNSTROM, J. Carrier aggregation explained. http://www.3gpp.org/technologies/keywords-acronyms/101-carrier-aggregation-explained, 2013.

[72] WANNSTROM, J. Lte-advanced. http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced, 2013.

[73] YANG, Z. Powertutor-a power monitor for android-based mobile platforms. *EECS, University of Michigan, retrieved September 2* (2012), 19.

[74] YOON, C., KIM, D., JUNG, W., KANG, C., AND CHA, H. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conference* (2012), vol. 12, pp. 1–14.

[75] YUAN, G., ZHANG, X., WANG, W., AND YANG, Y. Carrier aggregation for lte-advanced mobile communication systems. *IEEE Communications Magazine 48*, 2 (2010).