FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Smart Prefetching for Mobile Users under Volatile Network Conditions

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Michael Borkowski, BSc
Matrikelnummer 0925853

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Prof. Dr. Schahram Dustdar
Mitwirkung: Dr.-Ing. Dipl.-Oec. Stefan Schulte, BSc

Wien, 1. August 2015

Michael Borkowski	Schahram Dustdar

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Smart Prefetching for Mobile Users under Volatile Network Conditions

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Michael Borkowski, BSc
Registration Number 0925853

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Schahram Dustdar
Assistance: Dr.-Ing. Dipl.-Oec. Stefan Schulte, BSc

Vienna, 1st August, 2015

Michael Borkowski | Schahram Dustdar

Erklärung zur Verfassung der Arbeit

Michael Borkowski, BSc
Auhofstraße 158/5
1130 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. August 2015

Michael Borkowski

Acknowledgements

I most thankfully acknowledge Dr.-Ing. Stefan Schulte from the Distributed Systems Group of the Vienna University of Technology for advising and mentoring me in writing my thesis; while providing a necessary framework of requirements, Dr. Schulte made sure to give me freedom to find and pursue my path independently. For this trust, I am truly grateful. Also, thanks to Dipl.-Ing. Philipp Hoenisch, who supervised me at my previous project, I had the chance to immerse my acquaintanceship with Dr. Schulte. Furthermore, I was happy to receive a refreshment of knowledge in statistics, which I have gotten terribly out of practice with, by my fellow student Bianca.

Personally, I am deeply thankful to my parents, Irena and Tomasz, who not only laid the foundations of my interest in computer science, but also provided me with an incredible amount of resources, both material and intellectual, and allowed me to strive after what I found was both a passion and a profession for me. My parents as well as my sister Barbara stood by my side throughout my studies; where possible, they supported my ideas and dreams, and in times of necessity, guided my way, providing me with a skill set which paved the way for my work in this field.

The endeavour of study and research is one which along its way requires focus, discipline and persistence. I could not have found energy and endurance without a superb group of friends. I dedicate this work to Stefanie (who never failed to point out that this thesis is, in its core, about navigation systems), a friend whose willpower I have never seen matched, who is not only a marvellous discussion partner in a variety of matters, a promising scientist and an excelling physicist in the near future, but also a delightful companion for endless debates over coffee. Likewise, I was happily able to draw from the friendship and support of Christian, who I would not want to do without; his company on evenings after busy days allowed me to take a break from work where necessary; Elisabeth, a charming friend, who stood by my side at all times, never failing to supply me with witty gags and legal advice; Konrad, who was always in for a chess match and a deep intellectual discourse. I received a lot of professional and personal input from Manuel, with whom each and every discussion was enjoyable and thought-provoking, and whose views I could cut off a slice or two. Marco and Pablo, I am thankful to have such true friends who I can share both technical discussions and memorable adventures with. My group of peers is, amongst others – who have helped me become the person I am today – completed by trusted friends such as Manuela, Stefan, Lisa, Kathrin, Sophie, Krisztina and Daniela, who I am truly happy to share a deep friendship with.

I have to thank T-Mobile Austria GmbH, A1 Telekom Austria AG and Hutchison Drei Austria GmbH for the courtesy of providing me with the right to use their network coverage maps in my work. I furthermore acknowledge the work of thousands of contributors in the open source community – which I am proud to be a part of, and hope to contribute more to in the future –, be it in software facilitating the development of my experiments, support forums or other projects.

Abstract

In the field of mobile distributed computing, the term *prefetching* is used to describe the practice of retrieving data before its actual usage. Doing so results in several beneficial effects for the user, ranging from reduced perceived delay, the possibility to handling lower connection speeds to masking network outages completely.

This thesis proposes a prefetch scheduling algorithm. The described algorithm is capable of scheduling requests so that, in a best-case scenario, the results are fetched exactly on-time and no delay at all is experienced by the user (or client code).

After discussing some formal concepts necessary as a foundation for the proposed algorithm, a basic idea on how to approach a scenario of volatile network connectivity with a given context (predicted future network quality) is presented. The idea is first stated as a problem of mathematical analysis, and subsequently transformed into a concrete algorithm, of which a reference implementation stub in Java is shown.

The second main section of this thesis is a simulation environment capable of reproducibly evaluating the prefetching scenarios by running simulations of mobile units with fluctuating network quality. This thesis presents the architecture of this simulation environment in detail. Following this description, a series of simulation configurations is shown and the resulting simulation outcomes are presented. The results show that the algorithm indeed increases the overall quality of user experience significantly, in the typical use case modelled after a real-world situation by up to 70 %, while maintaining context constraints like limited energy usage.

Concluding, the algorithm is put into context of real-world applications and its feasibility is discussed, alongside with possible future fields of research and concepts of enhancement.

Contents

Abstract **ix**

Contents **xi**

List of Figures **xiii**

1 Introduction **1**
1.1 Mobile Computing 1
1.2 Connectivity of Varying Quality 2
1.3 Prefetching 4
1.4 Thesis Structure 5

2 Related Work **7**
2.1 Prefetching Strategies by Hummer et al. 7
2.2 A Prefetch Model by Tuah et al. 8
2.3 Semantic Caching and LDD by Ren et al. 8
2.4 Caching Strategies by Tabassum et al. 9
2.5 GreedyDual Least Utility by Shen et al. 9
2.6 Value-Based Adaptive Prefetch by Yin et al. 10
2.7 Caching Strategies by Barbará et al. 10
2.8 Cache Management with Invalidation Reports by Cao 10
2.9 Fuzzy Adaptive Buffering by Bagchi 11
2.10 Prefetching Heuristics by Gitzenis et al. 12
2.11 Piggybacking Prefetch Data by Schreiber et al. 12
2.12 Dual Caching System by Han et al. 13
2.13 Latency Reduction Strategies by Jayasudha et al. 13
2.14 Advanced Loading Strategies by Zagarese et al. 13

3 Background **15**
3.1 Term Definitions 15
3.2 Taxonomy of Data Items 16
3.3 Quantities and Units 23

4 Proposed Algorithm **27**
4.1 Discussion of Existing Approaches 27
4.2 Model 30
4.3 Target Variables 31
4.4 Requirements 33
4.5 Scheduling Algorithm 34
4.6 Reference Implementation 46

5 Simulation Environment **51**
5.1 Definition of Requirements 52
5.2 Selection of Tooling 53
5.3 Traffic Shaping Using spiceJ 54
5.4 Repeatable Simulations Using scovilleJ 57
5.5 Simulation of Prefetching Scenarios 64
5.6 Profiling Metrics 70

6 Testing and Analysis **73**
6.1 Pre-Selection of Strategies 73
6.2 Exemplary Simulation 74
6.3 Main Hypothesis 81
6.4 Discussion of Independent Variables 81
6.5 Regression Analysis Experiments 84
6.6 Result Analysis 91

7 Conclusion **93**
7.1 Findings 93
7.2 Outlook, Future Work 94

A Analysis Experiment Reproduction **99**
A.1 Environment Preparation 100
A.2 Exemplary Simulation 101
A.3 Regression Simulations 102

B Regression Analysis Results **105**
B.1 Relative Jitter (S1) 106
B.2 Prediction Amplitude Error Deviation σ (S2) 107
B.3 Prediction Time Error Deviation σ (S3) 108
B.4 Look-Ahead Time t_{horizon} (S4) 109
B.5 Error Correction Factor α (S5) 110
B.6 Prediction Amplitude Error Mean μ (S7) 111
B.7 Prediction Time Error Mean μ (S8) 113

C Fixity Assurance **115**

Bibliography **117**

List of Figures

1.1 Depiction of a mobile user moving through a network 4

4.1 Target variables as optimisation formulae 33
4.2 Original Situation: violated deadlines 35
4.3 After one iteration: semi-resolved situation 36
4.4 Issues resolved for all requests 36
4.5 The notion of $\int B_{link}$ to represent data transmission 39
4.6 Data transmission in discrete quantities ($\sum B_{link}$) 41
4.7 Limiting the effective bandwidth by the producer's bandwidth 43
4.8 Limiting the available bandwidth by an error margin β 44

5.1 Conceptual view of a prefetching scenario 51
5.2 Conceptual view of a scovilleJ simulation 59
5.3 Sequence of ticks and phases 60
5.4 Chronology of ticks and phases 61
5.5 Conceptual UML view of scovilleJ's core interfaces 63
5.6 Simplified view of the simulation setup and execution process 65
5.7 Slots with designated byte rates 67
5.8 Slots with introduced byte rate jitter 67
5.9 Types of error introduced to byte rate prediction 68

6.1 A graphical representation of the exemplary simulation's genesis 76
6.2 Exemplary simulation with strategy A (no prefetching) 77
6.3 Exemplary simulation with strategy B (simple prefetching) 79
6.4 Exemplary simulation with strategy C (advanced prefetching) 80
6.5 Response time over jitter 85
6.6 Data age over jitter 85
6.7 Response time over prediction amplitude error σ 86
6.8 Data age over prediction amplitude error σ 86
6.9 Response time over prediction amplitude error μ 87
6.10 Data age over prediction amplitude error μ 87
6.11 Response time over prediction time error σ 88
6.12 Data age over prediction time error σ 88
6.13 Response time over prediction time error μ 89

6.14 Data age over prediction time error μ 89
6.15 Response time over t_{horizon} 90
6.16 Data age over t_{horizon} 90
6.17 Response time over α 91
6.18 Data age over α 91

7.1 Coverage maps of Austrian mobile networks 96

CHAPTER 1

Introduction

When regarding software systems with spatially distributed components, it is often the case that part of the software is located on a mobile device, while another part is deployed on a server, running in a data processing centre. Often, the part interacting with the user (the user interface, also called front end) running on the mobile device is frequently being moved between different locations, as the user moves around, for example while driving in a car. The front end is constantly communicating with a server system, also called the back end. This kind of software systems is already very present in our lives when regarding mobile phones; modern *smart phones* have penetrated our everyday lives in a highly perceivable way; the amount of data accessible at the palm of our hands would certainly have impressed, if not startled someone who lived a century ago.

1.1 Mobile Computing

The number of mobile subscribers globally has grown from just 580 million in 2002 to 3.5 billion in 2007, as described by [Chu+11]. The specific use of smart phones alone has reached 32.2 million units in the second quarter of 2008, according to [Pal+10], which demonstrates the massive number of used mobile devices capable of providing more than the basic functionality of a mobile phone (voice calls and text messages). [Ver06] presents that mobile device usage has evolved in a vastly different way from usage of stationary devices. It is observable and discussed in [Bao+11] that mobile devices are increasingly replacing traditional, non-mobile devices for certain tasks. Mail, communication in general, or calendar management – these use cases only represent the *classic* business-related tasks that smart phones are capable of performing. In addition to the aforementioned activities, smart phones hold potential for entirely new use cases.

One notable example of a use case provided by smart phones is the kind of real-time, ubiquitous, ever-reachable instant messaging we experience using mobile applications

such as WeChat[1], WhatsApp[2], Facebook Messenger[3], Line[4] or Viber[5]. While the possibility of asynchronous text-based messaging was already provided using text messages (Smart Message Service, or SMS, which in some regions is used synonymously to the text messages themselves), it was usually rather expensive to use and the feature set was limited mostly to transmission of short text messages[6]; modern instant messengers, however, allow for a media-rich experience, where the transmission of photos, videos, audio, location or other media can be achieved with the same ease as sending a text message, which has led to a rapid increase in popularity of instant messengers (see [Par+11], [Law+06] or [KL05]).

A transition from simple (*dumb*) phones to lightweight high-performance computation devices such as smart phones observably opens up completely new use cases. What this implies is that new technology should be regarded in a broader context, possibly not only limited to the originally intended purpose, and that research in the field of mobile services is not only auspicious, it is obligatory. While examining the name *smart phones*, one can already observer the adjective *smart* in connection with an increasing number of everyday objects; not only do we possess *smart phones*; *smart cars* promise to autonomously drive us to our *smart homes*, which are all part of a *smart city*. The tendency to interconnect and to integrate virtually all our surroundings can clearly be observed considering the movement towards the what seems to be the great goal of current development, the *Internet of Things (IoT)*.

One particular instance of *IoT* is the development of increasingly integrated computers in cars. While a car itself, a device useful for changing one's location without employing a significant amount of manual labour (at least in proportion to the distance travelled), has a rather simple main use case, it is natural for us nowadays that a car excels this purpose by a rich facet of differing capabilities, the most used one maybe being the car radio (or, in recent years, CD or Line-In playback or perhaps Bluetooth capabilities). It is intuitive and seems quite likely that cars in several years or decades will work in ways that are not thought of at all today, or at least still under very juvenile development.

1.2 Connectivity of Varying Quality

This thesis stems from the development of a *smart car* system, where an on-board computer is constantly connected to a central unit (server), for example via a 3G or 4G mobile connection, and is integrated into a user's life for his convenience. The personal calendar could in such a scenario be synchronised with the car's heating system to allow for a comfortable temperature to develop shortly before the user needs to drive; other

[1] WeChat is a service developed by Tencent Holdings Ltd. in China; `http://www.wechat.com/`

[2] WhatsApp Inc., based in Mountain View, is a a startup founded 2009 by Jan Kourn and Brian Acton; `https://www.whatsapp.com/`

[3] Facebook, Inc. was founded 2004 by Mark Zuckerberg, Eduardo Saverin, Andrew McCollum, Dustin Moskovitz and Chris Hughes

[4] Line Corp. was founded 2000 in Japan

[5] Viber is a service developed by Rakuten, Inc. in Japan

[6] Vendor-specific extensions allow for transmission of other media as well.

IoT devices such as a smart fridge could notify the user of chores like stocking up on milk en-route; these are merely some examples provided in order to give the reader a broad understanding of the possibilities which are currently subject of imagination and vision.

A smart car on-board computer with its constant connection to a central server, however, is, as an inherently mobile device, exposed to certain limitations which accompany mobility, the main one being a network connection of fluctuating quality. This situation is schematically shown in figure 1.1. Such a network connection might have periods of time where metrics such as bandwidth or latency showing satisfactory quality, however, at other times, the connection might have sub-par attributes like low speed, high latency or frequent disruptions. While such disruptions are sometimes unpredictable, certain assumptions can be made nevertheless, for example, if the user's route is leading through a tunnel – these structures are known to have rather poor mobile network reception, leaving aside some cases where special measures have been taken to install equipment providing reception inside the tunnel –, it is foreseeable that network quality during that part of the journey will most likely be close to zero.

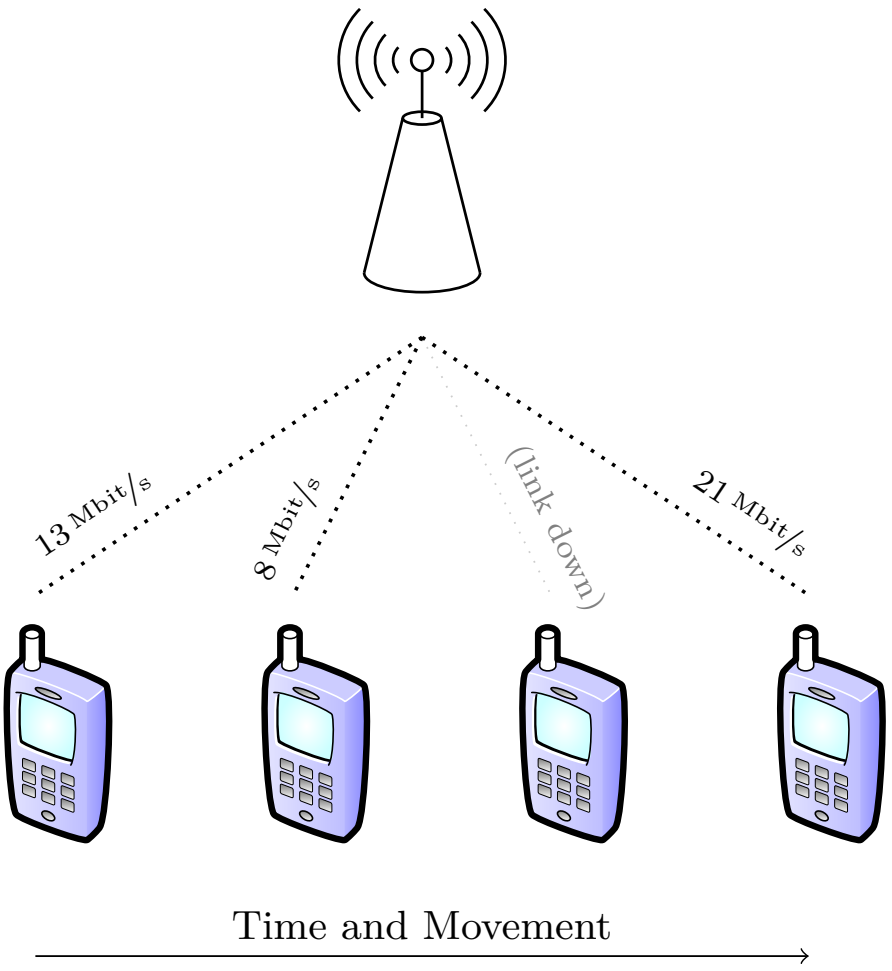


Figure 1.1: Depiction of a mobile user moving through a network with zones of varying quality

1.3 Prefetching

The described scenario of foreseeable network quality drops during some parts of, for example, a journey, immediately yields a possible solution for the delay the user is going to experience: data required in times of insufficient network connectivity can be transferred during periods where the connection quality is ample. This process of transferring data ahead of its requirement, in this case in order to have it available upon request despite degraded network quality, is called *prefetching*.

Our goal is to research, provided that certain limitations of the network connections are known or can be predicted with fair probability, mechanisms enabling the user to experience constant service. We therefore assume some degree of knowledge about future network connection quality parameters and attempt to derive a mechanism – an algorithm – for using that knowledge in mobile software systems.

1.4 Thesis Structure

Following these preliminary words in chapter 1, we present our research on existing literature in the field of mobile computing in chapter 2, discussing briefly some ideas and concepts (closely or remotely) related to the problem of prefetching.

On our journey of finding an algorithm, we first lay out some formal, theoretical groundwork about data transmission, upon which we construct an idea how to schedule data transfers in a manner aiming for proper reaction to changing network quality. Our findings and assertions are presented in chapter 3, which gives an overview of background information, including term definitions and a taxonomy of data unit types.

We elaborate on the necessary assumptions, restrictions, input and output data and target variables of a prefetching algorithm in chapter 4, after which we show how the formal background from the foregoing chapter can be used to derive an idea for a concrete prefetching algorithm. We present such an algorithm and discuss its development as well as its mode of operation.

The subsequent task we face is the one of analysing the performance of our algorithm in the described situation. In order to rule on the algorithm's efficiency, we develop an environment for simulating situations resembling the real-world problems which initially motivated our research, namely the scenario of a mobile computing unit fetching data under varying network conditions. Chapter 5 provides information and discussion about said simulation environment.

Finally, having implemented both algorithm and environment, we analyse and compare results for various scenarios in chapter 6. We do so not only by changing parameters of the simulation, but also by substituting our algorithm with surrogates, to compare its performance to scenarios lacking our algorithm. We discuss how the results reflect the improvement in certain key metrics provided by our algorithm.

Chapter 7 then concludes, reflects on lessons learnt and tries puts the results into relation to real-world scenarios; we also identify possible future research paths in the field of prefetching and mobile computing.

CHAPTER 2

Related Work

Very different approaches to the problem of prefetching are present in current scientific work. There are several aspects of the problem that are interesting and required research; some of those aspects have been addressed more frequently than others. Most of the related work covers aspects close to our problem. We have adopted some ideas and concepts from the work, as stated in detail in section 4.1.

This chapter aims at providing a broad overview of existing literature in mobile and distributed computing tackling in some way the problems of data caching, prefetching or sparse mobile resources (energy, bandwidth, connectivity) and the maintenance of a consistent user experience.

2.1 Prefetching Strategies by Hummer et al.

In "Context-Aware Data Prefetching in Mobile Service Environments" [Hum+14], Hummer et al. propose a decision problem for optimising data prefetching and continuous *Quality of Experience* and discuss different mechanisms for generating service requests for prefetching. The presented mechanisms assume expected network quality to be known, and only server-client communication is investigated (as opposed to end-to-end communication).

The paper describes the relation of prefetching and caching; it shows that those two concepts are often combined to achieve continuous Quality of Experience by storing prefetched data in a transient cache. Also a part of discussion is the aspect that the prefetching can not only be used to mask service or link interruptions, but also to reduce user-perceived delay in general.

Two general types of prefetching are discussed: *Periodic Prefetching* and *Context-Aware Prefetching*. Context-Aware Prefetching is analysed thoroughly and a classification

of three request types is shown: *Constant Requests (Polling)*, *Template-Based Requests* and *Complex Request Patterns*.

A reference implementation is provided as well as a test setup for analysing its performance. The results of the presented experiment are analysed and show that context-based prefetching provides the best performance in terms of result age. Context-based prefetching is also shown to outperform periodic prefetching regarding unused prefetches.

One of the parameters of the proposed algorithms is t_P, the look-ahead time; in "Context-Aware Data Prefetching in Mobile Service Environments", this parameter is fixed for a given simulation run. See section 4.5.3 for a discussion of this parameter.

2.2 A Prefetch Model by Tuah et al.

In "Investigation of a Prefetch Model for Low Bandwidth Networks" [Tua+98], Tuah et al. investigate speculative prefetching, assuming some knowledge about future accesses to data sources and investigates the performance of a prefetcher that utilises this knowledge. Under the assumption that prefetching is neither preempted nor aborted, the paper derives a theoretical limit of improvement in access time due to prefetching and shows an algorithm for prefetching one access ahead under the assumption that retrieval time is uniform. For non-uniform retrieval time, the paper shows the two prefetch strategies mainline (which fetches a single path of documents the user is predicted to select) and branch (which selects multiple possible paths of advancements and fetches them based on their probability) and discusses their behaviour.

2.3 Semantic Caching and LDD by Ren et al.

Given the context of *Location Dependent Data* (LDD), Ren and Dunham describe in "Using Semantic Caching to Manage Location Dependent Data in Mobile Computing" [RD00] semantic caching by showing its aspects of putting semantic attributes onto previously non-semantic elements such as the user's location. This allows for an efficient cache management by taking into account the user's position and speed and, based on this data, optimising queries in many ways.

One use case for semantic caching is the situation where a user's query is locally overlapping with a previously-issued query, in which case the results for the overlapping region can be read from the cache, allowing not only for reducing overall network traffic, but also supplying the user with a preliminary, incomplete result until the rest of the result is fetched.

Another advantage of semantic caching is the decision on a cache replacement candidate. The paper describes strategies for managing the results of location dependent queries in a semantic cache. Basic methods of discarding cache entries are presented, including Furthest Away Replacement (FAR), Least Recently Used (LRU) and Most Recently

Used (MRU). FAR is the method proposed by the paper, which is also shown to outperform MRU and LRU in experiments, exposing its strength by utilising the user's position for cache replacement decisions.

2.4 Caching Strategies by Tabassum et al.

In "A Location Dependent Semantic Cache Replacement Strategy in Mobile Environment" [Tab+12], Tabassum et al. target *Location Dependent Data* (LDD), resulting from *Location Dependent Queries* (LDQ). The authors improve upon known mechanisms such as Furthest Away Replacement (FAR) by extending the model by the dimension of *Segment Frequency* (S_f). In the resulting mechanism, called RBF-FAR, S_f denotes the number of times the semantic segment was accessed.

The paper concludes with experimental evidence that adding S_f into consideration when deciding on cache eviction candidates does increase cache hit rates.

2.5 GreedyDual Least Utility by Shen et al.

In "Energy-efficient Data Caching and Prefetching for Mobile Devices Based on Utility" [She+05], Shen et al. propose an energy and bandwidth efficient data caching mechanism, called GreedyDual Least Utility (GD-LU). A *utility function* is used to assess the necessity of each data item. The utility function assigns to each data item in the cache a value which determines how much value this data item has with respect to the necessary energy cost involved in fetching the item as well as the probability of the item being needed.

The mechanism uses this utility function for two major decisions. Firstly, cache replacement, i.e. discarding existing cache items in favour of a new data item, requires some sort of prioritisation amongst the data items. In GD-LU, the data items are sorted by their utility, and the least utile data items are being discarded.

Furthermore, GD-LU describes a *passive prefetching* methodology, where a variant of utility, *relative utility*, is used to decide whether to keep in cache data items that have been received via a broadcast channel without having been explicitly requested (thus *passive*). This relative utility, however, can easily be used for active prefetching as well, where clients using unicast channels can actively and speculatively prefetch data items which are likely to be used by client code in the future.

"Energy-efficient Data Caching and Prefetching for Mobile Devices Based on Utility" also describes important aspects about prefetching and caching in general, such as the relation between the two, general aspects of cache invalidation and cache replacement, the difference in uplink communication cost versus downlink communication cost and the related gradient of efficiency versus energy consumption.

2.6 Value-Based Adaptive Prefetch by Yin et al.

Similar to Shen et al. in "Energy-efficient Data Caching and Prefetching for Mobile Devices Based on Utility" [She+05], Yin et al. describe in "Power-aware prefetch in mobile environments" [Yin+02] a function assigning a *value* to each data item. This value is derived from parameters such as the item access probability, the size, the retrieval delay and so on, making it a very customisable algorithm. The scheme, called Value-Based Adaptive Prefetch (VAP) uses this value to actively prefetch data items. VAP uses information about the current energy status of the device to determine how many of the data items with the highest assigned value are prefetched.

2.7 Caching Strategies by Barbará et al.

In "Sleepers and workaholics: Caching strategies in mobile environments (Extended version)" [BI95], Barbará et al. provide a classification of mobile clients in a broadcast scenario, where the clients are regarded either as *sleepers* or as *workaholics* – which means that they are classified depending on their ratio of online time to offline time. If this ratio is above a certain threshold, meaning that the client is online most of the time, that client is treated differently with respect to cache invalidation strategies than if the ratio is below this threshold, meaning that is is usually offline.

Barbará and Imieliński use *Invalidation Report* (IR) messages as the base strategy for cache management, and discuss several specific strategies for the usage of IR messages. The strategies are Broadcasting Timestamps, where the IR contains the timestamps of the changed data items' modification, Amnesic Terminals, where the mobile unit rebuilds the cache from scratch upon start up, and Signatures, where hash sums of data items are sent along with the IR.

Finally, "Sleepers and workaholics: Caching strategies in mobile environments (Extended version)" shows that for *sleepers*, the Signatures method is the most effective, where the periods of disconnection are long and difficult to predict. In contrast, for *workaholics*, Amnestic Terminals has proved to be the best method.

2.8 Cache Management with Invalidation Reports by Cao

In "Proactive power-aware cache management for mobile computing systems" [Cao02], Cao discusses an approach to prefetching in broadcast communication. It is based on the idea of *Invalidation Report* messages being sent by the server, and further develops the concept. For instance, *Updated Invalidation Report* (UIR) messages are proposed, containing only identifiers of data items changed since the last IR was transmitted, posing a *delta*, allowing clients to have shorter tune-in intervals while maintaining data consistency.

"Proactive power-aware cache management for mobile computing systems" also discusses a mechanism to selectively decide on prefetching data from a broadcast channel based on whether this item has been changed since the last IR. Subsequently, Cao shows how to use such an approach for adaptive prefetching by introducing the *Prefetch-Access Ratio* (PAR) as the number of prefetched divided by the number of accesses. This ratio can then be used as a relative indicator of the necessity for prefetching an item (or, in a different taxonomy, its utility); the paper also discusses how the threshold value of a prefetching decision can be dynamically adapted, either manually by the user or automatically based on attributes of the application context, such as the computer's current power level.

2.9 Fuzzy Adaptive Buffering by Bagchi

In "A Fuzzy Algorithm for Dynamically Adaptive Multimedia Streaming" [Bag11], Bagchi discusses an algorithm for buffer management at mobile client side in order to establish a good Quality of Experience by avoiding buffer overflow and underflow, while optimising energy consumption in order to save battery power. The algorithm is called Fuzzy Adaptive Buffering (FAB) and aims at the situation of a media playback by one client from a (streaming) server. The media is allowed to have a varying playback rate and the algorithm tolerates varying network latency (induced by a change in the overall network quality).

The proposed algorithm is based on the client-pull model and essentially constantly yields an answer to the two following decision problems:

(1) How much data has to be prefetched.
(2) How much time to sleep until the next prefetch should be performed.

The presented algorithm has several properties:

- It is logically located purely on the client side, while still being general in nature – the algorithm is a feedback loop with input received from the current buffer level and playback module.
- It is fuzzy and adaptive in nature, effectively tracking the bandwidth and playback rate while prefetching data blocks from the server.
- No a priori knowledge about the network (regarding performance parameters such as latency and bandwidth) or the media (regarding its playback rate) is required.
- Total sleeping time (with respect to CPU cycles) is maximised to enhance the saving of battery power.

2.10 Prefetching Heuristics by Gitzenis et al.

Gitzenis and Bambos model the data items and access probabilities as *Markov Chains* in "Power-controlled data prefetching/caching in wireless packet networks" [GB02]. After defining a thorough theoretical basis for the applied model, several heuristics for on-line look-ahead prefetching decisions are proposed:

- No Prefetching: A simple heuristic where data is fetched when it is required and not found in the local buffer.
- Neighbor Prefetching: A heuristic where items can be prefetched before they are requested, based on the Markov model and using a depth of one.
- Deeper Horizon Look-Ahead and On-Line Algorithms: The authors argue that a deeper search for prefetch candidates imposes higher computational complexity, however even a small search can already yield results leading to significant performance gain. Note that this heuristic class is merely a generalisation of Neighbor Prefetching.

The heuristics provides an on-line (continuous) solution to the following decision problems:

(1) What data should be prefetched into the cache.
(2) What data must be discarded from the cache.
(3) What power level should be used during the item transmission.

"Power-controlled data prefetching/caching in wireless packet networks" regards the decision problems as a trade-off between performance (in terms of transmission speed), which relates to problem (1), and power (in terms of transmission strength), which relates to problem (3).

2.11 Piggybacking Prefetch Data by Schreiber et al.

In "Reducing User Perceived Latency with a Proactive Prefetching Middleware for Mobile SOA Access" [Sch+11], Schreiber et al. present a method of prefetching data: additional data is being piggybacked onto regular responses, which reduces the overhead of creating a separate connection or protocol state transition for the prefetching process. This implies that the server is responsible for the prefetching decision. Subsequently, the prefetching decision can only rely on context transmitted explicitly (and a priori) by the client. Any data-rich context can not trivially be taken into account by the prediction mechanism, since transferring the context would outweigh the performance gain of prefetching.

Schreiber et al. show that prefetching and caching reduce *user perceived latency*, thus improving the Quality of Experience. Figures are provided for typical scenarios, such as using mobile data connections (GPRS, EDGE, UMTS, HSDPA) and local connections.

2.12 Dual Caching System by Han et al.

"A Semantic-Based Dual Caching System for Nomadic Web Service" [Han+13] proposes a dual caching architecture and development method of web services for mobile devices. The setup consists of two cache systems, a Client Side Cache (CSC) and a Provider Side Cache (PSC). Han et al. describe the cacheability of different types of data (permanent, stable, random) under different types of consistency constraints (strong consistency, eventual consistency, unconstrained consistency). Based on this classification, "A Semantic-Based Dual Caching System for Nomadic Web Service" proposes a caching strategy for both cache layers (CSC and PSC) and evaluates this strategy in an experimental implementation.

The paper concludes by stating that such dual caching approach overcomes problems arising from temporary loss of connectivity and fluctuations in bandwidth. Figures indicate that such dual caching improves loading time over a non-caching strategy.

2.13 Latency Reduction Strategies by Jayasudha et al.

In "Latency Reduction in Mobile Environment" [JV12], Jayasudha et al. propose a model for bandwidth usage reduction by prefetching URLs based on access probabilities. Uniquely to this paper, these probabilities do not depend on an individual user, but are global weights given to URLs.

For its prefetching decision, the algorithm assumes the bandwidth as well as the users of the network connection to be known. For deciding how many pages to prefetch, the pages are assumed to have an upper bound which is the only element of size used in the calculations (which is equivalent to the assumptions of all data items being the same size).

2.14 Advanced Loading Strategies by Zagarese et al.

Zagarese et al. introduce *Dynamic Offloading* into web service calls. In "Enabling Advanced Loading Strategies for Data Intensive Web Services" [Zag+12], the authors describe a mechanism which transparently offloads data chunk transmission in web service calls, postponing the actual transmission to the point in time when the data is actually needed (which is then *lazily loaded*).

The paper's goal is to predict access patterns in such a way that data unlikely to be used is not transmitted in the original message, reducing the overall amount of sent data, provided that the predicted accesses have indeed happened. Technically, this offloading happens transparently, meaning that the actual user code receives stub objects (for example by using proxy objects) and offloaded parameters or values are lazily loaded upon request.

Such offloading is used equally for IN and OUT parameters, in other words, for method arguments and return values. “Enabling Advanced Loading Strategies for Data Intensive Web Services” proposes a concrete middleware architecture, including client and server components.

The paper concludes that a simple learning technique, as well as random strategy, can still outperform pure-eager or pure-lazy loading, indicating that a lot of optimisation depends on the actual decision algorithm, all of which poses possible future work originating from these findings.

CHAPTER 3

Background

In order to lay down a basic understanding of our problem's notion, we present some background information gathered during our research. We start by defining some possibly ambiguous terms in section 3.1, and provide a taxonomy of data items in section 3.2 in which we discuss the various types of data sources possibly being a candidate for prefetching. Section 3.3 provides an overview of quantities and units used throughout this thesis to prevent disconcertment caused by differing ideas of units.

3.1 Term Definitions

Data Ageing Prefetching data immediately leads to the fact that data that has been fetched before its actual usage is in some way *old* at the time of use. In some cases data ageing poses a problem – data ageing issues are discussed throughout section 3.2.

Data Chunk Data items are not always transferred as a whole, but sometimes split into parts, which we call data chunks.

Data Item In generic terms, a data item is data originating from some kind of source (a service); such data can be fetched by a mobile unit. Data items have a variety of types, which classify their size, their fetching mechanisms and strategies, amongst other aspects – see section 3.2 for a summary of the most relevant types of data items.

Mobile Unit For our purposes, devices which are part of a distributed system and capable of being moved over longer distances are called mobile units. They are usually connected to the distributed system via some type of mobile connectivity, e.g. 3G or 4G wireless network. The main properties of mobile units besides mobility are fluctuating network connection quality, limited bandwidth, high network latency, as well as frequent usage of battery power, thus aiming to conserve as much energy

as possible to enable long battery running times. This is an enhancement of the notion of mobile units introduced in [BI95].

Quality of Experience While the experience of a user performing an action using a system can be quantified by variables such as delay, stretch, power usage and so on, we use the term Quality of Experience (QoE) described in [Jai04] and [BH10] to denote in a rather generic and informal manner the way a user is experiencing a system. QoE is different from Quality of Service (QoS) in the way that it is not measured using objective methods, but tries to subjectively quantify the user's satisfaction with the service. For instance, a high delay (or stretch) and rather poor QoS – irrelevant content, unsuitable media formats, and so forth – constitute poor QoE, while a short delay (possibly due to prefetching) together with good QoS can be seen as a good overall QoE. This distinction of the two related terms QoE and QoS is further explained in [Mok+11].

3.2 Taxonomy of Data Items

For this work, we classify types of data items in order to describe their behaviour in the context of prefetching and caching. Such a distinction is inspired by [Hum+14] (see Table 1 in said paper), where services are also classified, using aspects such as *Importance* (a fuzzy rating used for prioritising amongst concurring requests), *Time Criticality* (an indicator of the impact of *data ageing* on the service quality), an *Access Pattern* (from which we derived our types of data items) and the resulting *(Pre-)Fetching Strategy* (determining the strategy used for the handling of requests for the given service).

Our classification is closely related to the one given in [Hum+14], and specifies the main properties of each type of data items. In the respective sections, we also discuss three main attributes of each class:

Bandwidth Specifies how bandwidth-intensive the data item is. This usually correlates with the size of the data item in bytes.

Timing Specifies whether the data item has timing constraints. Such constraints can either consist of strict real-time requirements, such as maximum latency, maximum jitter or loose specifications like the ageing of data items and the represented information becoming outdated over time.

Suitability for Prefetching Specifies whether the data item is suitable for prefetching and what constraints have to be taken into account when choosing a prefetching scheme for such data items.

3.2.1 Streaming Data

We classify data items consisting of a continuous stream of data as Streaming Data. We identify such data items using the following indications:

(1) The data item is not supposed to be stored in the mobile unit's memory, either because its total size exceeds the memory resource, or because doing so is not feasible, for instance because it is a live media stream and mainly not intended to be stored. A permanent storing of the received data can be seen as a *recording* of the stream.
(2) Even though the data item is transmitted in chunks, it is generally seen as one coherent chain of data. Chunking is only employed to allow for partial transmission and simultaneous playback, but the chunks are not independent of each other by design. This means that the loss of one chunk is generally not tolerable for a stream receiver, even though established streaming techniques usually circumvent this by using formats which tolerate transmission gaps, for example MPEG, where I-Frames are inserted, which can be decoded independently of other frames and constitute a point of re-start in case of loss of data chunks. The point where a chunk split is performed is completely arbitrary with respect to the stream content and the decision of splitting is made solely with regards to transmission control.
(3) The data item is consumed simultaneously to its fetching, often with a buffering mechanism to avoid underruns, ensuring a constant Quality of Experience.
(4) The consumption (playback) of data items has a minimum required bit rate of arriving data – in other words, playback rate is not adjustable, which means that buffer underruns cannot be amortised by lower playback. Instead, in case of an underrun, playback must be paused until enough data is available again. The minimum required bit rate is not necessarily constant, and can in some cases be actively adapted to network conditions – see (2) and (3) under the discussion of bandwidth.
(5) Once the stream transmission starts, chunks of the data stream are transmitted by the sender without explicit request by the receiver. This does not exclude transmission control mechanisms such as buffer overrun prevention or Adaptive Bit Rate – see (3) under the discussion of bandwidth.
(6) The stream takes up a certain amount of bandwidth continuously throughout the duration of transmission – the data link can be seen as mainly busy.

Examples of Streaming Data include VoIP calls transmitted to the mobile unit (possibly with an additional video feed, making it a video call), a movie being played back from a Video on Demand provider, or music played back from a personal music cloud provider. [Bab+02] provides *financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others* as examples for data streams.

Note that we differentiate between this traditional intuition of streams and Periodic Updates, which could also be argued to be a special case of Streaming Data. For our requirements, however, we do not use this classification – see section 3.2.2 for further information about the differentiation used in our work.

Discussing Streaming Data in terms of the presented taxonomy, we define the following attribute properties:

Bandwidth While we classify Streaming Data as rather high-bandwidth applications, the actual used bandwidth may vary from case to case. Not only will a video call have a higher bandwidth requirement than its audio-only variant, in case of audio and video streams, different formats used for encoding (produced by different codecs) can significantly change the amount of data required.
There are several variants of Streaming Data behaviour in terms of bandwidth:

(1) Streams having a static bit rate are classified as *Constant Bit Rate* (CBR) streams [LT98]. Such a property facilitates the prediction of data usage at any point in time, but also significantly reduces the adaptability to changing network environments.
(2) Multimedia streams often have the ability to adapt the effective bit rate to changing stream properties. Such streams are usually classified as streams with *Variable Bit Rate* (VBR) [LT98]. The main property of of VBR streams is that the bit rate depends on the complexity of the stream; for example, large amounts of movement in video frames cause a higher bit rate than a steady shot. Often, VBR is an effect of compression, where repeating patterns are compressed either using lossless or lossy compression.
(3) In network streaming, the term *Adaptive Bit Rate* (ABR[1]) refers to stream providers which allow clients to selectively decide what bit rate they consume. [Zha+13] states that using ABR, content is "transcoded into a set of media files with diverse playback rates, and appropriate files will be dynamically chosen in response to channel conditions and outlet forms". This choice is usually automatically made by the client software monitoring network performance, to use the highest possible bit rate while avoiding network congestion. Naturally, since the client chooses the bit rate independently of the original stream's bit rate, the only way of satisfying this choice is by employing data compression, leading to lower quality when a lower bit rate is chosen.

Timing Timing is often a critical aspect of Streaming Data. There are several types of timing constraints to streams:

Minimal Jitter For a continuous and fluent consumption (playback) of a data stream in real-time applications, it is necessary to have minimal jitter.[2] High-jitter network transmission leads to frequent buffer overrun and underrun, both of which cause interruptions and degrade Quality of Experience.

Minimal Delay Streams containing real-time data, often used for bidirectional communication such as VoIP calls or video conferences, require an upper bound for end-to-end transmission delay[3] to allow for a fluent communication without

[1] Not to be confused with *Available Bit Rate*, for which the abbreviation ABR is also used.

[2] In this context, *jitter* refers to *delay jitter*, i.e. the variability over time of the delay between a packet's sending and receiving time.

[3] The precise formulation of the requirement is an upper bound for the delay between the media content being generated by the sender, e.g. spoken, and being consumed by the receiver, e.g. heard. However, since encoding and decoding performance is not in the scope of this work, we assume that time required for encoding and decoding is negligible and focus on transmission time.

generating silences between two parties' statements generally perceived as awkward.
There are stream types where *Minimal Delay* is not as crucial as *Minimal Jitter*, such as unidirectional communication (broadcasts) – in this case, a much higher upper bound is put on delay – or playback of stored media, where the delay is almost irrelevant.

Data ageing is usually not an explicit issue for Streaming Data, since the requirements of minimal delay limit the data age to a very short amount of time. In cases where minimal delay is not required, for example unidirectional communication, data ageing can be crucial.

Suitability for Prefetching Streaming Data can be prefetched and stored under some circumstances. In real-time applications such as communication, prefetching is not possible by nature, however, if the sender uses prerecorded (or generated) data as a source, prefetching can actually be realised by requesting chunks which otherwise would be transferred at a later point in time (*future* data).
Apart from this issue, Streaming Data can be prefetched with respect to available resource. Since we argued that streams often have a rather high bandwidth requirement, prefetching them results in a high usage of memory on the client side.

Streaming Data is generally not classified as either *push* or *pull*, since it contains elements of both strategies. The initial request for a stream usually follows the *pull* scheme, with the client explicitly requesting the stream transmission, but the following chunks are sent by the server without explicit subsequent requests – see also (5) –, which follows the *push* scheme.

3.2.2 Periodic Updates

Data items where the server pushes chunks of data in a periodic manner, updating data on the client side, are classified in our taxonomy as Periodic Updates. In contrast to Streaming Data (section 3.2.1), data from items using Periodic Updates is not *constant*, i.e. does not constitute an uninterrupted chain of data segments or stream. Hence, the key difference between Streaming Data and Periodic Updates is that the former requires a buffer, which should always be non-empty, to allow proper consumption of the data, while the latter does not require buffering, since data chunks can be processed immediately upon reception. It is also worth noting that Periodic Updates usually has significantly more time in which the data link is idle, while streams cause the data link to be mainly busy.

The properties we use to classify data items as Periodic Updates are as follows:

(1) The size of one data chunk is small enough to be stored in the mobile unit's memory and processed immediately (even though it can also be saved for later processing). This is in contrast to Streaming Data properties (1) and (3).

(2) The data chunks are structurally independent of each other. This property can manifest itself in the way that the loss of one data chunk does not cause the complete transmission to be corrupted. This is in contrast to Streaming Data property (2).
(3) Once the transmission starts, chunks of data are transmitted by the sender without explicit request by the receiver. This does not prohibit transmission control mechanisms such as buffer overrun prevention; this property is analogous to Streaming Data property (5).
(4) The data link is only used whenever an update chunk is transmitted, but it is idle most of the time; this property is in contrast to Streaming Data property (6).

Examples of Periodic Updates are a road service providing the client with traffic information en-route, allowing the client software to dynamically re-route the driver in case of heavy traffic. Another example is a periodic status update of the user's smart home, consisting of the home lock status, temperature, movement detection status, a camera snapshot or similar. One can also implement electronic mail delivery in such a way, as it is the case with the Internet Message Access Protocol (IMAP).[4]

Discussing Periodic Updates in terms of the presented taxonomy, we define the following attribute properties:

Bandwidth By its nature of being small enough to be stored and processed in the unit's memory – see property (1) –, Periodic Updates data usually shows rather low bandwidth usage, especially when observed for a longer time and averaged, since most of the data time, the data link is idle – see property (4).

Timing While Periodic Updates data items tend to have loose timing constraints compared to streams, there are cases where even requirements similar to real-time applications can be observed. One example of such an situation is a media server communicating the current status to a front end client, including the seek position in the current song. This position is under constant change (provided that the song is playing), which means that a delay of one or two seconds invalidates the information. However, it is arguable that most cases of Periodic Updates are rather tolerant of delays; notification of an incoming e-mail can indeed have a delay of a few seconds without degrading user experience, as it is also the case with traffic updates.

Similar to Streaming Data, avoiding data ageing of fetched items can be a requirement, depending on the application. In some cases (e.g. traffic information) a delay of a few minutes is acceptable, while in other cases (e.g. movement detection status) the excess of sub-second durations in data ageing greatly affects user experience.

Suitability for Prefetching Data originating from Periodic Updates is subject to the same restrictions as Streaming Data regarding the ability to be prefetched only

[4] IMAP only uses a push-like mechanism for notification of new messages, the client has to fetch the messages on its own, which is not an example of Periodic Updates in the classic sense anymore since it contradicts property (3).

when the data is actually available to the system at a sooner point in time than it would originally be fetched (non-real-time data); this is a design issue.
Apart from this restriction, there is generally no reason for data items to be unsuitable for prefetching, especially since we argued that Periodic Update data items are generally less bandwidth-intensive, and thus need less memory to be prefetched on the client side. Furthermore, Periodic Update shows a repeating pattern or periodicity, which eases the prediction of future accesses.

3.2.3 Intermittent Pull

We classify data items which can be requested by the client on-demand in a *pull* manner, and do not follow a periodic pattern, as Intermittent Pull. The term *pull* designates the notion that the requests happen either upon immediate user action, or because change in the application context dictates so.

Note that Intermittent Pull can overlap with Streaming Data in the way that a request for starting a stream transmission is classifiable as an Intermittent Pull request; nevertheless, we differentiate Intermittent Pull from Streaming Data using the following differentiation properties.

The properties used for identifying Intermittent Pull in our taxonomy are as follows:

(1) The size of the data item is small enough to be stored in the mobile unit's memory and processed upon reception (even though it can also be saved for later processing). This is analogous to Periodic Updates property (1).
(2) The fetching of a data item is generally perceived as an *event*, not a *process*. It takes a relatively short amount of time, the major reason for delay is the end-to-end network delay, the server-side processing (generation) time for data items and the computation time needed for initiating the transmission, not the actual transfer itself.
(3) The data item is structurally coherent and constitutes a single unit of data (although it can be of any complex type). There is no periodicity involved. In general, subsequent fetches of the same data unit can yield the same result (e.g. requests for a static web page) or different results (e.g. requests for traffic information). If the information changes, it is generally independent of the requests themselves. This relates to Periodic Updates property (2).
(4) The data items are transferred only using a *pull* scheme, i.e. the client side explicitly requests the data item, and after complete transfer, not further data transmission occurs. This is in contrast to to Streaming Data and Periodic Updates properties (5) and (3), respectively.
(5) The data link is only used whenever a data item is transmitted, and is idle the rest of the time; this property is similar to Streaming Data property (4).

Intermittent Pull includes all kinds of requests made from the client side to the server side, including websites, map services, e-mail fetching requests and similar.

Discussing Intermittent Pull in terms of the presented taxonomy, we define the following attribute properties:

Bandwidth Since Intermittent Pull denotes the fetching of a data item for on-demand usage, and we have established that the difference between Intermittent Pull and Streaming Data is the fact that the latter is seen as a long-lasting process while the former is seen as an event – see properties (1) and (2) –, Intermittent Pull by logic only describes data items where fetching takes a rather short time. This leads to the conclusion that bandwidth usage for fetching Intermittent Pull data items is rather small.

Timing As discussed in property (2), the transmission time for Intermittent Pull data items is negligible. This means that the experienced delay for fetching a data item is solely caused by the delay induced by initiating data transfer. Timing constraints for Intermittent Pull data items are usually bound to Quality of Experience requirements, where opening a web page, scrolling over a map or fetching an e-mail is expected to happen with minimal or no delay, and any delay is perceived as disruptive.
Data ageing is, similar to Periodic Updates, heavily dependent on the type of data transmitted. Acceptable ages of fetches can range from fractions of a second (e.g. camera snapshots) to months (e.g. map chunks).

Suitability for Prefetching Data items following the Intermittent Pull scheme are usually easy to prefetch (in the sense of implementation), since the request can simply be sent ahead of the actually necessity for data, caching the response and answering the later (actual) request with the response from cache (this assumes that data ageing is taken into consideration). However, as simple as implementing prefetching is, the prediction of actual demands of data is significantly harder than with Streaming Data and Periodic Updates, since no pattern is involved. Depending on the application, a prefetching mechanism has to predict user actions or context changes. In some cases this is a matter of applying methods from the field of Artificial Intelligence, in other cases this may be easier, for example when the route of the user is known, map chunks needed alongside the route can be prefetched.

3.2.4 Summary

To summarise the types of data items, Table 3.1 shows our taxonomy of data item types and their main properties. We use the terms described in this section to roughly describe data items during our discussion in the subsequent sections, as well as in our simulation environment.

Table 3.1: Summary of the data item type taxonomy

Item Type	Bandwidth	Timing Constraints	Suitability for Prefetching
Streaming Data	high	min. jitter, delay	only non-real-time; RAM intensive
Periodic Updates	low	mostly loose	well suitable (periodicity)
Intermittent Pull	low	mostly strict	hard to predict

Table 3.2: Unit of data: multiples of bits

Long	Short	Meaning
1 bit	1 b	base unit *bit*
1 kbit	1 kb	10^3 bit
1 Mbit	1 Mb	10^6 bit
1 Gbit	1 Gb	10^9 bit

3.3 Quantities and Units

For the purpose of this work, we establish three main quantities and describe some of their properties. The quantities are described using their name, their symbol and their unit.

3.3.1 Time

Time in our context relates to points in real time or durations thereof. For those real time quantities, we use the symbol t and the SI unit *second* (1 s) as the unit of time for the sake of simplicity, familiarity and easy applicability to common scenarios. Where applicable, we use the commonly accepted everyday units for time, *minute* (60 s) and *hour* (3600 s).

However, any strictly ordered dimension is sufficient for our purposes; for example, the concept can easily be applied to any continuous dimension independent of real time, such as simulated time or time slots. Where applicable, we abstract from real time and use the auxiliary unit *tick* ($1\,\mathcal{T}$).

3.3.2 Data

In our context, data[5] is a quantity for the amount of payload that is being fetched or prefetched. We use the symbol D and the *bit* (1 bit or 1 b[6]) as the unit of data. As prefixes, we strictly use the SI prefixes *kilo* (one thousand), *mega* (one million) and so on. We do not use prefixes of binary multiples – steps of 1024, for instance *kibi-* (1024) or *mebi-* (1024^2) – in our work; see table 3.2.

Where necessary or practicable, we use *byte* (1 B) as an alternative unit, where naturally 1 B = 8 bit. Analogous to bits, we strictly use SI prefixes for bytes; see table 3.3.

[5]Data, as as opposed to *information*, is not parsed; however, we use the unit of information as a unit of data for our purposes, since the distinction between data and information is irrelevant.

[6]The use of 1 bit is less ambiguous since 1 b can be mistaken for byte, but 1 b is sometimes used in figures for the sake of shortness.

Table 3.3: Auxiliary unit of data: multiples of bytes

Short	Meaning
1 B	base unit *byte*
1 kB	10^3 B
1 MB	10^6 B
1 GB	10^9 B

3.3.3 Bandwidth

Transmission of data takes time, and the quantity describing the speed of transmission of data used in this thesis is *bandwidth* or *byte rate*; the following paragraphs provide detailed differentiation of these two terms.

In our context, bandwidth is the rate of (transmitting) *data per time*:

$$B = \frac{D}{t} \tag{3.1}$$

Alternatively, Bandwidth can be seen as the momentary speed of (the transfer of) data over time. Since data is not continuous but discrete in nature, the infinitesimal notation is only for illustration purposes and does not reflect reality.

$$B = \frac{\mathrm{d}}{\mathrm{d}t}D \qquad\qquad B = \frac{\Delta D}{\Delta t} \tag{3.2}$$

We use the symbol B and the derived unit *bits per second* ($1\,{}^{\mathrm{bit}}/_{\mathrm{s}}$) for bandwidth. Analogous to data, we use bytes where necessary or practicable for bandwidth ($1\,{}^{\mathrm{B}}/_{\mathrm{s}}$). Following from our definition of using bits and bytes as units of data, we strictly use SI prefixes (multiples of 1000) rather than binary multiples (multiples of 1024).

Throughout this thesis, the terms *bandwidth* and *byte rate* appear side by side. While, at least in the context of digital transmissions, they seem to often be used synonymously, we try to distinguish the meaning and use the term *bandwidth* whenever the abstract idea of (infinitesimal) *speed* of data is used, and the term *byte rate* whenever an actual (integer) amount of bytes in an (exact) time span is described.

While *bandwidth* may sometimes refer to available speed of data transfer, or to the property of a transmission channel, and *byte rate* refers to the amount of data actually transferred over such channel, we do not use this differentiation, and instead use terms such as *available byte rate* for that purpose. We furthermore do not consider aspects of bandwidth in analog contexts or contexts of signal processing, where bandwidth is the difference between the bounds in a continuous band of frequencies.

3.3.4 Percentage Points

When using quantities provided in percent (1 %), it is in certain cases necessary to refer to *percentage points*. In particular, this is necessary when denoting changes in quantities – for example, a change of link usage from 50 % to 60 % is a change of *10 percentage points*, rather than *10 percent*. Another example is the standard deviation σ of random variables with the original unit 1 %. When describing σ, we need to use the correct unit *percentage points*.

To explicitly denote percentage points in cases where typesetting does not allow the usage of the full term *percentage points*, we use the abbreviation %p. In our previous example, the link usage has changed from 50 % to 60 %, which is a change of 10 %p.

CHAPTER 4

Proposed Algorithm

In this chapter, we discuss all necessary information to create a solution for prefetching data on a mobile unit to enhance the Quality of Experience. In order to achieve this, we develop an algorithm for prefetching data chunks. More specifically, our algorithm solves the decision problem of finding a time for fetching data items, given a set of requested data items and those requests' properties.

The first section of this chapter, section 4.1, discusses briefly the present approaches and models in existing related literature; the goal is to describe how our model and algorithm are related to and inspired by present work. Section 4.2 describes said model and its variables and dimensions, as well as the input variables for our algorithm stemming from the described model. Section 4.3 discusses in depth what our target variables, the quantities we are aiming to influence, are.

After discussing the groundwork, we elaborate on the requirements imposed onto the algorithm in section 4.4, which allows us to finally present the idea and realisation of our proposed algorithm in section 4.5. A reference implementation written in Java is briefly shown in section 4.6.

4.1 Discussion of Existing Approaches

We generalise the idea of *prefetched data* vs. *non-prefetched data* and simplify the two decisions of *whether and when to prefetch* R_i to the single decision of *when to fetch* R_i. The result of this decision is a point in time at which fetching is scheduled to be started and is referred to as λ_i throughout this work. The schedule set (set of resulting schedule times for all requests) is defined as $\mathcal{S}$ with one schedule $S \in \mathcal{S}$ defined as $S = (R, \lambda)$.

This is in contrast to [Hum+14] – see equations (1) and (2) in that paper –, where a boolean condition determines whether the available quality q_a is sufficient for the required quality q_r, for example in the following condition: $q_a(c_x) > q_r(r_x, m_x, c_x)$ – this condition

is true if the available network quality is greater than the required network quality for the given request r_x for a certain mobile service consumer m_x and its context c_x.

A notion we adopted from [Hum+14] is the *look-ahead time*, which denotes how long into the future a prefetching decision is made – the paper has a fixed parameter for this, t_p; we discuss this parameter and the possibilities of its variation over time in section 4.5.3.

[Hum+14] uses a set of services (S) to describe the used cloud services. Services have certain properties which are comparable to our notion of *requests* (requests do also have an own model term, R, in [Hum+14]). Additionally, we have adopted the notion of an a priori known network quality, q_a, which is comparable to our function B_{link}; both map a point in time to the corresponding network quality, and both are used to create prefetching decisions.

[Tua+98] follows a *document-centred*, *cost-based* approach; the authors discuss prefetching of *documents* within the viewing time of other documents. A similar approach is found in [Han+13]. For our purposes, this approach is too specific to be suitable; while the notion of documents can be transferred into our notion of data items, we do not introduce the quantity of *viewing time* into our model because not all data items have applicable concepts. For example, a background data service periodically fetching electronic mail from a remote server and displaying a notification upon arrival of mail would not have any viewing time, while a navigation system loading map tiles on-the-go would have virtually perpetual viewing time.

In [She+05], the authors follow yet another approach; the access of items is assumed to be *recurring*, and the decision problem is whether to keep items in cache, and which items to evict from cache upon arrival of new data to be cached. This model is partly overlapping with our model; a recurring request can be modelled in our approach as multiple requests to the same data item source, while a single request from our model would correspond to an item from [She+05] which has a low hit rate. The authors of [She+05] describe a *utility function* to decide which items to evict from cache. A similar approach is found in [Yin+02], [GB02], [JV12].

Since we do not handle the case of actually caching results beyond their required time, our approach does not require such utility function or eviction strategy. A downside of this approach is that multiple accesses to the same resource, which might be cached for future access, are re-scheduled and re-fetched in our model. This is a possible field for future work and is discussed in section 7.2.

[Cao02] and [BI95] cover strategies for distributed cache invalidation, where a server is broadcasting *Invalidation Report* (IR) messages. An IR contains the timestamps of changed data items' modification and allows clients to update or flush their caches accordingly. For our purpose, this model is not applicable for several reasons:

(1) A broadcast channel is assumed, which is not generally true for packet-switched networks such as 3G networks in our case. Some providers of 3G networks (or internet service providers in general) might not forward broadcast and multicast

packets, so a broadcast channel would be hard to realise without explicitly sending data to each client; such an approach would defy the purpose of using broadcast channels to reduce traffic duplication.

(2) Data items in our approach are dynamic and individual in nature, and, as such, are generally not shared amongst clients. While some requests might yield the same result for each client, we handle each request independently. Note that this is not a general assumption but a simplification, and provides some ground for future work, by optimising the algorithm presented here for situations where broadcast is indeed available.

(3) Caching is not performed beyond the data item's usage in client code, as described in the previous paragraphs. We only cache data for the time until its request is served, after which it is evicted from cache. This, in fact, satisfies one of our target variables – see section 4.3.2.

[Bag11] in turn regards the case of a constant data stream, and focuses on the buffering mechanism of the playback (consumption) of such a data stream. An algorithm called Fuzzy Adaptive Buffering (FAB) is developed, which aims to provide a solution to the decision problem of how much data to buffer, and when (in other words, how much time to sleep until the next fetching decision is due). The algorithm presented by [Bag11] is suitable for a constant playback, and our notion of *Streaming Data* (see section 3.2.1) could be combined with FAB for further optimisation of energy usage on the client device. Since energy usage is not a goal in the scope of our work, we reserve this potential for future work (see section 7.2).

[Zag+12] uses an interesting approach for intelligent prefetching, albeit on a different layer, thus not directly applicable to our problem: the paper presents a way to predict access patterns in web services, so data which is unlikely to be used is not transmitted in the original message (request or response). The overall amount of transmitted data is reduced, leading to a more responsive service. The offloading is realised transparently for the higher software layers, meaning that objects returned by the middleware responsible for this offloading are proxy objects which lazily load the offloaded data upon request. While this approach is very interesting, we could not apply it in our case since we treat data as a byte stream without any further semantics; a deeper inspection of the higher-level protocol (or a definition thereof) would enable a combination of our prefetching algorithm and [Zag+12]'s offloading mechanism.

[Sch+11] presents an novel idea on reducing latency; piggybacking data onto regular responses. Specifically, a server, given a request to resource A, analyses this resource and determines predictions on the likelihood of other resources B being subsequently accessed by the client requesting A. The server then appends the content of resource B to the original response to request A. Similarly to [Zag+12], we can not directly use this approach because we have no knowledge about the inner structure of the data we transmit as payload.

Altogether we have decided to generalise our approach by assuming a unicast (point-to-point) scenario, neglect the evidence that data might be recurring and re-usable among

subsequent requests and abstract the request to payload data, not taking into account the internal structure of the data being transmitted. While this allows us to cope with a wider range of requests, it has to be noted that certain optimisations, as the ones stated in the last paragraphs, become unavailable.

4.2 Model

The top-level goal of the algorithm is that for each request $R \in \mathcal{R}$ a scheduling decision is to be made – as such, this goal led us to the creation of the following model to describe the required properties and describe the proposed algorithm.

A request R is defined as $R = (\tau, \beta, \delta)$, where τ is the request deadline (specifying at which point of time a data chunk must be available in local memory), β specifies the maximum bandwidth provided by the data source, and δ is the amount of data (designating how much data is required to be transmitted with the given bandwidth to fetch the data chunk). The algorithm also takes as input a bandwidth prediction function $B_{link} : t \to B$, which assigns to each point in time an estimate (a prediction) of the available bandwidth.

Note that the request variable R as well as its attributes (τ, β, δ, λ) are used with the index i in situations where the distinction to other requests (R_i, not R_j) or the explicit association of an attribute with its request (τ_i is explicitly the deadline of R_i) is necessary. In other cases, where no distinction is necessary and the semantics are clear, the variables are used without an index for better readability (τ, when referring to any request's deadline).

Summarising, the following symbols are used for attributes of a request object R_i and constitute the input parameters for our algorithm:

τ_i ..... deadline of R_i
β_i ..... maximum bandwidth of R_i
δ_i ..... data volume of R_i
$B_{link}(t)$ ..... predicted available bandwidth at time t

The following symbols are used to designate values derivable from the symbols described in the preceding list, as well as λ, our algorithm's outcome:

T_i .... expected duration of transmitting δ_i bytes at bandwidth β_i.[1]
L_i .... expected time of completion, if fetching starts at λ_i.[2]
λ_i .... scheduled time of fetching for R_i (this is the outcome of our algorithm)

We see that the data chunks will be available upon request if and only if the reception of the data item (L_i) is before the request's deadline (τ_i). On the other hand, upon receiving the data, the prefetched content must be stored (cached) in memory until the data is requested. Thus, the cache duration is $L_i - \lambda_i$.

4.3 Target Variables

In this section, we identify the specific values that we seek to optimise, i.e. the *target variables* of an algorithm suitable for our problem. For each variable, we denote the target condition as optimisation formulae. Furthermore, for each target variable, the respective variable in the model (section 4.2) and the relation between the target variable and the algorithm variable is discussed. The target variables discussed here furthermore relate to the metrics recorded in our simulation, which are described in chapter 5 (see section 5.6 for a listing of those metrics).

4.3.1 Perceived Response Time

Perhaps the most important variable is the perceived response time $\mathcal{RT}$, which is the duration that the user has to wait after (explicitly or implicitly) requesting a certain data item – for example by selecting it from a menu – until the data is usable. Note that this may also be an indirect (implicit) process, where a running process is constantly requesting data (see section 3.2) and the user is not explicitly opening files. In this case, we define the perceived response time as the time from the application's request to the middleware (which is using the scheduling algorithm) until the actual disposition of the data to the process.

In the description of our proposed algorithm, we denote the *deadline* of a certain request as τ – this already assumes that there is the notion of a foreseeable *plan* as opposed to a non-predicted *request* –; the algorithm then schedules a time of fetching (λ), which results in a time of completion L. Therefore, the perceived response time which is to be minimised by the scheduling algorithm is $\mathcal{RT} = \max(0, L - \tau)$. If the algorithm manages to schedule the fetching *in time*, i.e. in a way that allows the complete data item to be fetched before its request, $L - \tau$ becomes negative and $\mathcal{RT}$ is zero.

We therefore aim to minimise the $\mathcal{RT}$ of all requests. We do not differentiate between requests – no prioritisation is taken into consideration – and therefore establish that the goal of the algorithm is to minimise the average perceived response time of all requests, $\overline{\mathcal{RT}}$, which yields our *first-order condition*, (4.1).

[1]The value of T_i is derivable from δ_i and β_i by the following formula: $T_i = \frac{\delta_i}{\beta_i}$ – this follows from equations (3.1) and (3.2).

[2]The value of L_i is derivable from λ_i and $B_{link}(t)$ and all other scheduled fetches. In our further work, we limit ourselves to fetching one chunk at a time and avoid simultaneous fetching of data, which eases the calculation of L_i.

4.3.2 Data Age

A trivial solution after specifying (4.2) is to fetch data items as early as possible, so the corresponding trivial scheduling algorithm yields $\lambda = t_0$. This would lead to *data ageing* (see section 3.1) and high memory usage (because all items necessary at any time need to be kept in cache). For some data item types, low data ageing (or data *freshness* is a functional requirement – for an example, see section 3.2.2 –, and reducing memory usage is in any case an effort towards increasing Quality of Experience, as free memory allows for more concurrent applications to be run by the user.

We therefore denote the data item age as $\mathcal{DA}$. $\mathcal{DA}$ can be regarded in context of our proposed algorithm as the time span between the prefetching of the data item (denoted by λ[3]) and the actual usage by the application (the deadline, denoted by τ). The resulting age to be minimised by the algorithm is $\mathcal{DA} = \tau - \lambda$.

Similarly to $\mathcal{RT}$, we achieve $\mathcal{DA} \rightarrow \min$ by minimising the average of the data age of all requests. Our *second-order condition* follows in (4.2), which means that assuming condition (4.1) is satisfied, the latest possible λ must be used.

4.3.3 Data Volume

An aspect not considered so far is the amount of transmitted data. Since fetches may be repeated, the amount of data transferred is variable; indeed, there is another trivial solution available, which would be to fetch all data items for which the deadline lies in the future, with some constant interleaving time Δ employed to allow for some interleaving between two fetches. Each fetch overwrites the result of the preceding fetch for the data item.

$$\lambda = \{t_0, t_0 + \Delta, \ldots, \tau_i\}$$

The advantage of such a solution is that $\mathcal{RT}$ is zero[4] since regardless of any network conditions, no delay is experienced by the user.

Meanwhile, $\mathcal{DA}$ is kept at the absolute possible minimum and is only bounded by:

(1) The fetch interval, which is the sum of the interleaving time and the transmission duration: $\Delta + T$
(2) The last time a successful data fetch was possible possible at all, due to network conditions: $\tau - t_{lastSuccessfulFetch}$

[3]We use λ instead of L because we assume that data ageing starts at the *start* of the transmission, since this is the latest possible point in time for the server to update its readings of values.

[4]Assuming that the data item can be fetched at all, otherwise $\mathcal{RT}$ is not defined since no successful fetch is possible.

$$\overline{\mathcal{RT}} \rightarrow \min, \qquad \text{where } \mathcal{RT} = \max(0, L - \tau) \tag{4.1}$$

$$\overline{\mathcal{DA}} \rightarrow \min, \qquad \text{where } \mathcal{DA} = \tau - \lambda \tag{4.2}$$

$$\mathcal{DV} \rightarrow \min, \qquad \text{where } \mathcal{DV} = |\lambda|\ \delta \tag{4.3}$$

Figure 4.1: Target variables as optimisation formulae

[Hum+14] uses a strategy similar to this solution: periodic prefetches of data are employed to cache values in case the actual fetch – which is still performed, regardless of the presence of cached data, which is in contrast to our solution – fails.

However, the downside of such a solution is that the data item is transmitted multiple times, and, depending on the network quality, the amount of transmissions might be highly excessive. This has several disadvantages:

- High network usage, which itself prevents other applications from using the available bandwidth, and might lead to costs for the user.
- Increased Power consumption, which is crucial with mobile units.

Therefore, we denote the transferred data volume as $\mathcal{DV}$ and aim to minimise it. For this purpose, we introduce a *third-order condition* in (4.3), which forces the amount of data transferred to be minimal. $\mathcal{DV}$ is defined as $|\lambda|\ \delta$ since the data amount δ has to be transferred as many times as a fetch is scheduled, which we denote as $|\lambda|$. Note that in the proposed algorithm, $\mathcal{DV} \rightarrow \min$ is trivially satisfied since only one scheduled fetch time is returned by the algorithm (the last possible one). This, however, increases the risk of a fetch not being successful due to the network conditions being worse than expected; this risk is amortised by several strategies, as explained in section 4.5.3.

4.4 Requirements

The outcome of the scheduling algorithm underlies two key requirements, derived from the set of target variables in section 4.3. On one hand, the expected time of completion must be before the deadline,[5] which is expressed in condition (4.4); on the other hand, the time of the data chunk stored in memory must be minimised to prevent data ageing and prevent unnecessary resource usage. This is formally stated in condition (4.5).

$$L_i \quad < \quad \tau_i \tag{4.4}$$

$$\tau_i - L_i \quad \rightarrow \quad \min \tag{4.5}$$

[5] Even though strictly speaking, the relation between the time of completion and the deadline is of *less-than-or-equals* nature, since time is in this context expressed as a continuous dimension, we use a *strictly-less-than* relationship for simplicity.

These two conditions correlate with the target variables (section 4.3). Condition equation (4.4) causes the data item to be fully fetched before the deadline, which leads to the item being present in cache upon request. It follows that the perceived response time resulting from this condition is zero: $\mathcal{RT} = 0$ – this trivially satisfies target variable condition (4.1).

Condition (4.5) is derived from condition (4.2) and causes the data age to be minimal ($\mathcal{DA} \rightarrow \min$). Otherwise, the trivial solution would be to schedule all fetches at $\lambda_i = t_0$, which would cause high memory usage and high data ageing effects (see section 4.3.2).

Note that condition (4.3) (minimal transferred volume) is trivially satisfied since our algorithm only returns one fetch time decision per request, which means that no repeating fetches of a single data item occur.

4.5 Scheduling Algorithm

As described in section 4.2, the scheduling algorithm takes as input a set of requests $\mathcal{R}$ with $R \in \mathcal{S}$ and $R = (\tau, \beta, \delta)$, as well as a function $B_{link} : t \rightarrow B$ specifying the predicted available bandwidth at any point in time. The expected output is a schedule set $\mathcal{S}$, where $S \in \mathcal{S}$ and $S = (R, \lambda)$. R is the request in question and λ is the time at which the fetching of the result data should be started in order for the deadline (τ) to be met.

4.5.1 Core Idea

The idea behind the algorithm is that diminishing link quality also lowers the transmission bandwidth of data fetches, and in order to ensure finishing the fetch in a timely manner, it must start earlier, according to the lower transmission bandwidth.

We assume that data volumes can be described as integrals of data rates (this assumption is based on the notions described in section 3.3.3 and explained in further detail in section 4.5.2). Following this assumption, we think of data volume as the area below the data rate in the diagram. In figure 4.2, the areas are represented as red and green rectangles starting at λ_i and ending at τ_i, having a height of β_i – this means that we assume the data transfer to take place in the time span $[\lambda_i, \tau_i]$ and consume a bandwidth of β_i[6]. This assumption is obviously correct for requests where the source bandwidth β_i is lower than the available link bandwidth B_{link} – we refer to these requests as *non-violating* requests – but does not hold for the *violating requests*, marked red in the figure.

In the initial state as depicted in figure 4.2, the fetches have been scheduled by ignoring B_{link} and simply calculating the required fetch time using β_i. While this approach works well for requests R_0, R_1 and R_4, requests R_2 and R_3 cannot be finished on time and cause a cache miss, violating condition (4.1) in figure 4.1 by imposing unnecessary response time ($\mathcal{RT}$).

[6]For the sake of simplicity, this bandwidth is equal for all requests in this example.

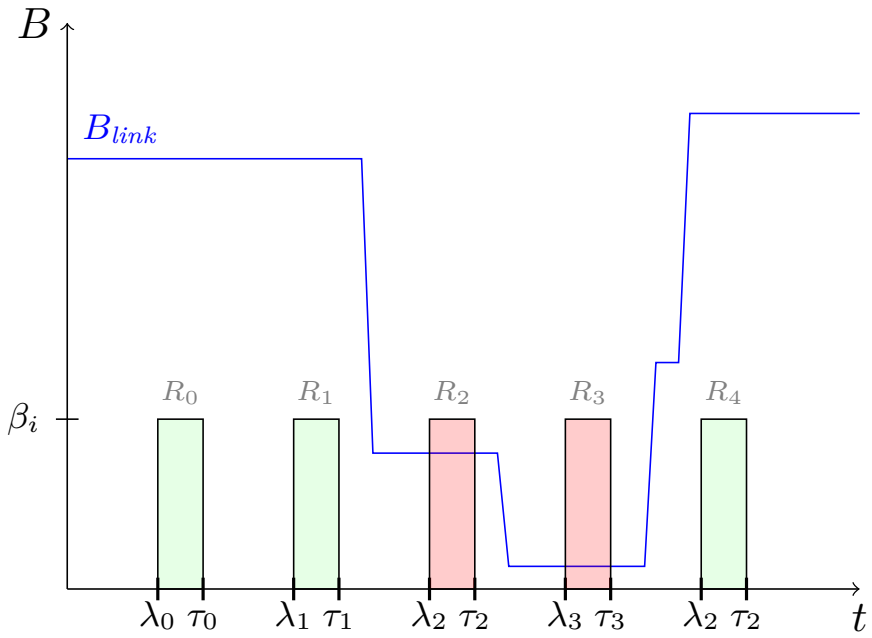


Figure 4.2: Original Situation: Requests R_2 and R_3 cannot be finished according to their deadlines τ_2 and τ_3, respectively

In order to resolve violating requests, we reschedule R_3 by moving λ_3 to an earlier time. Since we still aim to transfer the same amount of data, the area below the line representing B_{link} must be equal to the previously-red rectangle below β_i. To achieve this, we use integrals (this approach is further explained in section 4.5.2) and solve for their lower boundary.

We perform this procedure for the violating request with the latest deadline, R_3, since its rescheduling might affect preceding requests (as it does with R_2). We determine the optimal fetching time λ_2. See figure 4.3 for a graphical representation.

We have successfully solved the deadline issue for R_3. However, we also observe that the newly scheduled λ_3 is in conflict with the schedule for R_2, which is additionally also not scheduled correctly, since it would – even by itself – result in a deadline violation. Furthermore, the early fetching of R_3 would further increase fetching time for R_2, and this situation would lead to a subsequent delay for R_3. To resolve this, we again reschedule R_2, this time by adjusting both the upper and lower boundary of the area integral for R_2. The upper boundary (the new deadline) becomes λ_3, because fetching R_2 must be finished at the point in time where the scheduled fetching of R_3 is supposed to start. We again use integrals while keeping the area under B_{link} equal to the bytes to be transmitted, solve for the lower boundary and obtain a new fetch schedule for R_2, λ_2.

This schedule resolves the conflict for R_2 without influencing R_3. Since R_1 is also not restricted in any way by the new schedule, we have resolved all issues and found

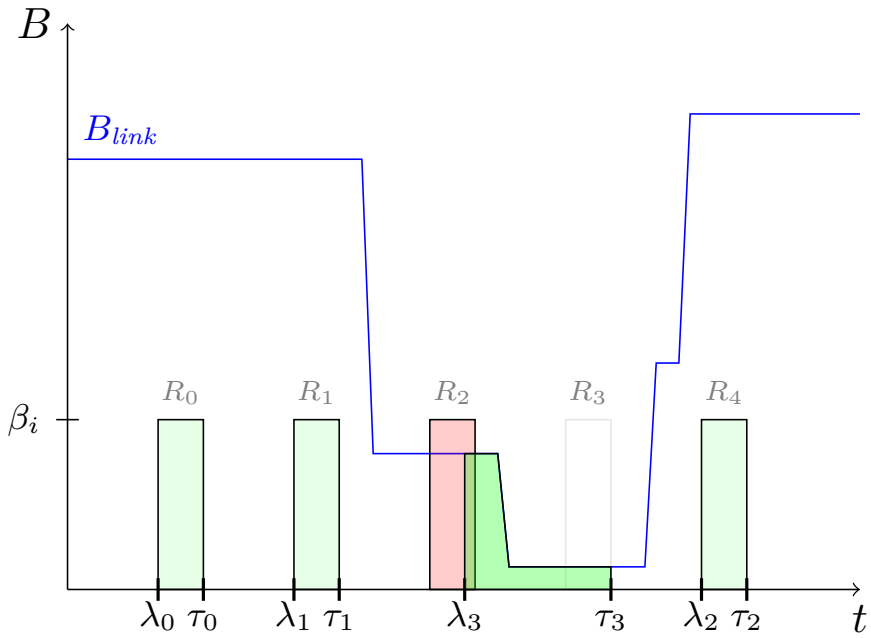


Figure 4.3: Resolved violation of R_3, R_2 is now in conflict with both its deadline and scheduled fetch of R_3

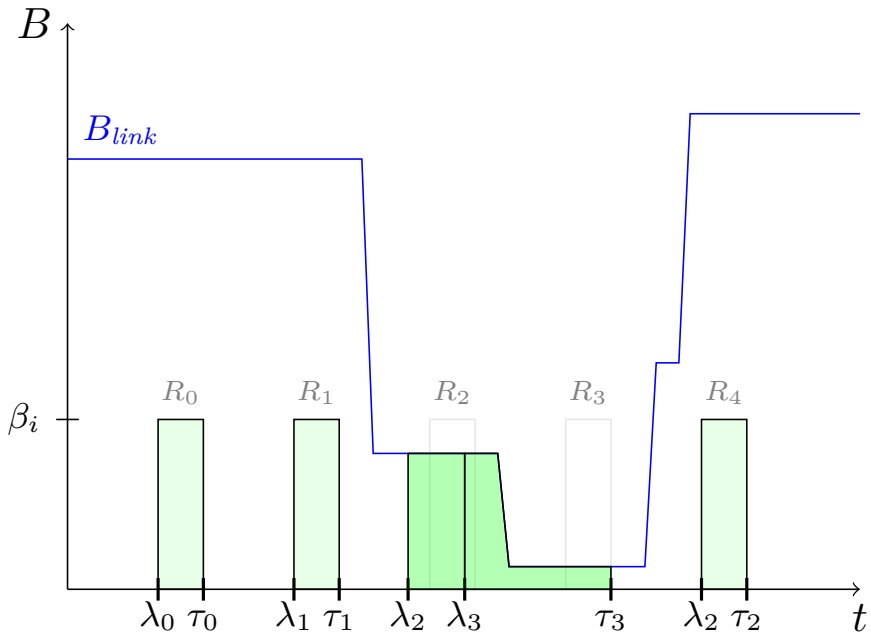


Figure 4.4: Issues resolved for all requests

a schedule for each request, which, under optimal conditions, will cause all requests to be fetched just in time for their required deadline. Figure 4.4 shows the resulting fetch schedule; it is clearly visible that all requests have their deadlines satisfied.

It should be noted that this is a simplification of a real-world scenario; in the latter, additional factors such as non-perfect prediction or sharing of the link with other applications play a role, which requires additional steps to be taken to compensate for the possible interferences. These measures are discussed in section 4.5.3.

4.5.2 Formal Concepts

For a basic understanding of our scheduling algorithm, we revisit the relationship between *bandwidth*, *time* and *data*, as described in section 3.3.3:

$$B = \frac{\mathrm{d}}{\mathrm{d}t} D \tag{4.6}$$

Obviously, this assumes continuous properties for B, t and D. While this might be true for t (time can indeed be treated as continuous), data (D) is certainly discrete because there is an integer unit of data that we use (bytes), and even though we often transmit parts of this unit (bits) one after each other (even though this is not always the case), those parts are themselves integer; one can split information into a smallest (atomic) unit. For our purposes, the atomic unit is the byte. From this it follows that B can also not be truly continuous.

However, we ignore the discrete nature of data for the purpose of this section and assume continuous properties in order to present the fundamental principle of our algorithm. We will later see that the idea can – and even requires to – be adapted for discrete values in an analogous way.

Expressing Data as an Integral of Bandwidth

We examine equation (4.6) and transform it by indefinite integration (antidifferentiation) of D, using t as the parameter variable. This yields the following equation:

$$D = \int B \, \mathrm{d}t \tag{4.7}$$

This is intuitive since the transmitted data is the sum of all transmittable data at each point in time. This infinitesimal statement is subject to three simplifying assumptions:

(1) The quantities D and B are continuous.
(2) The denoted bandwidth (B) will be used exclusively for transmission of the examined data chunk.
(3) The denoted bandwidth is always less than the bandwidth of the data source, i.e. the limiting factor is always B.

These assumptions are obviously merely an approximation of reality, and we will account for this approximation in the later parts of the discussion of the scheduling algorithm.

We see that the integral in equation (4.7) is indefinite, in other words, it has no boundaries (no interval $[a, b]$). The boundaries of this integral, however, represent the start and finish of the transmission of data in time.

We observe that the goal of our algorithm is for the transmission to and at τ (the deadline of the request), hence, we set the upper boundary of the integral to τ. The lower boundary is not yet defined – we denote it as a and will discuss it in the following steps.

Finding Integral Boundaries

When looking at equation (4.7) and taking into consideration our context, we observe that D is constant (the data required by a request does not change over time), but B is dependent on t. Furthermore, we are given a function of t, namely $B_{link}(t)$, which provides us with the (predicted) available bandwidth for any given time t:

$$B = B_{link}(t) \tag{4.8}$$

This allows us to use $B_{link}(t)$ as a substitution for B; yielding the following equation (note that the boundaries a and τ have also been inserted into the equation):

$$D = \int_a^\tau B_{link}(t)\,\mathrm{d}t \tag{4.9}$$

Following this equation, we see that the data, D, is provided with each request $R = (\tau, \beta, \delta)$, namely with the variable δ. We substitute δ – the data volume required by the request – for D in the equation:

$$D = \delta \tag{4.10}$$

Substituting D with δ leaves the lower bound a of the integral as the last unbound variable in the equation.

We now examine the resulting equation:

$$\delta = \int_a^\tau B_{link}(t)\,\mathrm{d}t \tag{4.11}$$

It is visible that all variables except for a are provided as input for the algorithm. The equation represents the transmission of δ bytes until the time τ, given that the bandwidth at each point in time t is specified by $B_{link}(t)$ – the transmission here starts at a. Given that a is the point in time at which the transmission starts, it becomes evident that this is the result we require our algorithm to provide. We therefore replace

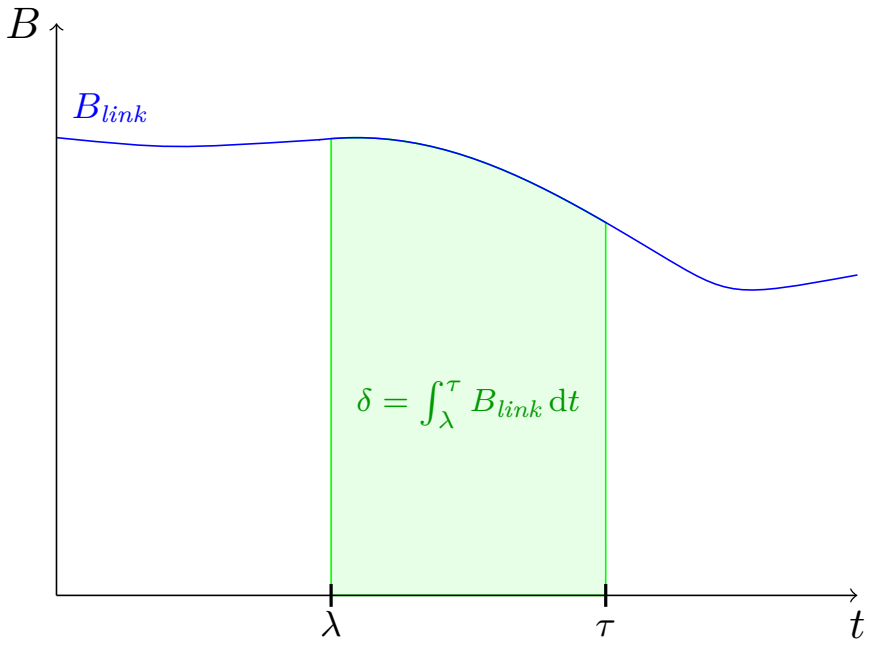


Figure 4.5: The notion of $\int B_{link}$ to represent data transmission

the symbol a for the lower boundary of the integral with λ, which denotes the scheduled fetching time for the request:

$$\delta = \int_{\lambda}^{\tau} B_{link}(t)\,\mathrm{d}t \tag{4.12}$$

A graphical representation is provided in figure 4.5; it is evident that the boundaries denote the start and end of transmission (λ and τ).

In order to solve for λ, one needs the antiderivative of B_{link}. We define this antiderivative as $D_{link}(t) = \int B_{link}(t)\,\mathrm{d}t$. This allows us to rewrite equation (4.12):

$$\begin{aligned} \delta &= \int_{\lambda}^{\tau} B_{link}(t)\,\mathrm{d}t \\ \delta &= D_{link}(\tau) - D_{link}(\lambda) \\ D_{link}(\lambda) &= D_{link}(\tau) - \delta \\ \lambda &= {D_{link}}^{-1}(D_{link}(\tau) - \delta) \end{aligned} \tag{4.13}$$

To continue solving for λ, we need to inspect D_{link} more closely, since to express λ from equation (4.13), the inverse function of D_{link} is required. We now encounter a boundary where no further solving is possible without deeper insight into the nature of D_{link}.

Translation into Discrete Quantities

We recall that $D_{link} = \int B_{link}(t)\,\mathrm{d}t$ under the aspect of finding the inverse function for D_{link} and establish that we need to inspect B_{link}. At this point, we need to relax an assumption we made (see section 4.5.2), namely the continuous nature of data (D) – assumption (1) –, and subsequently of bandwidth (B_{link}). We need to translate our continuous, infinitesimal model into a discrete context, and we do so by dividing the time t into slices. The bandwidth then is changed from $B = \frac{\mathrm{d}}{\mathrm{d}t}D$ to $B = \frac{\Delta D}{\Delta t}$ – see equations (3.2) and (4.6) – and denotes the transmission of a discrete amount of data (e.g. bytes) per discrete time span.

We then examine equations (4.7), (4.11) and (4.12) and their development, and translate them into a discrete context using analogous transformations, yielding the following equations:

$$\begin{aligned} D &= \sum_{t} B \\ \delta &\leq \sum_{t} B \\ \delta &\leq \sum_{t=\lambda}^{\tau} B_{link}(t) \end{aligned} \tag{4.14}$$

Note that we have replaced the equality relation ($=$) with a not-greater relation ($\leq$), since the required data volume might not exactly align with the available bandwidth of a slice. The sum of the transmittable bytes (the right-hand side of the equation) must not be lower than the required amount of bytes (the left-hand side of the equation). This equation is satisfied by any point in time before λ, since going back in time allows for more transmission of bytes (the right-hand side increases in magnitude). We implicitly only regard the *greatest* value for λ as the solution; we will make this assumption explicit in the following.

Figure 4.6 demonstrates equation (4.14) graphically; it is visible that δ is the actually required data (area printed in green), and $\sum_{\lambda}^{\tau} B_{link}$ is the data capacity of all time slices required for transmitting the actual data (area printed in fuchsia). Since we use discrete time slices, we have to round up, and reserve more than the necessary δ for transmission. It is also evident from the figure that the inequality in (4.14) holds ($\delta \leq \sum_{\lambda}^{\tau} B_{link}$), and that we implicitly assume the latest (greatest) possible value for λ (since any point in time before the λ in the figure would satisfy the inequality).

We continue by expanding in an integral-similar way, i.e. similar to $\int_a^b f = F(b) - F(a)$, by subtracting the sum using the lower boundary (λ) from the sum using the upper boundary (τ):

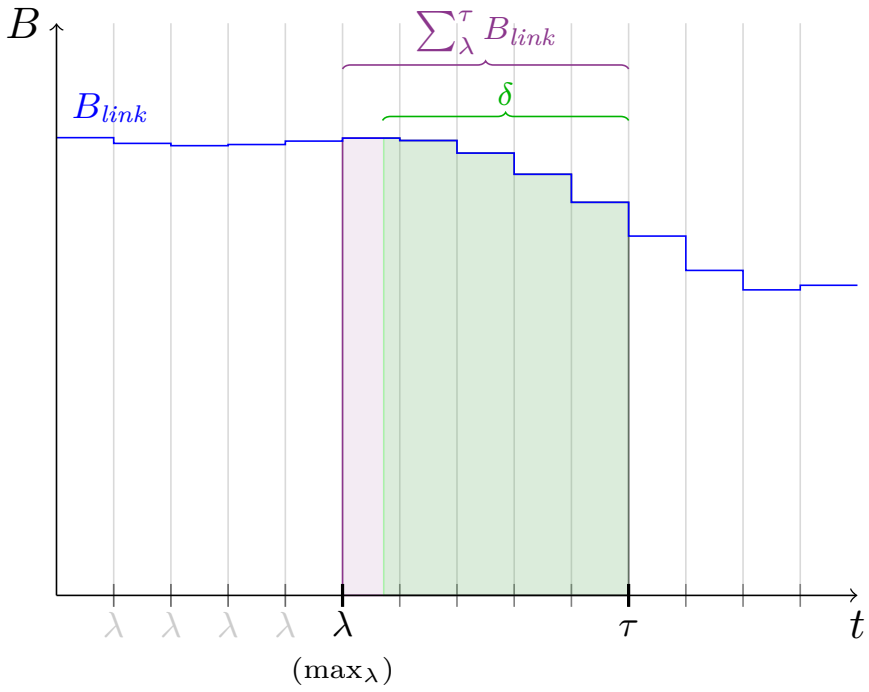


Figure 4.6: Data transmission in discrete quantities ($\sum B_{link}$)

$$\delta \leq \sum_{t=\lambda}^{\tau} B_{link}(t) \tag{4.14}$$

$$\delta \leq \sum_{t=t_0}^{\tau} B_{link}(t) - \sum_{t=t_0}^{\lambda} B_{link}(t)$$

$$\sum_{t=t_0}^{\lambda} B_{link}(t) \leq \sum_{t=t_0}^{\tau} B_{link}(t) - \delta \tag{4.15}$$

As mentioned before, we implicitly assume the largest λ as the desired outcome, which we now make explicit. This results in our final equation:

$$\lambda = \arg\max_{a} \sum_{t=t_0}^{a} B_{link}(t) \leq \underbrace{\sum_{t=t_0}^{\tau} B_{link}(t) - \delta}_{\text{Constant for } R} \tag{4.16}$$

Note that while this is a rather complex equation when using mathematical notation, its implementation is significantly simpler, as we will see in the next section. Furthermore, it is crucial to bear in mind that this is an approximation, which works under *optimal*

circumstances, especially under the assumption that bandwidth prediction is exact, and that the available bandwidth is available exclusively for fetching the single request – assumption (2) in section 4.5.2. We elaborate on the realistic evaluation of these assumptions in the next section.

4.5.3 Use Case Application of Formal Concept

We have shown how to calculate the scheduled fetching time λ for one request $R = (\tau, \beta, \delta)$ under certain assumptions. We now extend the use case towards the actual scenario our algorithm will work on by loosening constraints and introducing additional assumptions:

(1) The data source for R may have a lower bandwidth β than B_{link} – as opposed to assumption (3) in section 4.5.2 –, and this bandwidth is specified as β_R. There are several possible reasons for this case, including:
   - Servers providing data for R have a slow uplink, possibly because of high load, and are thus limited to β_R.
   - The data for R is produced on-demand, and the producer of the data has a certain maximum bandwidth β_R.
   - The nature of the data item for R prevents it from being generated at bandwidths higher than β_R.

(2) The prediction B_{link} is not perfect, i.e. it has an error margin (new assumption).
(3) The available bandwidth is shared amongst other processes – as opposed to assumption (2) in section 4.5.2.
(4) We calculate scheduled item fetching for multiple requests $R_1, R_2, \ldots, R_n$ instead of just one request R (new assumption).
(5) Our knowledge about the future, regarding the predicted link bandwidth B_{link} as well as the requests $\mathcal{R}$ only exist, or are only feasible until a certain point in time; the duration from the current time until this point in time is called *look-ahead time* (new assumption).

The following sections provide information about our proposal of handling those new constraints.

Data Source with Lower β than B_{link}

As described in assumption (1), the available bandwidth is not only dependent on the bandwidth of the network link, which was denoted as B_{link} so far, but also on the bandwidth of the data provider, β. This bandwidth might be significantly lower or higher than B_{link}. For data providers with a significantly higher bandwidth than the network link bandwidth ($\beta \gg B_{link}$), the described strategy of using B_{link} in calculations does not cause any issues; however, for network sources with similar or lower bandwidth than B_{link} ($\beta \lesssim B_{link}$), the algorithm must use the lower of the two for its calculations.

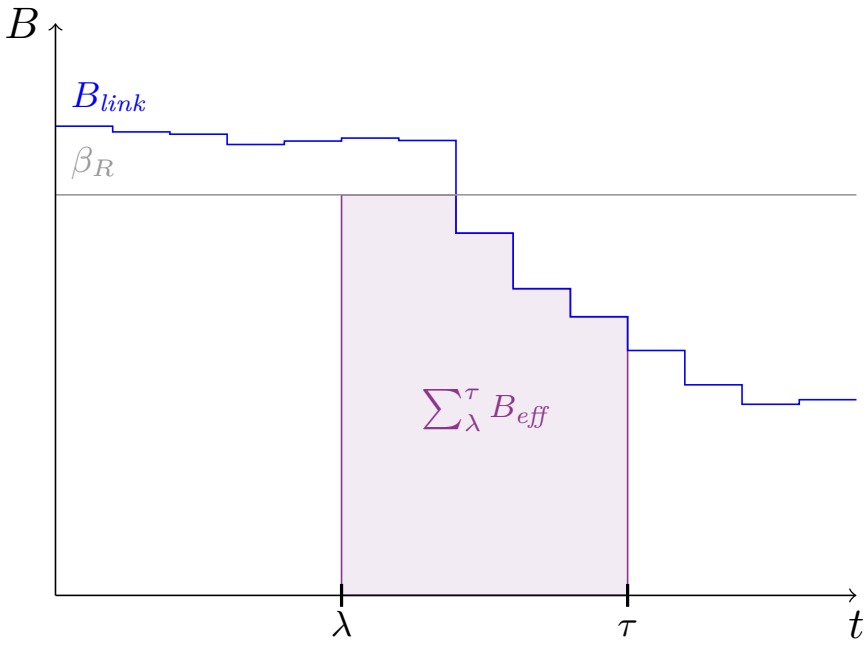


Figure 4.7: Limiting the effective bandwidth taken into account by the available bandwidth of the data producer β: $B_{eff} = \min(\beta, B_{link})$

For a complete resulting formula, see the next section, where the correction factor α is introduced and the resulting expression of B_{eff} – equation (4.17) – is shown.

Correction for B_{link}

In order to take an error of B_{link} – assumptions (2) and (3) – into account, we use a fairly simple approach; the value of B_{link} is simply multiplied with a constant correction factor α. The correction has to account for both prediction errors of B_{link} compared to the actual available network quality, as well as other applications consuming bandwidth in parallel to our algorithm. The former aspect has the possibility of being defined in a more precise way, for example by measuring the actual available bandwidth and using learning algorithms to correct future predictions, or by using heuristics such as the fact that shared Internet access links may be more overloaded at certain times of day. The latter aspect, sharing bandwidth with other applications, can also be detected by monitoring the device's running services, also in conjunction with machine learning. All of this, however, goes beyond the scope of our work, which is why we have settled on using a constant factor. For a graphical representation, see figure 4.8.

We therefore use the effective available bandwidth B_{eff} in our algorithm and define it as shown in equation (4.17). For a graphical representation, see figure 4.7.

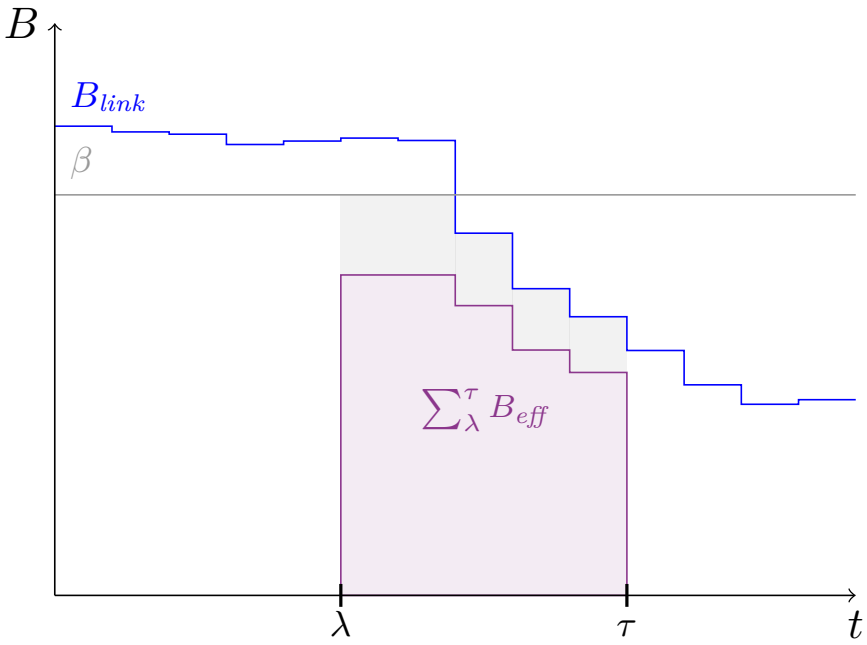


Figure 4.8: In addition to limiting the effective bandwidth by taking into account by the available bandwidth of the data producer β, an error margin is added, indicated by the gray area ($\alpha = 0.8$): $B_{\mathit{eff}} = \alpha \min(\beta, B_{\mathit{link}})$

$$B_{\mathit{eff}} = \alpha \min(\beta, B_{\mathit{link}}) \tag{4.17}$$

Too high values for α will generate fetches behind schedule, since the algorithm will use an overly optimistic prediction to base its calculations on. This violates condition (4.1) in figure 4.1. Too small values for α, on the other hand, will trigger fetches at a too early time, which leads to the results staying in cache for longer than necessary. This violates the condition for data ageing – condition (4.2) in figure 4.1.

During our simulation, we have used a factor of $\alpha = 0.90$, which results in 10 % of tolerable error in B_{link} – however, we were also in control of the influencing factors, which means that we were able to proactively set α to a well-performing value. See chapter 6 for an analysis of the impact of various factors on the results, including the value of α.

Handling Multiple Requests

Until now, we only examined a single request's fetch scheduling. In a real-world scenario, however, a high number of requests might require scheduling. We react to this by scheduling the request with the highest (latest) deadline τ – denoted as R_n – first. We then have to take into account that all ticks from this request's scheduled fetch start λ until the deadline τ are in use by R_n and cannot be used for any preceding requests. To

achieve this, we use a *marking* mechanism which stores the ticks used for the scheduled request as used. We then cease from using the marked ticks from scheduling of R_{n-1}, mark those we did use, and continue in the same way with R_{n-2} and so on.

The mechanism works as follows. A mark for a time slice is denoted as $\textsc{Mark}(t)$. Initially, all marks are set to zero, indicating that they can be used for fetching:

$$\forall t : \textsc{Mark}(t) = 0 \tag{4.18}$$

After a request R_i has been processed and the resulting fetching time λ_i is known, all ticks from λ_i to τ_i are marked as used:

$$\forall t \in [\lambda_i, \tau_i] : \textsc{Mark}(t) = 1 \tag{4.19}$$

For all subsequent requests, the mark is used in calculating the effective bandwidth B_{eff} as an additional factor:

$$B_{\mathit{eff}}(t) = (1 - \textsc{Mark}(t))\ \alpha \min(\beta, B_{\mathit{link}}(t)) \tag{4.20}$$

As we can see from this resulting equation for B_{eff}, a mark of 1 will result in a B_{eff} of zero for the given t, which will make the time slice unusable for all subsequent requests, which is the behaviour we require.

Look-Ahead Time

For the purpose of realism, we define a look-ahead time t_{horizon}, which describes how long into the future our algorithm can access its input. In other words, t_{horizon} specifies how much time before the actual fetch the middleware receives notice of the planned request.

Intuitively, if t_{horizon} is significantly higher than the average time needed to fetch a request's data, it is not determinable whether there is a look-ahead time at all. However, it has to be noted that an increased look-ahead time is not only less realistic (future requests might simply be unknown to the client software yet), it also increases computational power required to schedule fetches – power, which, in the context of mobile units, is regarded as valuable. Since recalculation must occur whenever a new request is added to the list of planned requests, conserving computational power can be seen as a soft requirement.

On the other hand, if t_{horizon} is too small, the algorithm might not take into account an upcoming data-intensive (and, as such, time-intensive) fetch or a prolonged time of degraded network quality and fail to prefetch data, even though it could have been possible to do so.

While the look-ahead time might seem like an inherent property of the algorithm itself, we separate those two matters and define the look-ahead time as a limitation of the environment in which the algorithm operates. The look-ahead time is a limitation

of the middleware and client code, and the algorithm is merely provided with a limited view of predicted future events.

We analyse the impact of t_{horizon} on the target variables in chapter 6.

4.6 Reference Implementation

We have implemented the algorithm in Java; to be precise, we implemented a simulation environment (discussed in chapter 5), of which the algorithm is a part (in a way that makes our algorithm easily replaceable with other implementations or algorithms). In this section, we provide a rough sketch of the algorithm's mode of operation.

4.6.1 Signature

The input parameters of our algorithm follow those discussed in section 4.2, and consist of the following elements.

- A collection of requests, with the following parameters per request:
  - A *deadline*, i.e. at which point in time the data is required.
  - The amount of data to be transferred.
  - The byte rate at which the data is supplied by the data source.
- A function ($B_{link} : t \to B$) supplying the algorithm with predictions for byte rates throughout the simulation time.

We have implemented the collection of requests as a simple Java `Collection` of objects of the `Request` class – a class which is a part of our simulation environment implementation. Following the service-oriented architecture of our simulation (a member of the simulation can obtain references to *services*), we have implemented B_{link} as a *rate prediction service*, which provides the algorithm with means of evaluating B_{link} for any t.

In return, the algorithm generates a scheduling function mapping each request to a point in time ($S : R \to t$), denoting when the fetching of a request should start in order for the deadline to be satisfied (given the prediction). The function is, in Java, implemented using a `Map` object. The map assigns each request with the scheduled fetching time.

As data types in our simulation, we use `long` values for points in time and `int` values for the data size. We thus derive the surrounding environment – or context – of the algorithm and the algorithm's signature itself and display it in listing 4.1 and listing 4.2, respectively.

Listing 4.1: Context (input) of a prefetch scheduling algorithm

```
public class Request {
   private final long deadline;
   private final int data;
   private final int availableByterate;

   /* ... getters ... */
}

public interface RatePredictionService {
   Integer predict(long tick);
}
```

Listing 4.2: Signature of a prefetch scheduling algorithm

```
public interface PrefetchAlgorithm {
   Map<Request, Long> schedule(Collection<Request> requests, RatePredictionService
     ratePredictionService);
}
```

4.6.2 Implementation

The implementation itself follows the idea discussed throughout section 4.5, using some optimisations or simplifications stemming from platform specifics. In this section, we show a version of our scheduling algorithm lacking any correction for B_{link}, as it is discussed in section 4.5.3.

The first part of the algorithm sorts the requests in descending order by deadline, which yields the last requests first. This is necessary to properly handle the situation depicted in figure 4.3, where the later request overlaps with its predecessor. This can only be resolved by handling the requests starting at the last one, without having to re-iterate over the requests and resolve overlapping requests using subsequent passes. Thus, we handle the latest requests first and are able to use a single-pass approach. We use the Java-inherent `Comparator` pattern together with `Collections.sort` for sorting the requests. The details of the sorting algorithm are not covered in our work.

We then define a variable, `previousStart`, which holds the value of the scheduled fetch start (λ) of the last processed request. It is required to resolve the overlapping discussed in the last paragraph. The variable is initialised with the maximum `long` value at the beginning, to avoid having to handle the first processed request in a special way.

```
long previousStart = Long.MAX_VALUE;
```

The implementation then iterates over the requests in a for-each loop and assigns to each request a scheduled starting time (λ, called `start` in the code). The starting time is calculated using the `previousStart` variable, the request deadline and the rate prediction service. After having calculated and stored the scheduled starting time, it updates the `previousStart` variable to `start` and continues with the next iteration.

```
HashMap<Request, Long> ret = new HashMap<>();
for (Request req : sortedByDeadline) {
    long start = getStart(previousStart, req, ratePredictionService);
    ret.put(req, start);
    previousStart = start;
}
return ret;

/* ... */

private long getStart(long busyUntil, Request req, RatePredictionService
    ratePredictionService);
```

We now inspect the implementation of `getStart`, the method responsible for scheduling a starting time for a particular request. As discussed before, there is a point in time until which the link is known to be busy – this is passed as a parameter to `getStart`, namely the first formal parameter, `busyUntil`. The parameter is assigned from the outer loop's variable `previousStart`.

The method starts by setting `tick`, an internal pointer, to 1 less than the request deadline, or the `busyUntil` parameter, whichever is earlier in time. This initial `tick` setting describes the latest point in time at which data can be received. Additionally, the method keeps track of how much data is to be fetched until the time `tick`, and it does so using the variable `data`.

```
long data = req.getData();
long tick = Math.min(busyUntil, req.getDeadline()) - 1;
```

We can see that the initial state of the method is that until `tick` (which points to the last point in time where data can be fetched before the request's deadline), `data` (which is the amount of data of the request) must be fetched.

After this initialisation, the method goes back tick by tick, assuming that the prediction service correctly indicates how much data can be transferred during that tick (again, this

variant of our algorithm performs no correction for B_{link}, as discussed in section 4.5.3). Given this prediction and the upper bound of data transfer, namely the request's data source byte rate, the algorithm determines the planned amount of data fetched from the source. This amount of data is subtracted from the still-to-be-transferred amount of data. The loop iterates until either the first tick (t_0) is reached (since no request can be fetched before that tick), or all necessary data has been scheduled for transfer. The resulting `tick` is either zero or the tick at which fetching should start for the deadline to be met.

```
while (data > 0 && tick >= 0) {
    Integer prediction = ratePredictionService.predict(tick);
    data -= Math.min(req.getAvailableByterate(), prediction);
    tick--;
}
return tick;
```

Note that this algorithm represents the iterative evaluation of the lower bound of the sum operator from equation (4.14). The complete implementation is shown in listing 4.3.

Listing 4.3: Implementation of our prefetch scheduling algorithm

```
public Map<Request, Long> schedule(Collection<Request> requests, RatePredictionService
    ratePredictionService) {
    HashMap<Request, Long> ret = new HashMap<>();

    List<Request> sortedByDeadline = sortByDeadline(requests);

    for (Request req : sortedByDeadline) {
        long start = getStart(previousStart, req, ratePredictionService);
        ret.put(req, start);
        previousStart = start;
    }

    return ret;
}

private long getStart(long busyUntil, Request req, RatePredictionService
    ratePredictionService) {
    long data = req.getData();
    long tick = Math.min(busyUntil, req.getDeadline()) - 1;

    while (data > 0 && tick >= 0) {
        Integer prediction = ratePredictionService.predict(tick);
        data -= Math.min(req.getAvailableByterate(), prediction);
        tick--;
    }

    return tick;
}
```

CHAPTER 5

Simulation Environment

In order to run the proposed prefetching algorithm and measure the resulting metrics, we have designed a simulation environment. It is capable of simulating scenarios where a mobile unit transitions through varying network quality while communicating with a server. This way, we can implement a prefetching algorithm, simulate fluctuating network quality and directly and reliably measure our key target variables.

This chapter describes the genesis and architecture of this simulation environment. The high-level structure is depicted in figure 5.1; the two main components are the *Client* and the *Server*, which are connected over a network link, which – in our case – is shaped (in other words, it imposes certain quality properties such as limited bandwidth and increased delay).

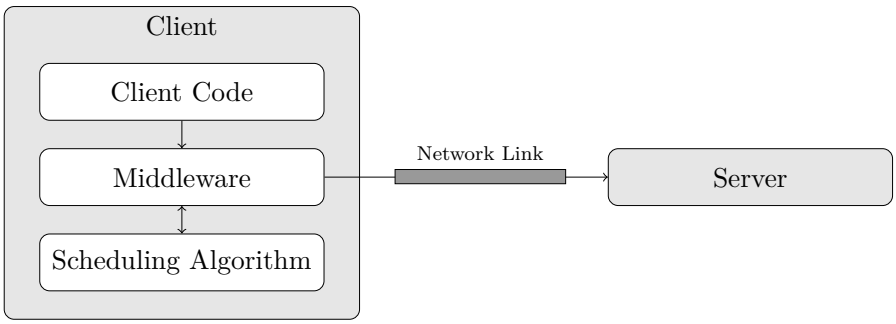


Figure 5.1: Conceptual view of a prefetching scenario

The client consists of the *Client Code*, the *Middleware* and the *Scheduling Algorithm*. The client code is providing the middleware with requests to which responses will be required at certain points (as described in section 4.2). The middleware then uses the scheduling algorithm to determine points in time at which the responses to those requests should be fetched, and does so independently of the client code. At a later point in time,

the client code actually requests data from the middleware. In the best-case scenario, the middleware was able to prefetch all responses and all requests to data made by the client software can be satisfied by the middleware with almost zero delay.

In this chapter, we discuss our implementation of a setup capable of simulating this scenario. We start by stating key requirements to the simulation environment in section 5.1, after which we briefly discuss our selection of tooling in section 5.2. We then present our implementation of a shaper for the network link in section 5.3 and a simulation environment in section 5.4. After having established the prerequisites, we present our top-level elements of the simulation in section 5.5, which represent the implementation of the components presented in figure 5.1. Finally, in order to analyse and discuss the results, we record performance metrics during our simulation, which we discuss in section 5.6.

5.1 Definition of Requirements

Key to our scenario is the simulation of fluctuating network quality. We identified the following core functionality conditions that must be met in order to approximate a prefetching scenario:

(1) Intercepting a communication stream between client and server software, transparently for the client and server application. Here, *transparently* refers to the fact that no code modification is necessary on neither client nor server code in order to employ network shaping[1]; the only permitted exception is that the client changes the endpoint of its connection from the original server to a (shaping) proxy server. Furthermore, we speculate that limiting our simulation to stream communication (for example TCP) is justifiable by assuming that specifics of Layer 4 protocols are not relevant for our research; any application employing non-stream communication by using UDP or any other Layer 4 protocol will have to employ some form of transmission control, which has a high likelihood of resulting in similar results for our purposes.
(2) Simulate real network conditions as close as possible, by employing limits to the traffic, namely limited bandwidth, increased delay and jitter in both attributes.
(3) The shaping parameters (bandwidth and delay) must support parametrisation and be dynamic, i.e. allow for change throughout the simulation.
(4) The simulation must be *reproducible*, meaning that even though pseudo-random events such as jitter in delay and bandwidth are allowed, the simulation itself must be purely deterministic and anyone with sufficient knowledge of the setup must be able to reproduce the exact outcome. The goal is to obtain the possibility of

[1] In our work, we define *network shaping* as the purposeful modification of a communication link's properties. This definition includes the creating of an artificial delay in the communication link or the limiting of its transmission speed.

documenting the setup of our experiments in enough detail to allow for reliable future reproduction of our results.

(5) The simulation should be as easy to set up and platform independent as possible.

5.2 Selection of Tooling

In search for a solution to realise this functionality in our simulation runs, we had to tackle two main problems, namely the shaping of traffic with regards to bandwidth and delay, as well as the simulation of a setup with multiple components and recording metrics heavily dependent on time.

In order to achieve the former functionality – shaping traffic – we evaluated several setups and describe their architecture as well as the foreseen advantages and disadvantages of such a solution in the following.

Network Simulation Tools There are several tools with focus on network simulation. ns-3 [Hen+08] is an open source network simulator with a rich set of features, which is developed in C++ with Python bindings, and offers a large amount of extendability. However, its rather complex architecture is a downside, since our simulation would only require a small subset of functionality. Reproducing simulations is enabled in ns-3 by using a fixed random number seed. ns-3 is supported on Linux, FreeBSD and Mac OS X platforms. Other alternatives include OPNET [Cha99] and NetSim [2], both of which are proprietary software. The setup using this software would include setting up a simulated network, implementing a client and server which use a simple protocol on top of, for example, TCP, and using the simulated network to impose quality setbacks to the communication. This process is rather time-consuming and prone to non-repeatability.

Network Shaping via Platform-Specific Mechanisms The Linux kernel, starting at version 2.6, provides the module netem, which exposes all the required shaping functionality. Its downsides are the dependency on the Linux kernel of at least version 2.6, and the lack of possibilities of reproducing the result. Similarly to the previous setup, the client and server would be implemented using a simple protocol on top of a network protocol such as TCP, and the platform-specific tool (in this case, netem) would be used to simulate low-quality network properties.

Creating a Testbed Using a testbed, preferably consisting of several virtual machines and employing network shaping tools poses a lot of effort in both setup and operation, which in turn vastly increases the effort necessary to ensure repeatability or even reproducibility. Our initial research has shown that issues arising from this approach, including the mechanisms required to reliably set up and tear down virtual machines while maintaining consistent state, as well as having to synchronise the machines' hardware clocks in order to reliably measure values, and the relatively

[2] See `http://www.boson.com/netsim-cisco-network-simulator`.

heavy resource usage when running such a simulation do not outweigh the fact that this setup is factually very close to a real use case.

Creating an Integrated Simulation Environment Designing and implementing a simulation environment for our purpose has the disadvantage of having to write and test additional code; however, it offers the ability to control the simulation process entirely down to the last detail. This includes fine-grained implementation specifics such as the exact way of processing data in the traffic shaper, or the way the network parameters (bandwidth and latency) are subject to jitter.

The closest match not involving the implementation of a custom tool, which was using a network simulator such as ns-3, was rather complex to set up and imposes a limitation to executing platforms. We therefore decided for the last solution described in the foregoing listing (creating an integrated simulation environment) and built a software framework in Java which suits our needs, enabling the execution of repeatable experiments on a wide variety of platforms, named scovilleJ. The detailed architecture and functionality of this framework is discussed in section 5.4.

Moreover, to provide traffic shaping functionality in scovilleJ simulations, we developed a library for this purpose, which can be used for both shaping of live traffic as well as shaping of traffic in a scovilleJ simulation. This tool is subject of discussion in the following section.

5.3 Traffic Shaping Using spiceJ

As described in section 5.1, our simulation requires a way of shaping traffic, in other words, imposing certain effects of degraded network quality in a simulation environment. For reasons described in section 5.2, we commenced developing such a tool in the process of developing a simulation environment for prefetching scenarios.

5.3.1 Traffic Shaper Design

The implemented traffic shaper – spiceJ[3] – is a generic tool capable of shaping traffic, not only in the context of our prefetch scenario. It covers two use cases:

(1) The interception of arbitrary TCP streams, where the traffic shaper works as a Layer 4 proxy accepting socket connections from hosts, initiating socket connections to the actual server and employing traffic shaping using this situation of being a proxy in the communication.
(2) The shaping of Java streams, where a (proxy) Stream object is calling methods of the actual Stream object, while employing traffic shaping. This can be useful if the

[3]Information, source code and documentation for spiceJ is available at `https://github.com/michael-borkowski/spiceJ` or `http://www.borkowski.at/spiceJ`. The version of spiceJ used in our analysis was 0.0.10.

algorithm under test is required to have the same source of simulation time as the traffic shaping module, which is necessary for condition (4) of the simulation in section 5.1.

Since scenario (1) can be realised using an implementation for scenario (2), we developed a stream implementation capable of repeatably employing bandwidth limits and add delay to transmissions using Java streams. We then developed a wrapper for this implementation which accepts and initiates socket connections as needed, and also simulates connection resets if the connection is simulated as stalled for an excessive amount of time. This wrapper binds the repeatable traffic shaping of scenario (2), to real time, which prohibits repeatability (preserving reproducibility); however, since we will not need this functionality for our simulation, this does not violate our precondition of repeatability, precondition (4) in section 5.1.

Our solution provides implementations of the Java stream classes, `InputStream` and `OutputStream`. The two core functions, limiting bandwidth and imposing a delay, are both implemented for input and output streams alike.

5.3.2 Traffic Shaper Implementation

All classes work by dividing the temporal dimension into slices, so-called *ticks*, where a tick may be triggered by an arbitrary event. Classes triggering tick events are called *tick sources*. Two implementation of tick sources are included in spiceJ: the classes `RealTimeTickSource` and `SimulationTickSource`. The former class provides a way of binding tick events to real time – in other words, it allows to fire a tick event periodically, with a certain interval. While the exact boundaries depend on the executing platform, we have found that events can be fired at intervals as low as 40 ns, which facilitates the generation of high-throughput rate-limited streams. The second implementation – `SimulationTickSource` – is more generic and can be used to fire tick events by calling a method, which is especially useful in the type of simulation we require for analysis of our algorithm's performance.

Rate Limiting

The rate limiting classes work by only allowing a certain amount of bytes to pass between two tick events. They are configured with two parameters, both of which can dynamically change during run time: *byte rate* and *prescale*. The byte rate specifies how many bytes are allowed to pass in between two tick events. The prescale allows to ignore each tick except for each n^{th} tick. In other words, a prescale of one means that each incoming tick event is considered for byte rate limitation, a prescale of two ignores every other tick event, a prescale of three ignores two out of three tick events, and so on. This mechanism is useful for creating byte streams with a low byte rate compared to the tick frequency. For example, given a tick source of 1 Hz, setting the byte rate to 1 B and the prescale to 4 results in a byte stream with a throughput of 0.25 $^{\mathrm{B}}/_{\mathrm{s}}$.

The input version of the rate limiting stream class works by implementing the stream's `read` methods in a way that only returns the amount of bytes which permitted by the settings. The stream can be configured to either block upon the first request of a byte where the tick's byte count has been exhausted or throw an exception. The rate-limited output stream simply blocks if more bytes have been passed for writing in the particular tick than allowed by the settings, and waits until the tick passes before continuing to write any further bytes. The Java output stream API does contain a non-blocking method for sending data and thus no other solution is possible.

Note that the input stream class can be used in a single-threaded, synchronous application while the output stream requires a separate thread to generate tick events, for the write method might block and wait for subsequent tick events.

Delay

The delay-imposing classes are configured with only one parameter: the imposed delay in ticks. The delay may be any non-negative real number and denotes how many ticks of delay have to be added to the byte stream. For input streams, this means that data is being read into a buffer upon each tick event, and `read` calls to the stream are working on that buffer, respecting the required delay. Our implementation keeps track of the ticks at which the buffered data actually becomes available to client code. The output version has a similar design; data is being written into an internal buffer with recorded time stamps, and at each tick event, parts of the buffer which may already be written (according to the configured delay) are sent to the underlying stream.

In contrast to the rate limiting streams, both the input and output version of delay-imposing streams can be used in single-threaded, synchronous applications.

Network Proxy

The core functionality of spiceJ consists of the presented stream implementations. Furthermore, as mentioned before, we have implemented a simple command-line program using these classes in order to provide a way of shaping actual live network traffic, a *proxy*. The proxy uses the `RealTimeTickSource` class in order to shape the traffic in a manner of rate limits in bytes per second and delays of seconds or fractions thereof.

Listing 5.1 shows an example of code using spiceJ. An input stream (`input`) and a tick source (`ticks`) are provided. A byte rate of $8\,\mathrm{B/tick}$ and a prescale of 2 are defined (this only serves the purpose of demonstration; the code might just as well use a byte rate of $4\,\mathrm{B/tick}$ and a prescale of 1 with a similar effect[4]). From the original input stream, a rate-limited version of the stream is created by using the `RateLimitInputStream` class, which is provided by spiceJ. The resulting input stream, `rateLimited` is then

[4]Obviously, while the overall duration of the read loop would remain the same, using $8\,\mathrm{B/tick}$ and 2 instead of $4\,\mathrm{B/tick}$ and 1 as parameters for byte rate and prescale, respectively, results in reads with twice as many ticks in between, but yielding twice as many bytes per read, resulting in an equal net byte rate.

Listing 5.1: Example of Code using the rate limiting functionality of spiceJ

```
// provided: an input stream and a tick source providing tick events
InputStream input = new ByteArrayInputStream(new byte[1000]);
TickSource ticks = createRunningTickSource();

int byteRate = 8;
int prescale = 2;

// byte rate of 8 and prescale of 2 results in an effective throughput of 4 bytes per tick

// result: a rate-limited version of the original input stream
RateLimitInputStream rateLimited = new RateLimitInputStream(input, ticks, byteRate, prescale
    );

byte[] result = new byte[1000];
int done = 0;

// the following loop takes 1000 / 4 = 250 ticks from tickSource to finish
while(done < result.length)
    done += rateLimited.read(result, done, result.length - done);
```

read in a read loop until the total amount of bytes has been stored. This read loop of 1000 bytes with a byte rate of 8 $^{B}/_{tick}$ and prescale of 2 (net byte rate of 4 $^{B}/_{tick}$) takes 250 ticks from the tick source to finish. If the rate limiting input stream is driven by a real-time tick source with an interval of 1 ms, the read loop takes 250 ms to finish.

In listing 5.2, the usage of spiceJ as a transparent TCP proxy is shown. All traffic towards a local port (in this example port 4001) is redirected towards a remote host (192.168.1.50 at port 4005) and traffic shaping in terms of limited bandwidth and increased delay is introduced. It is evident that the proxy implementation allows for asymmetric traffic shaping.

5.4 Repeatable Simulations Using scovilleJ

After having established a solution for traffic shaping (section 5.3), our commencing effort continued by designing a framework for simulating a distributed system, where a client and a server are communicating over a network link, the client repeatedly requests data from the server, and the server responds; all of this happens while relevant metrics are recorded for later analysis.

Listing 5.2: Example of using spiceJ to create a transparent TCP proxy

```
int localPort = 4001;

String remoteHost = "192.168.1.50";
int remotePort = 4005;

float rateSend = 48.2F;
float rateReceive = 128.0F;

float delaySend = 0.101F;
float delayReceive = 0.022F;

SocketProxy proxy = new SocketProxy(localPort, remoteHost, remotePort, rateSend, rateReceive
    , delayReceive, delaySend);
proxy.run();

// upon calling .run(), all TCP connections to the local port 4001 are
// forwarded to 192.168.1.50:4005, and underlie the following shaping:
//
// - upstream traffic (towards the remote host) is rate-limited to
//   48.2 bytes per second
// - downstream traffic (from the remote host) is rate-limited to
//   128.8 bytes per second
// - upstream traffic is subject to an additional delay of
//   101 milliseconds
// - downstream traffic is subject to an additional delay of
//   22 milliseconds
```

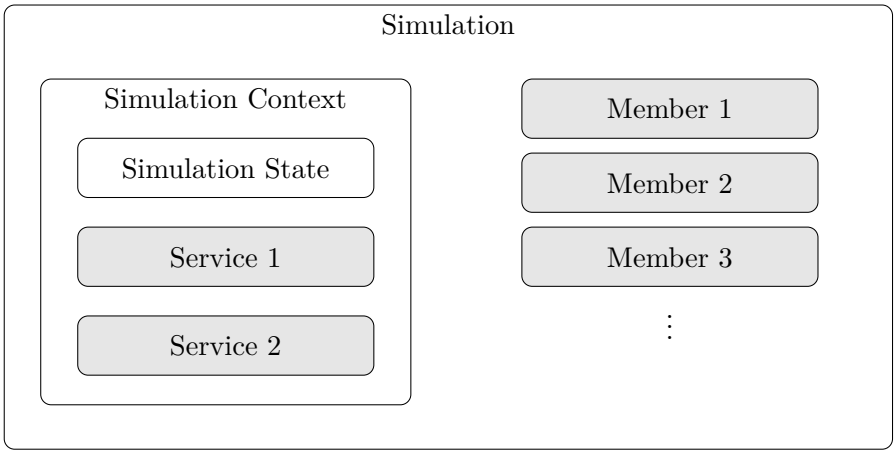


Figure 5.2: Conceptual view of a scovilleJ simulation

We developed a simulation and profiling framework and environment supporting this, scovilleJ.[5] Our framework enables several agent (*members*) to interact with each other while maintaining a consistent and repeatable notion of time. The latter is realised in a similar way as in spiceJ, namely using the notion of *ticks* to simulate temporal activity independently of real time. Using this strategy in combination with the deterministic nature of spiceJ, exactly repeatable results have been achieved.

Figure 5.2 depicts a simplified graphical representation of the architecture of a simulation in our implementation. A simulation contains an arbitrary number of members and services. The services, together with the current state of the simulation (out of which the main element is the number of the current tick and current phase, a concept we will discuss in the next section) are what is designated as *simulation context*. The simulation context is what each member is provided with for executing its business logic during each tick.

Members are each called once per tick[6] and may perform an arbitrary amount of business logic during such tick. The notion of what is *appropriate* or allowed per tick is heavily dependent on the exact, domain-specific kind of processes that are being simulated. Since for our purposes we only focus on the transmission time of data and disregard the computation time, we do not impose any limits on executed commands during one phase in our simulations. During each call, the members are provided with the *simulation context*, which, as described, contains the state of the simulation and serves as a way of obtaining references to services. A service is a special kind of member, which exposes an interface to other members to be called directly; it can be compared to a library. One example in our scenario is the communication service (which is defined

[5]Information, source code and documentation for scovilleJ is available at `https://github.com/michael-borkowski/scovilleJ` or `http://www.borkowski.at/scovilleJ`. The version of scovilleJ used in our analysis was 0.0.7.

[6]In fact, they are called once per phase, but for the sake of simplicity this is omitted in this preliminary explanation.

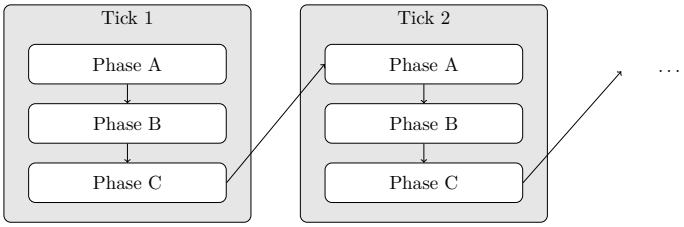


Figure 5.3: Sequence of ticks and phases

in the `CommunicationService` interface), which exposes a socket-like interface to members.

5.4.1 Simulation Framework Design

The core process is realised by the simulation *members* processing their business logic in so-called *tick handlers* (more precisely, as we will later see, *phase handlers*). As an initial constraint, operations within one tick, i.e. the actions (business logic) performed by the simulation members during the handling of a tick, must not interact with or affect other members. This leads to two desired effects:

(1) The order of calls to the handlers is not relevant for the outcome.
(2) Handlers for each member can also be called simultaneously (in multiple threads, or even on multiple machines).

This strict rule, however, greatly hinders simulation interaction between simulation members, since measures to avoid communicating inside one tick would have to be taken by communication services. For this reason, we divide the *ticks* into smaller time slices, so-called *phases*. While a tick is represented as a sequential number in our implementation, a phase is represented by a unique name.

Introducing *phases* for each tick, we change the restriction of interaction inside a tick to the restriction of interaction inside a phase. In other words, members are allowed to communicate with each other, as long as the sending and receiving of information is not taking place in the same phase. This way, the two properties (irrelevant ordering and the possibility of parallel execution) still hold for the simulation. Since members do not have the possibility to interact with each other directly (without regard to ways of bypassing the simulation process entirely, for example by using static classes or class members, sockets, files or other means of inter-process communication), it is the responsibility of services, especially those dedicated to providing communication between members, to ensure that this communication happens only across phase boundaries.

The sequence of ticks and phases as we have implemented it in scovilleJ is graphically represented in figures 5.3 and 5.4.

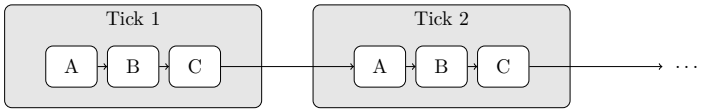


Figure 5.4: Chronology of ticks and phases

5.4.2 Simulation Framework Implementation

A simulation is structured by specifying a certain amount of *ticks* as the duration of simulation, and each tick is divided into an arbitrary number of *phases*. Furthermore, a set of *members* (participants) is specified for the simulation. Finally a set of *services* is defined – services are interfaces which may be acquired and called by simulation members.

Members are allowed to communicate with each other, as long as communication is forced to cross the phase boundary (see section 5.4.1 for an explanation). For example, a service may be defined to send messages between members by saving them to receive queues. An implementation following the limiting restriction would have buffers which are actually flushed at the final phase of each tick, to make the messages available to the receiving member at the beginning of the next tick. In fact, we use such a mechanism for our implementation of the communication service.

Simulation Procedure

The simulation is performed by the simulation supervisor. The supervisor sequentially executes each tick, where a tick is in turn split into the execution of each of its phases. Each member is notified (called) at the beginning of each phase, and the phase is finished when all members have finished their handling procedure. Only then, the next phase is started.

Service

As described in section 5.4.1, a service is an entity providing an interface for other members to call. Since such an entity contains business logic on its own that must be executed throughout the simulation, we derive *services* from *regular members*, making them a subclass of members. This also allows services to use other services in the same way that a regular member would.

We have implemented two major kinds of services. Firstly, the *communication service*, as already mentioned, allows members to employ socket-like communication with each other in a controlled manner, possible to shape. Secondly, we provide a *profiling service*, which serves as a central point of collecting recorded metric and allows for easy aggregation into statistical quantities.

In addition to those two services, we have some auxiliary service implementations, for example the *rate prediction service*, which allows algorithms that are bandwidth-aware to predict future available link qualities (thus evaluate values of B_{link}).

Profiling and Analysis

During the simulation, members may report values (measurement series) to the simulation framework. These values are aggregated and can be used by the calling process, which allows for analysis or output of simulation results.

5.4.3 Software Architecture

The implementation consists of a number of Java classes aiding the simulation of prefetching scenarios. No enterprise aspects such as persistence, clustering, session handling or similar are required: the simulation is configured with the required components, then run partly or completely, after which the recorded profiling data can be extracted by the using software.

Core Components

The core of scovilleJ consists of a few interfaces, to which implementations are given. The core interface is `Simulation`. A simulation can be queried for its phases (property `List<String> phases`), the tick it is currently in (`long currentTick`) as well as the total number of ticks (`long totalTicks`).[7] The user of `Simulation` can use `execute...` methods to partially or completely run the simulation – examples are `executeToEnd()` and `executeCurrentTick()`.

A `Simulation` is provided with certain child objects upon creation. These child objects are instances of the `SimulationMember` interface; in general, these objects are in some way influencing the simulation process. Members can consist of zero or more `PhaseHandler` implementations and zero or more `SimulationEvent` implementations. `PhaseHandler` objects are responsible for processing the tick phases of the simulation; in other words, they represent the actual behaviour or business logic of the simulation members. `PhaseHandler` classes are called for each phase of each tick.[8] Furthermore, the simulation member's `SimulationEvent` objects represent single-point-in-time events that happen at a certain tick. This scheduled tick is known a priori, before the simulation starts, and is as such part of the simulation configuration. Simulation events cannot be dynamically created throughout the simulation. Since

[7] Note that the actual implementation allows for a fine-grained control of the tick processing mechanism: the `Simulation` interface differentiates between the state of being about to process a certain tick, and having processed the tick. This description is a simplification to ease the understanding of the class structure.

[8] For performance reasons, a phase-subscription mechanism has been implemented, allowing phase handlers to only be called for certain phases – this reduces overhead.

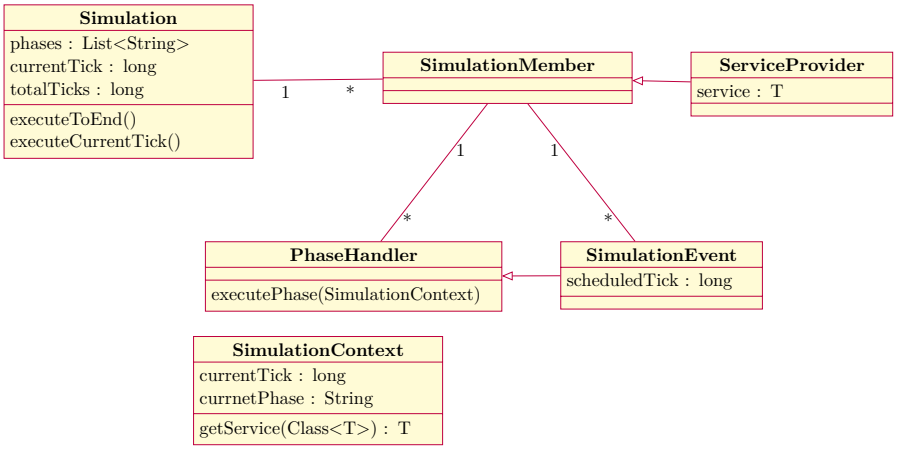


Figure 5.5: Conceptual UML view of scovilleJ's core interfaces

the `SimulationEvent` interface extends `PhaseHandler`, it is clear that events are a special form of phase handlers, namely handlers that will only be called for the scheduled tick.

A `SimulationMember` implementation can use any combination of `PhaseHandler` and `SimulationEvent` objects to achieve its business logic goal.

Furthermore, to allow for actual measurable (inter-)action of simulation members, the notion of *services* has been introduced; the simulation is configured with a certain set of services, which are identified by a Java class or interface. A service is provided by a `ServiceProvider` object and can be requested by a phase handler from the `SimulationContext` it is given during its execution.

Built-In Services

As mentioned in section 5.4.2, one of the main services we have implemented is the communication service, which provides a socket-like interface for members to communicate with each other. One member can start a listener with a given name, and any other member can open a connection to this name. The listening member can decide whether multiple clients are accepted by calling the `accept()` method once or multiple times. We use this communication in connection with the previously-described traffic shaper `spiceJ` (see section 5.3) component to simulate the communication between a mobile unit and its server.

The described profiling service is simply implemented as a store of measurements with an addition of aggregation functionality yielding statistical results such as mean, average, minimum and maximum values and so on.

To summarise the implementation overview of scovilleJ, figure 5.5 provides a view of the UML structure, showing the most important classes.[9]

5.5 Simulation of Prefetching Scenarios

Prefetching is simulated on top of scovilleJ. No real-time components of spiceJ are used, which makes the entire simulation synchronous. Currently, we have implemented a single *server* component and one *client* component as simulation members. However, the scenario is easily extendable to multiple clients[10] and multiple servers. We restricted our simulation to this point-to-point simulation since the results can easily be applied to multiple users, provided that each user is provided with an equivalent communication channel with the server. The source code and documentation for the prefetch simulation framework is available at `https://github.com/michael-borkowski/prefetch-simulation` or `http://www.borkowski.at/prefetch-simulation`. The version of the project `prefetch-simulation` used in our analysis was 0.0.2.

[9] Note that this is a simplified view; simplifications include the notation of attributes instead of getter methods, or the omission of the <<interface>> stereotype as well as some less relevant methods.

[10] In fact, the server component is already capable of handling multiple, simultaneous clients.

5.5.1 Prefetch Simulation Setup

The simulation setup is done in two steps. The first set of parameters to a simulation is called a *configuration*. This configuration consists of a fixed-size set of parameters and can be serialised (i.e. stored for later recall). A configuration can, in a subsequent step, be converted to a *genesis* – a genesis describes in detail the time line of a simulation. While the configuration contains dimensions which require randomisation, a genesis is a complete, deterministic specification of a simulation run. Figure 5.6 demonstrates the process graphically.

The *genesis* of a simulation can also be serialised, albeit requiring more (asymptotically increasing) storage space than the fixed-size *configuration*, because it lists all simulated events and their corresponding ticks explicitly. The gain of such an explicit description is, as already mentioned, the lack of non-determinism.

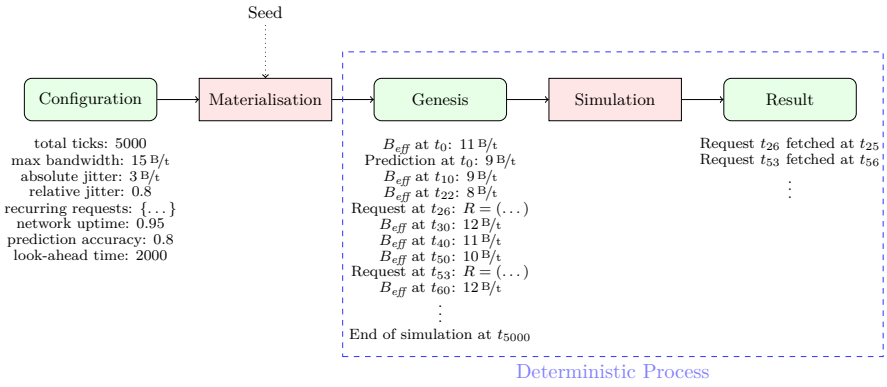


Figure 5.6: Simplified view of the simulation setup and execution process

As an example, a configuration contains the parameter *relative jitter*, which we will discuss in detail in the next section. Relative jitter specifies by how much of the available byte rate the actual simulated available byte rate varies throughout time. Given an exemplary byte rate of 10 $^{\text{bytes}}/_{\text{tick}}$, a jitter parameter of 0 would imply that the byte rate is completely stable (10 $^{\text{bytes}}/_{\text{tick}}$ at each point in the simulation). A jitter of 0.4 causes the byte rate throughout the simulation time to vary between 6 $^{\text{bytes}}/_{\text{tick}}$ and 14 $^{\text{bytes}}/_{\text{tick}}$ (since $0.4 * 10 = 4$) instead of being fixed at 10 $^{\text{bytes}}/_{\text{tick}}$. The materialised, deterministic *genesis* variant of such a scenario would be the following sequence of ticks:

Table 5.1: Example: Materialised byte rate

Tick	Byte Rate
130	6
140	13
150	5
160	14
170	11
180	13
190	8
200	7
220	14
230	7
240	14
250	12

It is clearly visible that the element of non-determinism has been eliminated by explicitly stating the values of variables underlying non-deterministic influence. We call this process materialisation.

Simulation Configuration

The parameters constituting the *configuration* of a prefetch simulation are *top-level* in the sense that they describe human-readable and human-writable aspects such as the number of services running on a mobile unit, the fluctuation properties of the network link, the overall maximum available byte rate or the total number of ticks.

The time line of the simulation is split into so-called *slots* of varying length. Each slot has its own net available byte rate, where the actually simulated byte rate is enriched by an amount of jitter. Slots have a certain probability of having a byte rate of zero, in which case the jitter has no practical effect (the byte rate remains zero). Figure 5.7 shows a graphical representation of the slots with their designated byte rates.

The jitter is calculated for tick intervals smaller than the *slot* length; for each of these ticks, the actual value for the available bandwidth is calculated using the following formula:

$$B_{\mathit{eff}} = a + b\, B_{\mathit{link}} \tag{5.1}$$

In this context, a denotes the *absolute jitter amplitude* and b is the *relative jitter coefficient*. Figure 5.8 shows a graphical representation of slots including the described jitter.

Additionally, for each slot, the prediction function (a function which maps a tick to a predicted or estimated byte rate) represents the non-jitter-subject version of the available byte rate, where two kinds of errors are introduced in the simulation for realism: firstly, an amplitude error is introduced, so the prediction function can randomly over- or under-estimate the actual bandwidth. Furthermore, a temporal error is added, which

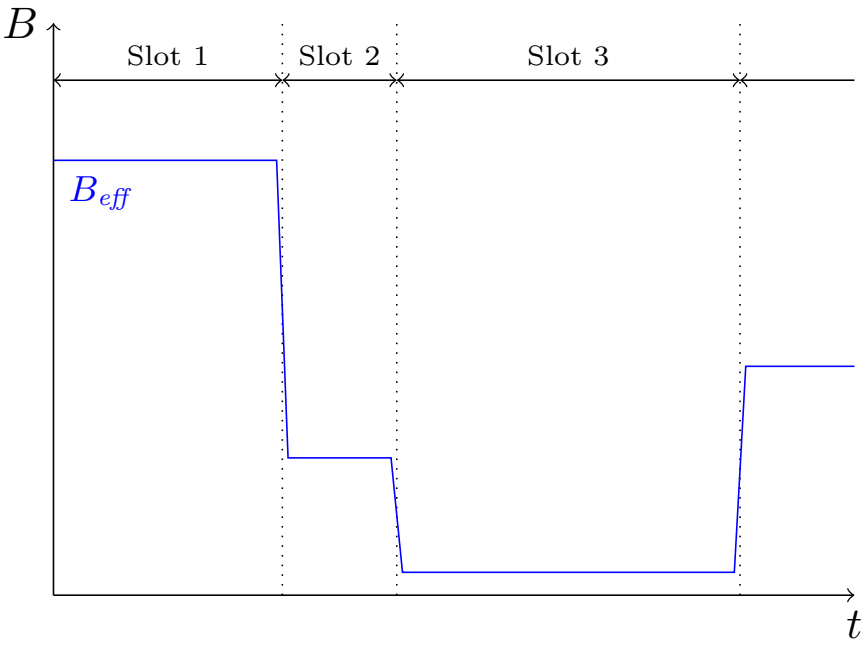


Figure 5.7: Slots with designated byte rates

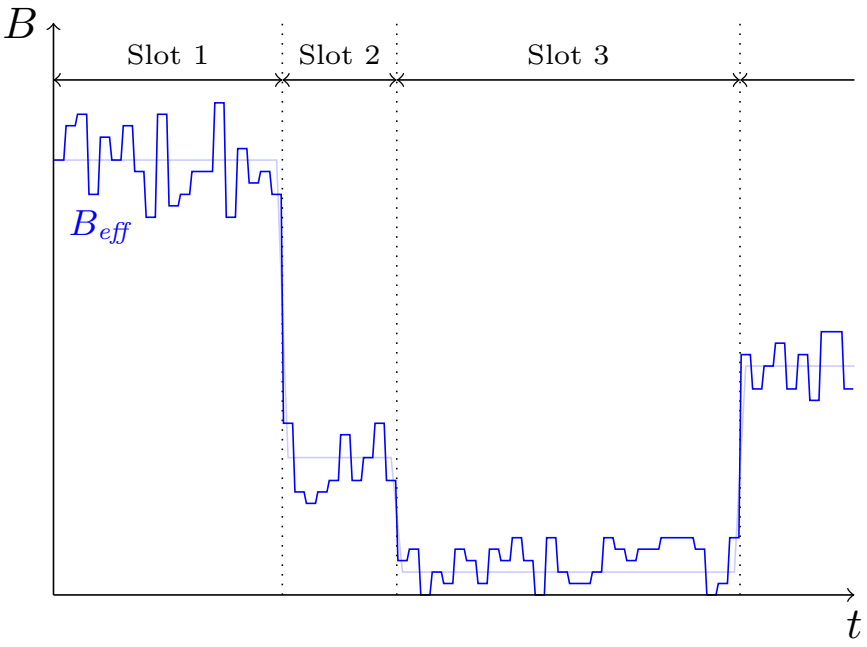


Figure 5.8: Slots with introduced byte rate jitter

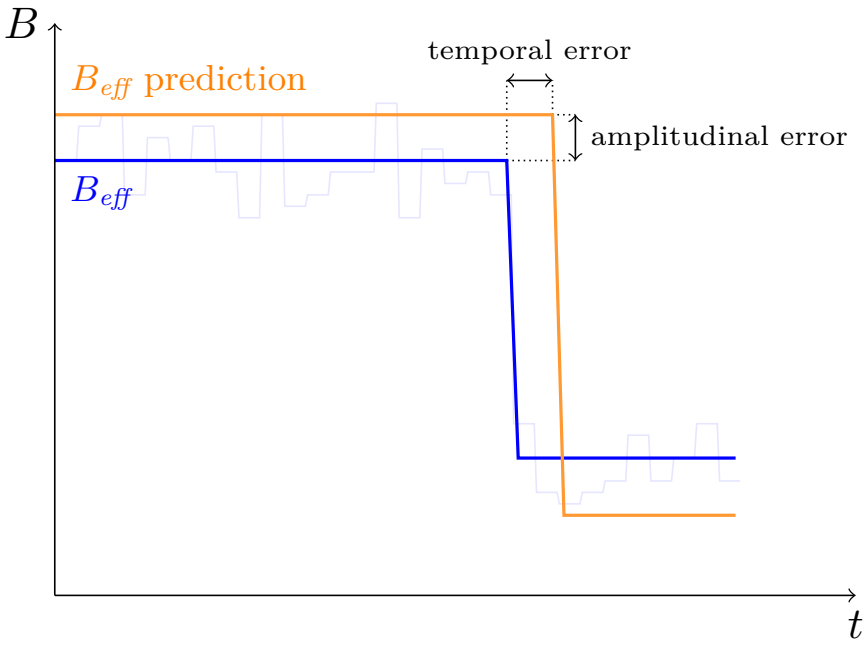


Figure 5.9: Types of error introduced to byte rate prediction

shifts the time stamps of the changes in the prediction function randomly to either direction. This means that the changes in bandwidth can happen before or after the corresponding change in the prediction function. Figure 5.9 depicts these two kinds of errors introduced.

Furthermore, the simulation configuration contains a list of *recurring* and a list of *intermittent* requests. The former describes a series of requests by specifying the interval in ticks of recurring events, the start and end times in ticks, as well as the data size and byte rate, which are already well-known parameters for requests, described in chapter 4. The latter describes a single request in time by specifying the request parameters of deadline, data size and byte rate. Finally the algorithm and look-ahead time used for scheduling request fetches is specified.

Summarising, the complete list of parameters of a simulation configuration is as follows. *Distribution* as a part of this configuration means that a configuration defines how a variable is randomly distributed. Available distributions are: *exact* (all draws from the random variable have the same value), *uniform* (values are evenly distributed between a minimum and a maximum value) and *normal* (values follow a gaussian distribution curve around a given mean with a given standard deviation).

- The total duration in the situation, in ticks.
- The distribution of byte rates per slot.
- The distribution of lengths of a *slot* of available byte rate, in ticks.

- Network uptime – i.e. the probability of a slot showing a non-zero byte rate, expressed as a ratio.
- Relative jitter, as a ratio, and absolute jitter values, in ticks.
- Accuracy of prediction (amount of temporal and amplitudinal error in prediction):
  - Accuracy in time, as a distribution of relative (as a ratio) and absolute (in ticks) measurements
  - Accuracy in byte rate, as a distribution of relative (as a ratio) and absolute (in ticks) measurements
- A list of *recurring requests*.
- A list of *intermittent requests*.
- The type (Java class) of scheduling algorithm to use.
- The look-ahead time t_{horizon} of the fetch scheduling.

In our implementation, we use a text-based format for storing this information in a file, in order to allow for persisting configuration used for our experiments.

Simulation Genesis

As shown in the last section, the simulation *configuration* must be converted to a simulation *genesis* in order to be run. This conversion (or materialisation) removes any dependency on non-deterministic variables.

In our initial implementation, we used the Java implementation of a Random Number Generator (RNG), which uses a *linear congruential generator*[11]. However, since we aim to reduce the dependency on the underlying Java run time[12], we stripped the Java RNG of all features not required by our code. Hence, we use a custom implementation of a random number generator, which is derived from the Java reference implementation by Oracle, which in turn is essentially an implementation of Knuth's Linear Congruential Method with the parameter set $m = \texttt{Long.MAX\_VALUE}, a = (\texttt{5DEECE66D})_{16}, c = 11$ and the seed X_0. This way, the only further variable to precisely describe the simulation and make the outcome repeatable is the seed of the RNG used to materialise the genesis.

The resulting genesis is of variable size (growing with the total number of ticks, the number of recurring and intermittent requests as well as the frequency of change in byte rate, which is derived from the duration of byte rate slots) and contains a list of entries, most of which are events happening during the processing of a specified tick.

[11] The official Java documentation refers to Section 3.2.1 of [Knu97], which in detail explains the Linear Congruential Method – Knuth discusses several variations and possibilities of using LCG to generate numbers of reasonably long periods; see the following paragraphs for the resulting implementation of RNG used in our experiments.

[12] Even though the Java documentation requires the use of LCG and targets the absolute portability of Java code, we failed to locate the explicit specification meeting this requirement and describing the exact type of LCG (the parameters) to use and decided to further strengthen the repeatability of results by essentially forking the reference implementation of Java's `Random` class into our code base.

The following list shows all the information contained in a simulation genesis:

- The class name of the algorithm used for scheduling fetches.
- The look-ahead time of the fetch scheduling.
- For each tick where the actual byte rate changes:
  - The tick number at which the change happens.
  - The byte rate valid from this point in time until the next byte rate change.
- For each tick where the predicted byte rate changes:
  - The tick number at which the change happens.
  - The byte rate valid from this point in time until the next predicted byte rate change.
- For each request (regardless of whether the request is part of a recurring request series or an intermittent one):
  - The tick number designating the deadline of this request.
  - The amount of data required for fetching.
  - The byte rate at which this request is provided by the data source.

Similarly to the simulation configuration, we designed a text-based format for storing this information in a file. This allows us to store the materialised version of simulations.

5.6 Profiling Metrics

In the decision process of which variables we measure during our simulations (in other words: what the *result* of the simulation is), we followed the target variables discussed in section 4.3. We revisit these variables and note that out of the three discussed values, $\mathcal{RT}$, $\mathcal{DA}$ and $\mathcal{DV}$, only the former two variables are truly relevant; $\mathcal{DV}$ will in our scenarios always be minimal since we fetch each request at most one time (see also the discussion in section 4.3.3). However, since we will also simulate scenarios where our algorithm is replaced by different substitutes, we also record $\mathcal{DV}$ as a result metric.

In addition to the three variables described in the foregoing paragraph, we are interested in the *hit count* and *miss count* of the cache, to learn how many requests could be completely prefetched before their deadlines. To be able to compare results across different absolute numbers of requests, we add the *hit ratio* to the list of metrics, and define it as the ratio between the *hit count* and the total number of requests. Hence, we identified the following metrics in need for profiling.

(1) The perceived response time, $\mathcal{RT}$, which is the time span between the request's deadline as given by the client software, and the point in time at which the data becomes available for the client software's disposition. In a best-case scenario, the prefetching algorithm correctly schedules the fetches and all requests can be

prefetched on time; in this case, $\mathcal{RT} = 0$ holds for all requests. We record the minimum, maximum, average and median of $\mathcal{RT}$ for all requests.

(2) The data age, $\mathcal{DA}$, which is the time span between the data being fetched from the server and it becoming available to the client software. As discussed in section 4.3.2, this is relevant as a metric of the *freshness* of data, since some services require the data not to exceed a certain age. We record the minimum, maximum, average and median of $\mathcal{DA}$ for all requests.

(3) The amount of transferred data volume, $\mathcal{DV}$, as discussed in section 4.3.3, since this metric implies both energy usage and possibly induced costs for the user. Again, we record the minimum, maximum, average and median of $\mathcal{DV}$ for all requests.

(4) The hit count, defined as the ratio of the cache hits (the number of requests served entirely from the cache, after being prefetched) and the total number of requests.

CHAPTER 6

Testing and Analysis

In this chapter, we evaluate the algorithm described in chapter 4 using the simulation environment discussed in chapter 5 and evaluate its results. We start by defining three strategies which we will use to simulate and evaluate – this definition can be found in section 6.1. Initially, in section 6.2, we present an exemplary simulation, discuss each of its configuration's aspects (section 6.2.1) and run the simulation for each of the three different strategies. We then thoroughly interpret the results and discuss their implications (sections 6.2.2, 6.2.3 and 6.2.4) and summarise the strategies' performance in section 6.2.5.

After having an in-detail presentation and discussion of the performance on the exemplary simulation, we commence by stating our hypothesis in section 6.3. To verify this hypothesis, we perform experiments in which we define dependent and independent variables, run simulations and record the behaviour of our target variables. The experiments are described in section 6.4 and their results shown in section 6.5. A discussion and summary of the results can be found in section 6.6.

A detailed, step-by-step description of how to repeat our experiments and reproduce the results can be found appendix A.

6.1 Pre-Selection of Strategies

We use three strategies for comparison of performance:

(Strategy A) The *no prefetching* strategy, which means that data is fetched starting the time of their request. We use the deadline for this time, which corresponds to the classical situation of an application issuing a request at the point in time where data is required. The class implementing this scheduling strategy is `NullAlgorithm`.

(Strategy B) The strategy of scheduling fetches ahead of their deadline, however, without regarding the predicted network quality – also called the *prediction-ignoring prefetching* strategy. This means that for each request, the scheduled fetching time is calculated only by the amount of data required and the *a priori* known available byte rate of the data source, but not the available link byte rate.[1] The base formula used for this strategy is $\lambda = \tau - \frac{\delta}{\beta}$ – in other words, the required amount of data (δ) divided by the byte rate of the data source (β) is subtracted from the deadline (τ). In addition to this calculation, a proportional error margin is added to β, as described in section 4.5.3. Hence, the resulting formula is $\lambda = \tau - \frac{\delta}{\alpha\beta}$. Moreover, the implementation prevents over-scheduling by selecting a lower (earlier) τ if a λ of a later request would overlap with the current request (this mechanism is described in greater detail in section 4.5.1, especially in figure 4.3 and figure 4.4). This strategy is implemented in the Java class `IgnoreRatePredictionAlgorithm`.

(Strategy C) The presented algorithm (section 4.5) with all its additions described in section 4.5.3: data sources with an available byte rate β lower than B_{link}, the correction margin for B_{link} (α; also used in strategy B), as well as the ability to handle multiple requests (also used in strategy B). Our algorithm is implemented in the class `RespectRatePredictionAlgorithm`.

6.2 Exemplary Simulation

We inspect in detail one specific simulation instance, with a duration of 3600 $\mathcal{T}$ and five planned requests. The available link byte rate ranges from around 1000 $\mathrm{B}/\mathcal{T}$ (around the first few hundreds of ticks of the simulation) up to around 4300 $\mathrm{B}/\mathcal{T}$ (1400 $\mathcal{T}$ throughout 2000 $\mathcal{T}$). There is a slot of a link outage (no transferable data) around 1100 $\mathcal{T}$..1300 $\mathcal{T}$, between the first and the second planned request. Details on the repeating of this simulation and reproduce its results can be found in section A.2.

Our hypothesis for this section is that given this exemplary simulation configuration, strategy A will perform *the worst* and strategy C *the best*, with respect to the $\mathcal{RT}$ target variable, as specified in section 4.3. We formulate this hypothesis and analyse in-depth the behaviour of the three strategies. We derive this hypothesis from the overall purpose of our proposed algorithm (strategy C) to perform better than a no-prefetching approach (strategy A) in means of the aforementioned target variables, and use strategy B as an intermediate measurement step, to determine whether knowledge about the link quality increases the performance.

[1] This can also be seen as a case of the scheduling algorithm being provided with infinity as the predicted byte rate.

6.2.1 Discussion of the Simulation's Configuration

The base configuration of the simulation is shown in listing 6.1. Line 1 sets the seed to the value used throughout this analysis (26031991001). Line 3 sets the number of ticks to 3600, line 4 determines the byte rate for a slot to be uniformly distributed between $1000\,\mathrm{B/ⅎ}$ and $5000\,\mathrm{B/ⅎ}$. In line 5, we see that the slot length has been fixed to $180\,ⅎ$ and in line 6 the link availability is set to be 0.9, which means that in average, 10 % of the slots will have a byte rate of zero (in fact, the seventh slot and the last slot are such slots, as it can be seen in the figure). The jitter is set to zero (disabled) in lines 7 and 8, and lines 9 through 12 set the prediction to be very accurate, with only a minimal amount of error (a relative amplitude error normally distributed around 0 % with a standard deviation of 1 %p, and an absolute time error, also normally distributed, with a mean of $0\,ⅎ$ and a standard deviation of $10\,ⅎ$). The look-ahead value (t_{horizon}) is set to $3600\,ⅎ$, allowing the algorithm to completely calculate the entire simulation from the beginning. In line 15, the α value of 0.9 determines that a margin of 10 % is applied to the predicted link byte rate to compensate for a prediction error. Finally, lines 17 and 18 define planned requests, line 17 representing a request series with an interval uniformly distributed between $400\,ⅎ$ and $600\,ⅎ$, a size normally distributed with a mean of $500\,\mathrm{kB/ⅎ}$ and a standard deviation of $60\,\mathrm{kB/ⅎ}$; line 18 defines a single request at $3400\,ⅎ$ with the data source having a byte rate of $3600\,\mathrm{B/ⅎ}$ and requiring a transfer volume of 480 kB.

Note that this configuration lacks the specification of a scheduling algorithm; we will use a different algorithm for each of the strategies (A, B and C) simulated, and a corresponding line is implicitly added to the respective simulation configuration.

We inspect the graphical representation of this configuration[2] in figure 6.1. The figure shows the five scheduled requests (named `000` through `004`). The right border of the grey boxes represents the deadline, the height designated the available bandwidth, and the array is the amount of data for this request; this results in the left border of the rectangle being positioned at the point in time where fetching should start, neglecting any limited link quality. For example, request `000` has a deadline at $1000\,ⅎ$, requires 560520 B of data and its data source is defined to provide a byte rate of $3696\,\mathrm{B/ⅎ}$. By calculating $\frac{560520}{3696}$, we obtain a duration of around $151.7\,ⅎ$, so the request's box is positioned with its left border starting at tick $1000 - 151 = 849\,ⅎ$. Using this representation, we can easily judge the amount and speed of data to be transferred as well as its deadline.

As for the link quality, we can see that it starts off at $3080\,\mathrm{B/ⅎ}$ at $0\,ⅎ$, and then increases before making an abrupt jump towards $2310\,\mathrm{B/ⅎ}$ at $360\,ⅎ$. At $1080\,ⅎ$, the byte rate drops to zero, signalling the non-availability of the network link. This outage persists until $1260\,ⅎ$, where the link seems to be re-established and the byte rate reaches a value of $3949\,\mathrm{B/ⅎ}$. At $1620\,ⅎ$, the link reaches its overall maximum speed of $4384\,\mathrm{B/ⅎ}$. Furthermore, it is clear that no jitter applied (steady blue line); also, predicted link byte rate, as described in this section, is clearly very close to the actual byte rate.

[2]The graphical representation is created by the `visualiser` component, which is a part of the provided code base; this tool creates a TEX file which visually represents the genesis resulting from our configuration, which, since the seed is provided, is derivable in a deterministic manner.

Listing 6.1: Configuration of the exemplary simulation

```
seed 26031991001

ticks 3600
byterate u/1000/5000
slot-length 180
network-uptime 0.90
relative-jitter 0
absolute-jitter 0
relative-prediction-time-error 0
relative-prediction-amplitude-error ~/0/0.01
absolute-prediction-time-error ~/0/10
absolute-prediction-amplitude-error 0
look-ahead 3600

algorithm-parameter alpha 0.9

request-series interval u/400/600 size ~/500000/60000 byterate ~/3800/100 start 1000 end
    3000
request tick 3400 byterate 3600 data 480000
```

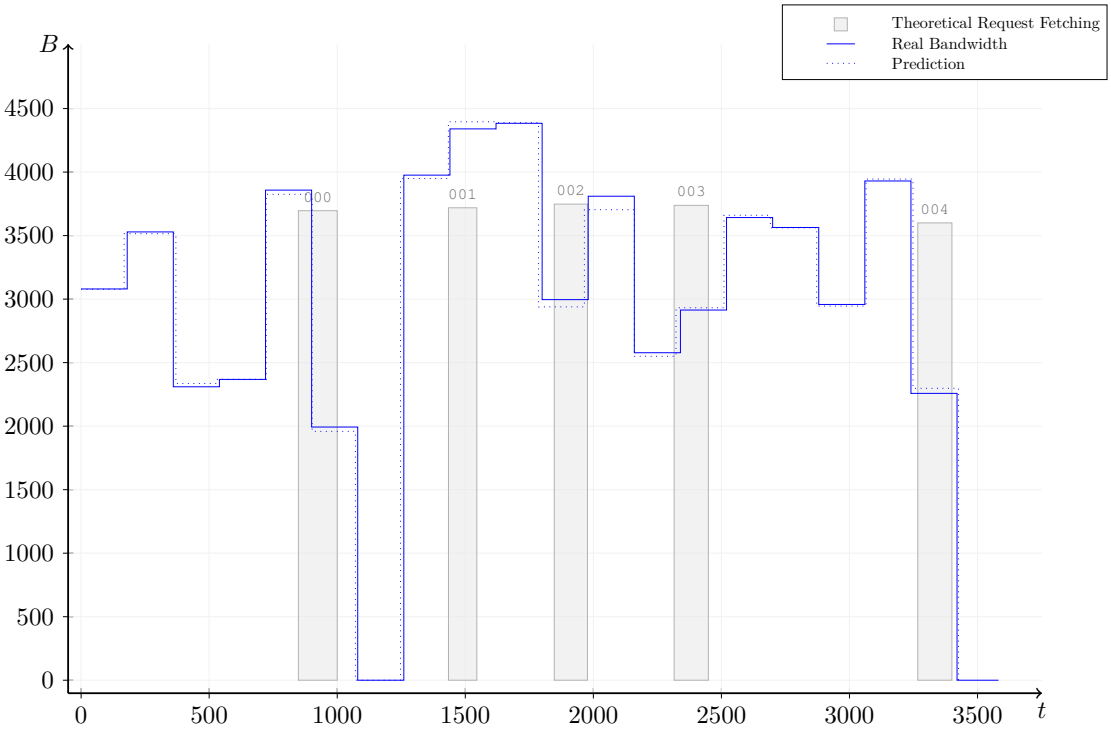


Figure 6.1: A graphical representation of the exemplary simulation's genesis

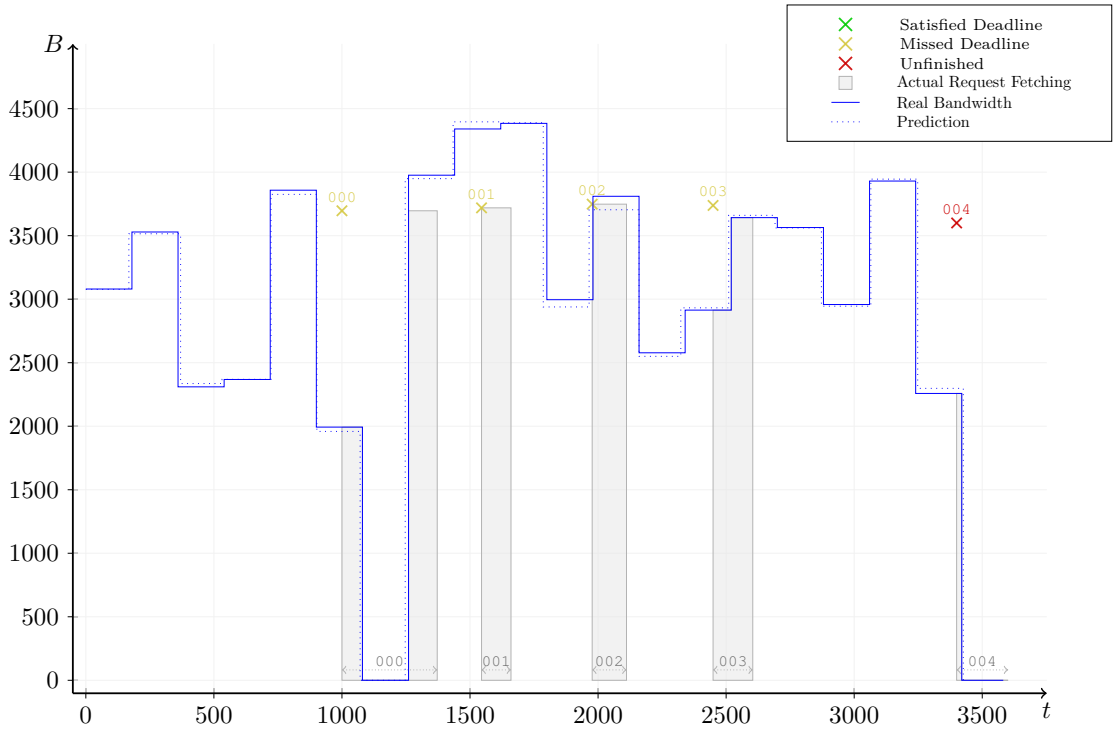


Figure 6.2: Exemplary simulation with strategy A (no prefetching)

6.2.2 Strategy A: No Prefetching

After having presented our exemplary simulation, we use the first prefetching strategy, namely strategy A. As stated above, this strategy is implemented in the `NullAlgorithm` class. A graphical representation of the resulting simulation is shown in figure 6.2. The representation of the results differs only slightly from that of a simulation's genesis, as one can see by comparing figure 6.1 with figure 6.2. The grey boxes determine at which times the actual fetching of a request took place, while the × marks symbolise the deadlines. The colour of the × marks indicates whether the deadline has been met (green), missed (yellow) or the request has not been finished during the simulation at all (red).

The nature of the strategy – namely, *no prefetching*, in other words, the fetches start only at the deadlines of the respective requests – can be easily seen by the fact that the grey boxes start at the time of the deadline (of the × mark) and end afterwards, resulting in all deadlines being marked as missed (yellow × marks); the last request – named `004` – is even unfinished because the connection is lost (byte rate of zero) towards the end of the simulation, leaving no time to finish fetching the request, which is signalled by the × mark being red.

Fetching of request `000` is started at its deadline, 1000 $\mathcal{T}$, and is clearly interrupted (suspended) during the period of non-functional communication link, indicated by the blue line showing a byte rate of zero. This makes request `000` the request with the

Table 6.1: Result metrics for exemplary simulation with strategy A

Metric	min	max	mean	median
$\mathcal{RT}$	115.00 $\mathcal{T}$	372.00 $\mathcal{T}$	194.00 $\mathcal{T}$	144.50 $\mathcal{T}$
$\mathcal{DA}$	115.00 $\mathcal{T}$	372.00 $\mathcal{T}$	194.00 $\mathcal{T}$	144.50 $\mathcal{T}$
$\mathcal{DV}$	411404.00 B	560520.00 B	489338.75 B	492715.50 B
Hit Ratio	0.00 % (0 / 5)			

highest $\mathcal{RT}$, 372 $\mathcal{T}$. Altogether, the numeric results of the measured metrics are displayed in figure 6.1. We see that the statistical metrics of $\mathcal{RT}$ and $\mathcal{DA}$ both have the same values, as expected, since in this scenario the data ageing and the response time correspond to the same time spans (data only ages during transmission, not in any prefetch cache).

We will compare the results from this strategy to other strategies in the summary of this section (see section 6.2.5).

6.2.3 Strategy B: Prediction-Ignoring Prefetching

The next strategy – strategy B – consists, as described, of the prefetching of data before its deadline, but regardless of the predicted link byte rate. Intuitively, this leads to good results in times when the link byte rate is high enough to accommodate the request (in other words, higher than the data source's bandwidth β).

Figure 6.3 shows the graphical representation of the results when applying strategy B. It is visible that request `001` has been served with its deadline being met (signalled by the green $\times$ mark); all the other requests still have unsatisfied deadlines because of the actual link byte rate being insufficient. In the figure, this becomes clear from the fact that the blue line indicating the byte rate is lower than the requests' byte rates, indicated by the Y position of the $\times$ marks. Clearly, the requests with unsatisfied deadlines – those with yellow $\times$ marks – are rated above link byte rate, while the satisfied request – `001` – is rated within the available link speed (underneath the blue line).

In figure 6.2, which shows the numerical results for strategy B, it is visible that strategy B is already an improvement compared to strategy A, since even though times of sub-par link quality cause deadline misses, there is an increased hit ratio (20.0 % instead of 0.00 %) and a significantly lower response time (an average of 23.00 $\mathcal{T}$ instead of 194.00 $\mathcal{T}$). Interestingly, the data age, $\mathcal{DA}$, has actually decreased, even though prefetching is facilitated. While this might seem counter-intuitive at first (we assumed any form of prefetching to increase data age since data is fetched ahead of its usage and stored somewhere), upon analysing the result, we see that the cause of the decreasing of $\mathcal{DA}$ is request `000`, which now takes a significant amount less time. In this instance, this stems from the fact that using strategy A, the fetching for `000` was started shortly before a period of no connectivity (zero byte rate), while strategy B dictated that fetching should start earlier – thus, strategy B avoids transferring the request during the time of outage, which reduces fetching speed, which in turn reduces data age.

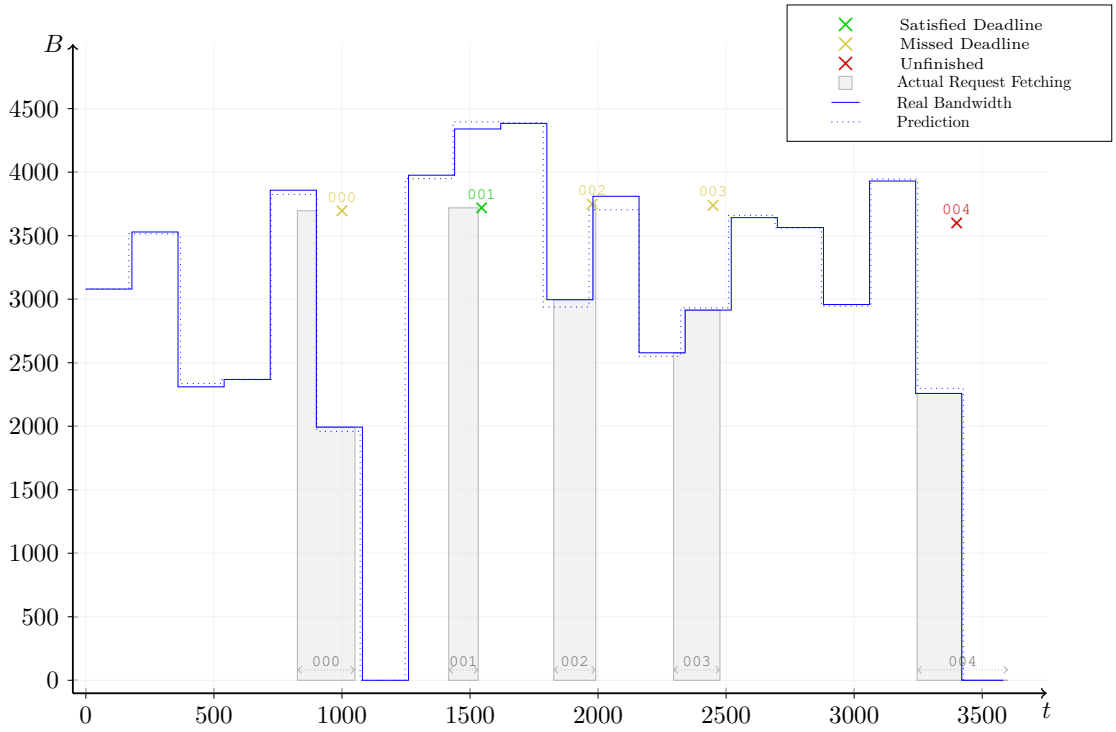


Figure 6.3: Exemplary simulation with strategy B (simple prefetching)

Table 6.2: Result metrics for exemplary simulation with strategy B

Metric	min	max	mean	median
$\mathcal{RT}$	0.00 $\mathcal{T}$	51.00 $\mathcal{T}$	23.00 $\mathcal{T}$	20.50 $\mathcal{T}$
$\mathcal{DA}$	128.00 $\mathcal{T}$	225.00 $\mathcal{T}$	174.50 $\mathcal{T}$	172.50 $\mathcal{T}$
$\mathcal{DV}$	411404.00 B	560520.00 B	489338.75 B	492715.50 B
Hit Ratio	20.00 % (1 / 5)			

6.2.4 Strategy C: Prediction-Ignoring Prefetching

Finally, we apply strategy C to our exemplary simulation, which consists of using our proposed prefetching algorithm including all its optimisations and features. The algorithm respects the (predicted) link byte rate and reacts accordingly, effectively minimising $\mathcal{RT}$.

Figure 6.4 shows the result of applying the algorithm; we can see that all requests have their deadlines met (green × marks), and one can observe how the algorithm worked its way around times of reduced link quality by starting the fetching sufficiently early.

The results as shown in figure 6.3 indicate that the prefetching algorithm has served its intended purpose. $\mathcal{RT}$ has been minimised (it is zero for all requests) while keeping $\mathcal{DA}$ to a low (we omit formal proof by stating that it is clear from figure 6.4 that the fetch could not have been issued significantly later, further minimising data ageing).

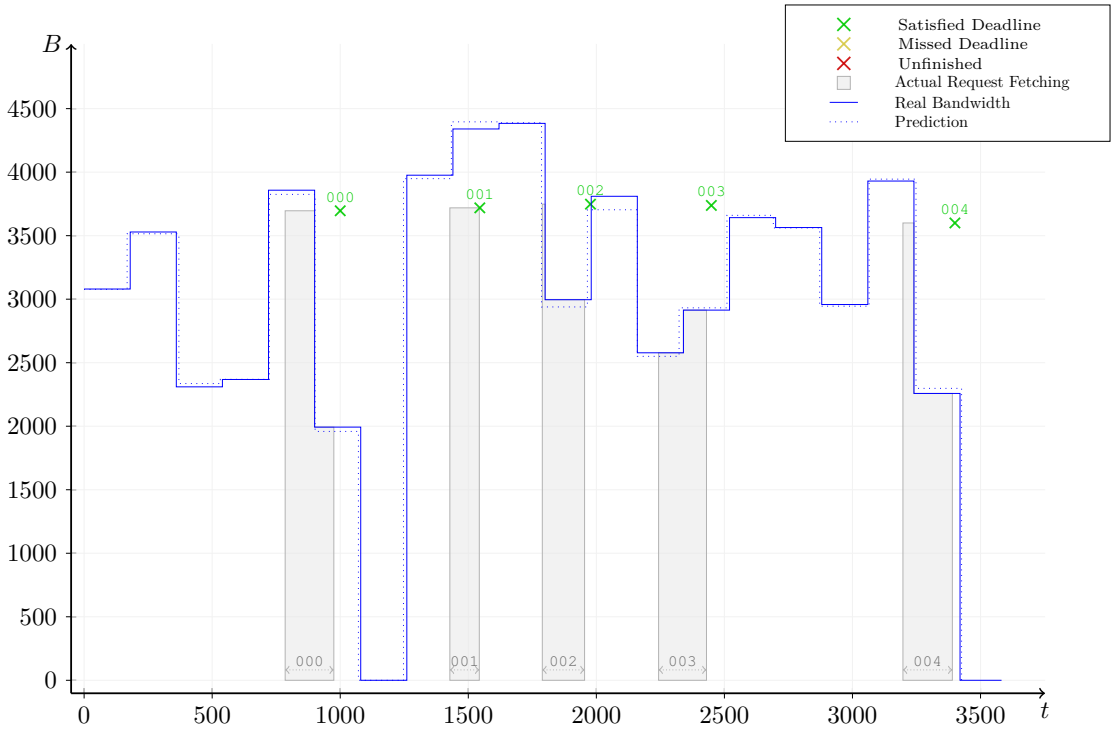


Figure 6.4: Exemplary simulation with strategy C (advanced prefetching)

Table 6.3: Result metrics for exemplary simulation with strategy C

Metric	min	max	mean	median
$\mathcal{RT}$	0.00 T	0.00 T	0.00 T	0.00 T
$\mathcal{DA}$	117.00 T	215.00 T	185.80 T	203.00 T
$\mathcal{DV}$	411404.00 B	560520.00 B	487471.00 B	485850.00 B
Hit Ratio	100.00 % (5 / 5)			

Also notable is the decreased average of $\mathcal{DV}$, which is caused by the fact that using strategy A and strategy B, fetching of the last request – `004` – could not be finished at all, so no $\mathcal{DV}$ value was recorded for this request, changing (increasing) the overall mean of $\mathcal{DV}$.

6.2.5 Summary of Exemplary Simulation

The exemplary simulation allows for an in-depth inspection of the mode of operation of the three strategies. We can clearly see that strategy A – using no prefetching – performs the worst with regard to response time ($\mathcal{RT}$), and we can actually observe the fact that fetching starts only at the deadlines of the requests in figure 6.2. Meanwhile, strategy B performs slightly better by taking into account the fact that requests require time to be

fetched, but disregarding the link quality prediction. In figure 6.3 we can see that requests in times of sub-par link quality are not being served on time – they are, however, being served in a substantially more timely manner than using strategy A. Finally, strategy C uses the algorithm we have described in chapter 4, and, as figure 6.4 shows us, it meets our expectations: all requests, regardless of the current link quality, are being served on time.

This analysis allows us to formulate our hypothesis and perform multiple test series to verify our algorithm's performance under different conditions. We perform these test series using experiments and describe our work and findings in the following sections.

6.3 Main Hypothesis

We formulate our main hypothesis: *Strategy C will perform better than strategy A and strategy B in terms of Response Time* $\mathcal{RT}$. This makes $\mathcal{RT}$ our *dependent variable* while there exists a number of *independent variables* (see the parameters of a simulation *configuration* in section 5.5.1. Our overall goal is to determine under which conditions (regarding the independent variable) our hypothesis holds.

In our first step, we examine each of our independent variables and decide on whether and how to perform a regression analysis on this independent variable. We then perform these regression analyses to determine for each independent variable a range of values in which our hypothesis may hold. For our regression analyses, we use a fixed set of values for the control variables and perform a series of simulation runs with changing values for the independent variable (*parameter sweep*). We identify ranges for the independent variable where our hypothesis holds by performing *t-tests*. The alpha level used in our t-tests was 0.05 (5 %).

During our experiments, we not only record $\mathcal{RT}$ as our primary target variable, but also $\mathcal{DA}$ as well as the *hit ratio* (the amount of requests being served instantly from the cache instead of having to wait for network transmission to finish). We discuss our results for each of our regression analysis runs.

6.4 Discussion of Independent Variables

For each of our variables, we need to define whether it is fixed (control variable) or sweeping (independent variable) in our regression analyses. In each analysis, we select one independent variable and run the simulation while only changing this variable, recording the results for each value of this variable (hence the term *sweeping*).

In the following sections, we discuss the values of our variables in both the fixed and the sweeping role. In other words, we define ranges for sweeping, and the value the variable has in situations where it is a control variable, with unchanged value.

For a summary of the variables as well as a presentation of the regression analyses results, see section 6.5.

6.4.1 Simulation Duration, Available Byte Rate and Slot Parameters

We regard the simulation duration (number of ticks) and the mean available byte rate and observe that the product of these two numbers results in a value which determines the overall number of bytes transferable over the simulated link throughout the simulation (we call it the *capacity* of the simulation in question). In order to reduce the number of necessary regression analyses, we fix the simulation duration, the available byte rate and the way slots are generated during the creation of the simulation genesis. We use the following values in all our regression analysis experiments:

- Simulation Duration: $36000\,\text{ℑ}$ (representing 10 hours in ticks of seconds).
- Byte Rate: uniform distribution between $30\,\text{kB/ℑ}$ and $200\,\text{kB/ℑ}$.
- Slot Length: normal distribution with $\mu = 120\,\text{ℑ}$ and $\sigma = 30\,\text{ℑ}$.
- Network Availability: 0.95 (5 % chance of a slot having zero byte rate).

This results in the following mean simulation *capacity*, which is fixed throughout all our experiments:

$$\text{capacity}_{\mu} = 36000 \times \frac{30000 + 200000}{2} \times 0.95 = 393300000$$

This means that the link of an average simulation is capable of transferring around 393.3 MB. For requests, we use a request size of 1967 kB and an interval of $1530\,\text{ℑ}$, which results in a load of around $0.1 = 10\,\%$ of the link *capacity*. The available byte rate of the requests is uniformly distributed between $30\,\text{B/ℑ}$ and $200\,\text{B/ℑ}$.

6.4.2 Network Jitter

Our a priori assumption is that while network jitter is influencing the simulation by adding entropy, it is not influencing the simulation in a way that would favour any of the three presented strategies. To confirm this assumption, we perform a regression analysis on the relative jitter[3] using values from 0.0 to 4.0 with steps of 0.04. We keep the absolute jitter at zero.

For regression analyses of other variables, where network jitter is a control variable and needs to have a fixed value, we use 0.05 (5 %).

[3]We operate based on the assumption that modifying the absolute jitter yields the same effect as modifying the relative jitter, as both result in a numerically absolute deviation from the nominal byte rate.

6.4.3 Prediction Amplitude and Time Error

The prediction errors influence only strategy C, we thus want to analyse in which ranges the errors significantly decrease performance of strategy C, compared to our control group, strategy A. Strategy B should also be completely uninfluenced by the prediction error.

Regarding time errors, we perform two series of regression analyses; one for the standard deviation σ and one for the mean μ error. During our initial experiments, we found that varying the standard deviation σ of the error did not produce noticeable differences in $\mathcal{RT}$ (see the discussion in section 6.5.2 for further details), and thus introduced the mean as an additional sweep variable. We sweep σ with values from 0.0 to 4.0 using steps of 0.05, and μ with values from -0.72 to $+1.00$ using steps of 0.04.

Similarly, for amplitude errors, we perform the same two regression analyses. We sweep the values for σ from 0.0 to 4.0 with steps of 0.05, and μ with values from -1.0 to $+1.0$ using steps of 0.04.

For regression analyses of other variables, where the prediction error is a control variable and needs to have a fixed value, we use means of 0 % and standard deviations of 0.01 (5 %).

6.4.4 Look-Ahead Time

The look-ahead time influences both strategy B and strategy C. Since we fixed the simulation duration to $36000\,\mathcal{T}$ (see section 6.4.1), possible look-ahead values lie between $0\,\mathcal{T}$ and $36000\,\mathcal{T}$. Initial experiments showed that with our current selection of variables, values between $250\,\mathcal{T}$ and $36000\,\mathcal{T}$ yield no significant change; stemming from this observation, we decided to change the testing interval; we perform a sweep between $0\,\mathcal{T}$ and $250\,\mathcal{T}$ with a stepping of $5\,\mathcal{T}$ (resulting in 50 simulation runs).

For regression analyses of other variables, where the look-ahead time is a control variable and needs to have a fixed value, we use $18000\,\mathcal{T}$.

6.4.5 Error Correction Factor α

The parameter α is described in detail in section 4.5.3. It reduces the estimation used by the algorithm by a constant factor, to compensate for unforeseen jitter. We perform a sweep between 0.24 and 1.00 with steps of 0.02.

For regression analyses of other variables, where α is a control variable and needs to have a fixed value, we use 0.9, which results in a usage of 90 % of the predicted byte rate for scheduled prefetches.

6.5 Regression Analysis Experiments

In this section, we summarise the variables used for our simulations; subsequently, we perform the discussed regression analyses and show our results. Our variables consist of four fixed variables, which have the same value throughout all simulation runs – they are called F1 through F5. Furthermore, there are eight independent variables which we run tests for, S1 through S8[4]. Figure 6.4 gives an overview of our analysis variables.

Table 6.4: Specification of independent variables for regression analysis

#	Variable	Type	Control	Sweep (from; step; to)	Unit
F1	Capacity (see section 6.4.1)	fixed	393.3		MB
F2	Request Load (see section 6.4.1)	fixed	10		%
F3	Abs. Jitter	fixed	0		B/s
F4	Abs. Prediction Amplitude Error	fixed	0		B/s
F5	Abs. Prediction Time Error	fixed	0		$\mathcal{T}$
S1	Rel. Jitter σ ($\mu = 0$)	sweep	0.05	$(0.0; 0.04; 4.0)$	ratio
S2	Rel. Prediction Amplitude Error σ	sweep	0.10	$(0.0; 0.05; 4.0)$	ratio
S3	Rel. Prediction Time Error σ	sweep	0.05	$(0.0; 0.02; 1.0)$	ratio
S4	Look-Ahead Time	sweep	18000	$(0; 5; 250)$	$\mathcal{T}$
S5	Error Correction Factor α	sweep	0.90	$(0.24; 0.02; 1.00)$	ratio
S7	Rel. Prediction Amplitude Error μ	sweep	0.00	$(-0.72; 0.20; +2.00)$	ratio
S8	Rel. Prediction Time Error μ	sweep	0.00	$(-1.00; 0.04; +1.00)$	ratio

Furthermore, for each variable combination, we actually perform 500 single simulation runs, each with an identical configuration, but a different seed, to increase the results' approximation towards a statistical mean value. The single simulations always use the seeds 199100 through 199699. For each single run, we use each one of our three strategies (A, B and C)

For our dependent variables (the measured values), we choose the three variables measuring the performance of our algorithm, namely the response time ($\mathcal{RT}$), the data age ($\mathcal{DA}$) and the hit rate (hit). For both values, we use the mean of mean results from the 500 simulation runs using a specific input variable vector.

Table 6.5: Specification of dependent variables for regression analysis

#	Variable	Unit
D1	Mean Response Time ($\overline{\mathcal{RT}}$)	$\mathcal{T}$
D2	Mean Data Age ($\overline{\mathcal{DA}}$)	$\mathcal{T}$
D3	Mean Hit Rate ($\overline{\mathrm{hit}}$)	ratio

In the following, we display the results of the regression analyses. For detailed numerical results, see appendix B.

[4]An unused independent variable has been removed after the design of the experiments, thus there is no variable named S6.

6.5.1 Relative Jitter

We performed a regression analysis for the independent variable *Relative Jitter* (S1); more specifically, we varied the standard deviation σ for the relative jitter. All other variables remained according to figure 6.4. Results are listed in section B.1 as well as depicted in figure 6.5 and figure 6.6.

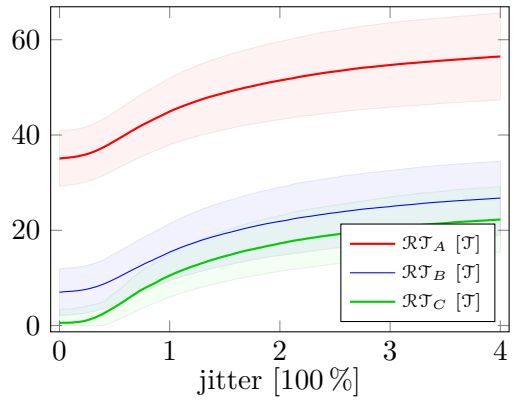


Figure 6.5: Response time over jitter

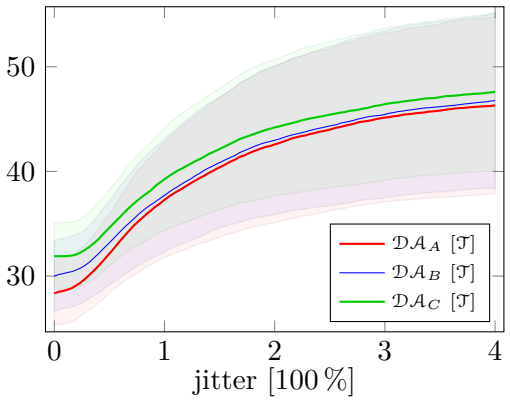


Figure 6.6: Data age over jitter

From the result it is clear that strategy C provides a better result regarding $\mathcal{RT}$ across the entire domain It is worth noting that strategy B performs only slightly worse than strategy C, in other words, taking the link byte rate into account provides an advantage smaller than the advantage of prefetching at all.

Data age is, as expected for prefetching, always higher than if no prefetching is involved, however, it is noteworthy that strategy B shows results rather close to those of strategy A. However, the difference between $\mathcal{DA}_B$ and $\mathcal{DA}_C$ is around $2\,\mathcal{T}$ for a jitter between 0.0 and 0.2, and even smaller ($1\,\mathcal{T}$) across the remaining testing domain, which is a relatively small setback compared to a fetching time of $20..30\,\mathcal{T}$. We argue that a possible reason for the increased data age is the fact that strategy C tends to move fetching to times with lower byte rates, which increases fetching time and thus data age. Analysing this hypothesis is part of our planned future work.

Lower jitter values are naturally better, regardless of what fetching strategy is being used. However, strategy C outperforms its competitors across all jitter values. We tested jitter values of up to 4 (400 %), which should cover practically all real-world link connection scenarios.

t-tests performed on the results indicate that $\mathcal{RT}$ is always best when using strategy C. When regarding $\mathcal{DA}$, strategy C is always performing worst, however, starting at a jitter value of just above 3, the t-test consistently reports that results from strategy C and strategy B are statistically indifferent (in other words, starting at a jitter value of 3, they yield results similar enough to be considered equivalent).

6.5.2 Prediction Amplitude Error Deviation σ

We performed a regression analysis for the independent variable *Relative Prediction Amplitude Error* σ (S2); thus, we varied the standard deviation σ for the relative prediction amplitude error. All other variables remained according to figure 6.4. Results are listed in section B.2 as well as depicted in figure 6.7 and figure 6.8.

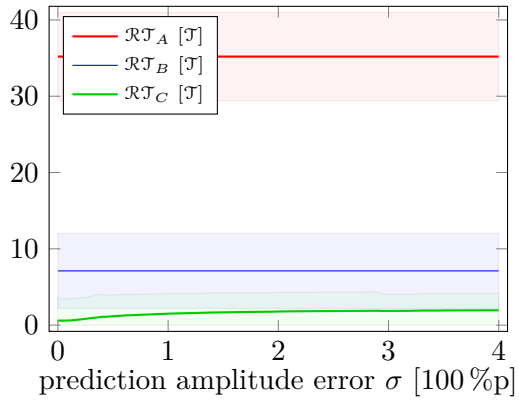


Figure 6.7: Response time over prediction amplitude error σ

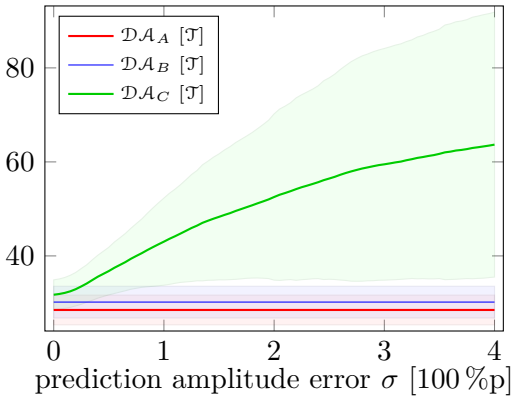


Figure 6.8: Data age over prediction amplitude error σ

Interestingly, the standard deviation σ of the prediction amplitude error influences the response time only very slightly. We observe that an increased *deviation* of predictions barely influences response time, while data age is increased. While this contradicted our intuitions, upon analysing the results, we concluded that an increased standard deviation of prediction error can (asymptotically) cause two scenarios for a particular request: extreme *over-estimation* of data link quality, and extreme *under-estimation* of data link quality. While over-estimation generally leads to bad results (which led us to expect a higher increase in $\mathcal{RT}$ for higher σ values), another effect is responsible for $\mathcal{RT}$ staying rather low: requests themselves have a maximum byte rate β, which bounds their transmission speed (and is also respected by the scheduling algorithm). If an extreme over-estimation occurs, it is higher than the request byte rate, causing the algorithm to disregard the seemingly high link byte rate and take into account only the (relatively low) request byte rate for its calculations.

On the other hand, if a heavy *under-estimation* occurs, the effect is that the algorithm tries to schedule requests for a very early time, to compensate for the low link quality. This, however, results in an increase of data age $\mathcal{DA}$, as it is clearly visible in figure 6.7.

We observe that while $\mathcal{RT}$ is robust against prediction amplitude errors, the error deviation should be between 0.00 and around 0.30 to avoid an unreasonable increase in $\mathcal{DA}$. t-tests show that both $\mathcal{RT}$ and $\mathcal{DA}$ are always statistically different enough to be considered non-equivalent. Strategy C has the best $\mathcal{RT}$ and the worst $\mathcal{DA}$ across the entire domain.

6.5.3 Prediction Amplitude Error Mean μ

We performed a regression analysis for the independent variable *Relative Prediction Amplitude Error* μ (S7); thus, we varied the mean μ for the relative prediction amplitude error. All other variables remained according to figure 6.4. Results are listed in section B.6 as well as depicted in figure 6.9 and figure 6.10.

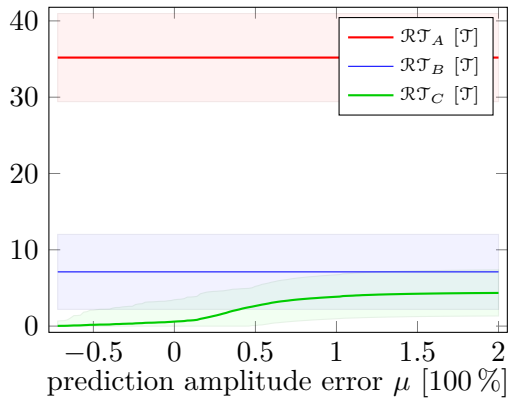


Figure 6.9: Response time over prediction amplitude error μ

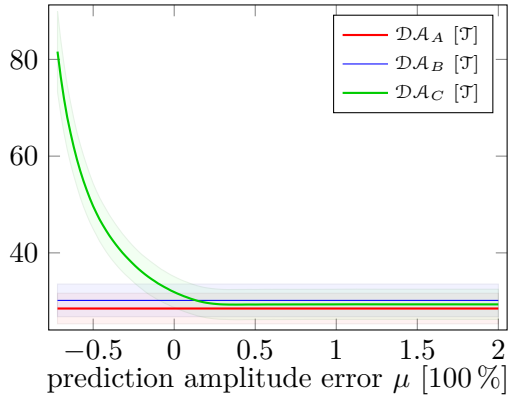


Figure 6.10: Data age over prediction amplitude error μ

The prediction amplitude error μ has two main value domains; values below zero determine that prediction is under-estimating the link. For these values, naturally, $\mathcal{RT}$ is low (for very low values for μ it is zero). This is caused by the algorithm under-estimating the link and expecting the fetch to take longer than it does – this reflects in an increase in $\mathcal{DA}$, as is clearly visible in figure 6.10.

Starting from zero, increasing μ causes the prediction to over-estimate, which reflects in an increased $\mathcal{RT}$ since the algorithm expects fetching to take to little time. The increase is not steep; this is caused by a similar phenomenon as described in section 6.5.2: the maximum byte rate of a request predominates the seemingly (predicted) high link byte rate, moderating the effect of the algorithm over-estimating the actual link speed.

We see that in order to have a reasonable $\mathcal{DA}$ (in this case, *reasonable* means that the mean of the $\mathcal{DA}_C$ is within the bounds of the standard deviation σ of $\mathcal{DA}_A$ or $\mathcal{DA}_B$), the prediction error amplitude μ should not be lower than 0.1; in other words, under-estimating the link leads to a drastic increase in $\mathcal{DA}$. On the other hand, increasing μ above zero, while causing the algorithm to perform worse in terms of $\mathcal{RT}$, the impact of positive μ on $\mathcal{RT}$ was not as severe as the impact of negative μ on $\mathcal{DA}$. This is a consideration necessary when designing a prediction algorithm.

t-tests performed on the results indicate that strategy C is always outperforming both its competitors without a reasonable statistical probability of chance with respect to $\mathcal{RT}$; $\mathcal{DA}$, however, is worse for strategy C at the beginning (negative μ) but becomes better with increasing μ. During the transition (around values for μ of 12..16 %), the t-test indicates a statistical comparability of $\mathcal{DA}$ using strategy B and strategy C.

6.5.4 Prediction Time Error Deviation σ

We performed a regression analysis for the independent variable *Relative Prediction Time Error* σ (S3); thus, we varied the standard deviation σ for the relative prediction time error. All other variables remained according to figure 6.4. Results are listed in section B.3 as well as depicted in figure 6.11 and figure 6.12.

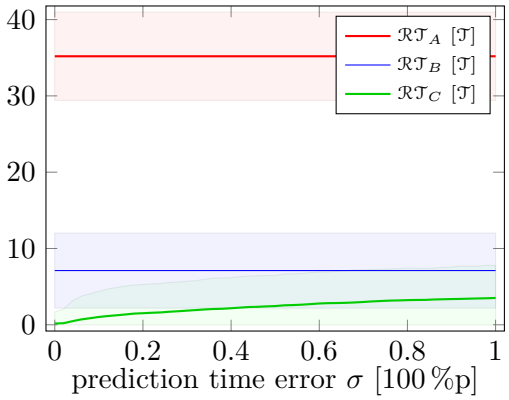


Figure 6.11: Response time over prediction time error σ

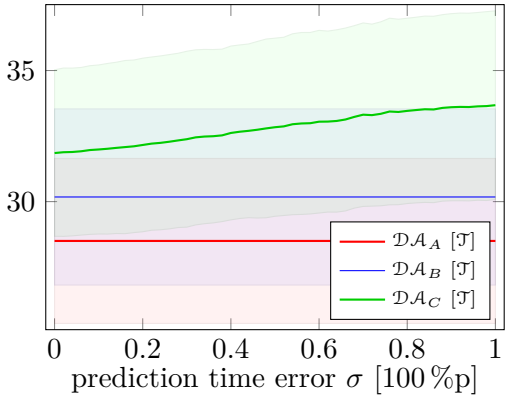


Figure 6.12: Data age over prediction time error σ

The relative prediction time error changes the temporal behaviour of the prediction function (see figure 5.9 and section 5.5.1 for more details). It is noticeable that while the impact of an increased prediction time error deviation is higher than an increased prediction amplitude error deviation (see section 6.5.2), it is still rather low, and outperforming strategy B throughout the entire test domain. We tested values up until 100 %, which leads to a standard deviation of 100 % (relative to the duration of a slot).

t-tests performed on the results indicate that both $\mathcal{RT}$ and $\mathcal{DA}$ are always statistically different enough to be considered non-equivalent. As mentioned, strategy C has the best $\mathcal{RT}$ and the worst $\mathcal{DA}$ across the entire domain.

6.5.5 Prediction Time Error Deviation μ

We performed a regression analysis for the independent variable *Relative Prediction Time Error* μ (S8); thus, we varied the mean μ for the relative prediction time error. All other

variables remained according to figure 6.4. Results are listed in section B.7 as well as depicted in figure 6.13 and figure 6.14.

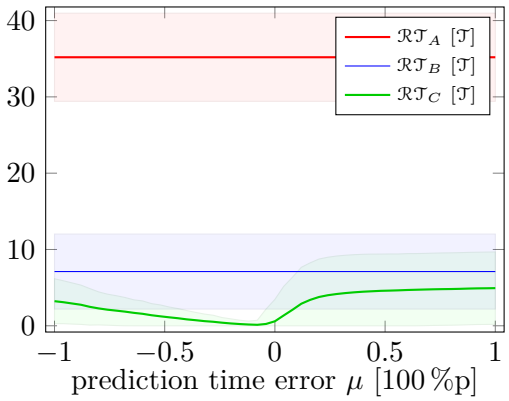


Figure 6.13: Response time over prediction time error μ

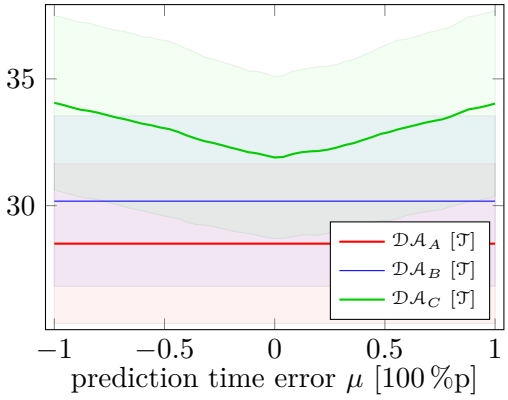


Figure 6.14: Data age over prediction time error μ

The prediction time error mean specifies by how much the temporal aspect of byte rate prediction is biased; in other words, a negative value causes a prediction of byte rate change ahead of time on average, while a positive value causes the prediction to be late compared to actual changes. Naturally, for best results, the prediction should be as close as possible to zero. However, we see that there is a difference in performance between the negative and the positive changes of μ: negative values of μ cause less increase in $\mathcal{RT}$ than positive values. The reason for this symptom is that negative values of μ essentially shift the predicted time line of the link quality back in time (predictions happen before actual changes in link quality), while positive values shift the predicted time line forward in time (predictions happen after actual changes). Hence, negative values give the algorithm more time to schedule.

t-tests performed on the results indicate that both $\mathcal{RT}$ and $\mathcal{DA}$ are always statistically different enough to be considered non-equivalent. As mentioned, strategy C has the best $\mathcal{RT}$ and the worst $\mathcal{DA}$ across the entire domain.

6.5.6 Look-Ahead Time t_{horizon}

We performed a regression analysis for the independent variable *Look-Ahead Time* (S4). While we performed a sweep of the look-ahead time parameter t_{horizon}, all other variables remained according to figure 6.4. Results are listed in section B.4 as well as depicted in figure 6.15 and figure 6.16.

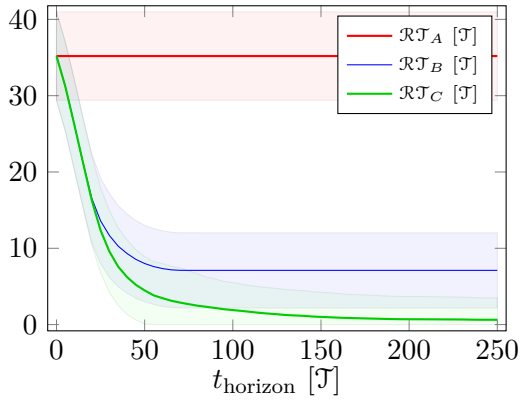


Figure 6.15: Response time over t_{horizon}

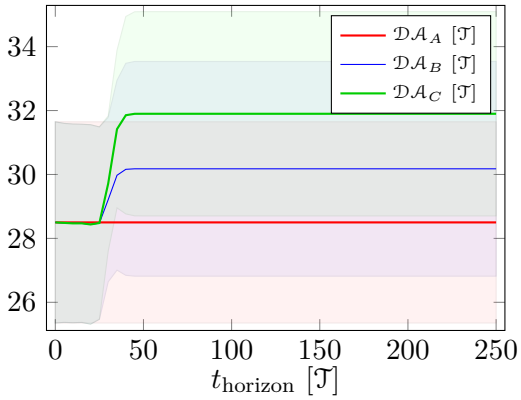


Figure 6.16: Data age over t_{horizon}

As the look-ahead time decides to what extent the scheduling algorithm is provided with future requests, it is clear that an increased value provides better results. More specifically, there is a threshold above which results are optimal (and thus identical); for our simulation, the threshold above which $\mathcal{RT}$ is more or less constant around $120\,\mathcal{T}$. Our request interval is $1530\,\mathcal{T}$, and the mean time of fetching a request without any prefetching or caching is $28.5\,\mathcal{T}$ (see section B.4). We see that $\mathcal{RT}$ is decreasing fast with the increase if t_{horizon} up to the point of around $55\,\mathcal{T}$, which is roughly twice the fetching time. $\mathcal{RT}_B$ does not change at all with a t_{horizon} of $80\,\mathcal{T}$ or higher. It can be concluded that a t_{horizon} of roughly 2-3 times of the average fetching time is sufficient to make a reasonable fetching prediction.

Data age is actually rising until a t_{horizon} value of around $45\,\mathcal{T}$, which is caused by the fact that with small look-ahead values, the algorithm effectively performs no prefetching, resulting in a low $\mathcal{DA}$ value. This, however, comes at the cost of a high $\mathcal{RT}$.

Results for $t_{\text{horizon}} = 0\,\mathcal{T}$ yield the same result for all strategies used, which is obvious since a look-ahead time of zero effectively disarms any scheduling algorithm. t-tests performed on the results for $\mathcal{RT}$ show that results for strategy B and strategy C are similar enough to be considered equivalent until a look-ahead time of around $25\,\mathcal{T}$ is reached ($t = 3.51$). From the graph it is visible that they both diverge quickly from results of strategy A.

Regarding $\mathcal{DA}$, we see that all strategies yield comparable results for the first few iterations; this is confirmed by the results of t-tests performed, which indicate that until $t_{\text{horizon}} = 30\,\mathcal{T}$ is reached, results from all strategies yield statistically similar results.

6.5.7 Error Correction Factor α

We performed a regression analysis for the independent variable *Error Correction Factor* α (S5). While we performed a sweep of α, all other variables remained according to figure 6.4. Results are listed in section B.5 as well as depicted in figure 6.17 and figure 6.18.

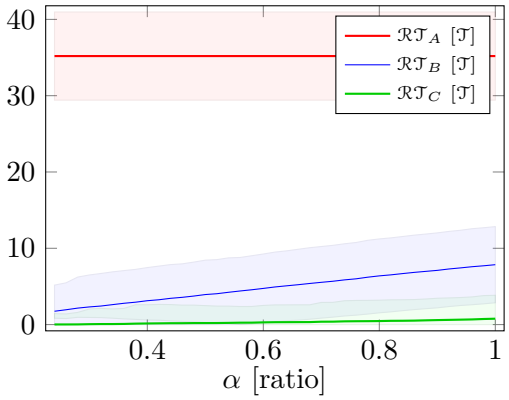


Figure 6.17: Response time over α

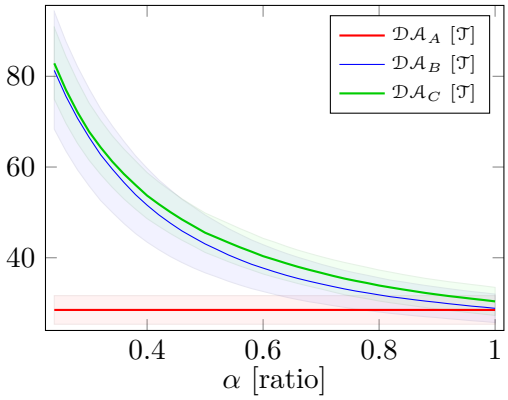


Figure 6.18: Data age over α

As described in section 4.5.3, the error correction factor α decides how much of the predicted byte rate the algorithm actually uses. The goal of this factor is to provide a counter-measure for network jitter and unpredicted periods of low byte rate (or network outages). Our simulations were controlled by our input parameters, and as such, the results of the regression analysis for α have little meaning for real-world applications, since our experiment setup is merely an approximation of actual link behaviour. As such, the analysis for α should be re-run on a real-world test setup – this we plan to do in the future.

In our analysis, we can see that the more α is moved from 1 to 0, the lower $\mathcal{RT}$ becomes – which is intuitive, since the scheduling algorithm reserves more link capacity. Note that the decrease of $\mathcal{RT}$ is rather low, while the increase of $\mathcal{DA}$ – which is caused by the algorithm scheduling fetches earlier with decreasing α – is drastic. In situations where data age is playing a major role, thus, α should be kept high, maybe even at 1 (disabling the prediction error amortisation feature completely).

Performing t-tests on the resulting numbers shows that even though the $\mathcal{DA}$ results for strategy B and strategy C seem similar, they are statistically different enough to not be considered equivalent. It is, again, safe to say that strategy C outperforms all other strategies regarding $\mathcal{RT}$ and performs the worst regarding $\mathcal{DA}$; these statements hold across the entire domain.

6.6 Result Analysis

In order to summarise the results of our regression analysis experiments, we deduced the following statements from our experiments given the described test setup as the testing environment, and propose these guidelines for optimising the performance of strategy C:

Jitter Jitter influences A, B and C equally; above a relative jitter of 3.0, B and C show statistically similar results.

For best results of strategy C, jitter should be as close to zero as possible. Worst results of strategy C occur at high levels of jitter (influencing both $\mathcal{RT}$ and $\mathcal{DA}$).

Relative Prediction Amplitude Error Mean (μ) $\mathcal{RT}$ is mostly robust with respect to changes of μ; $\mathcal{DA}$ increases when μ becomes negative. *For best results of strategy C, μ should be between 0.0 and 0.3, negative μ should be avoided if $\mathcal{DA}$ is an issue.* Worst results of strategy C occur at very negative values for μ.

Relative Prediction Amplitude Error Deviation (σ) $\mathcal{RT}$ is mostly robust against changes in σ; $\mathcal{DA}$ increases with higher σ.

For best results of strategy C, σ should be as close to zero as possible. Worst results of strategy C occur at high values for σ.

Relative Prediction Time Error Mean (μ) $\mathcal{RT}$ is mostly robust against changes in μ; $\mathcal{DA}$ increases for higher absolute values of μ, however, not drastically.

For best results of strategy C, μ should be as close to zero as possible; slightly negative values perform better than slightly positive values. Worst results of strategy C occur at high values for μ.

Relative Prediction Time Error Deviation (σ) $\mathcal{RT}$ and $\mathcal{DA}$ are mostly robust and show barely any significant increase.

For best results of strategy C, σ should be as close to zero as possible. Worst results of strategy C occur at high values for σ.

Look-Ahead Time ($t_{\mathbf{horizon}}$) Low values of t_{horizon} cause an increase of $\mathcal{RT}$; values above around $50\,\mathcal{T}$ – close to twice the duration of a request fetch in the presented simulation – show stable results in our setup.

For best results of strategy C, $t_{horizon}$ should be as high as possible in general. Since this is not always feasible, we propose to keep t_{horizon} at least above the minimum fetch time plus the average request interval. Worst results of strategy C occur at $t_{\text{horizon}} = 0\,\mathcal{T}$.

Error Correction Factor (α) Low values of α cause a drastic increase of $\mathcal{RT}$; values around 1.0 show the best results.

For best results of strategy C, α should be as close as possible to 1.0.[5] Worst results of strategy C occur at $\alpha \sim 0$.

[5] Note that there might be an increase of $\mathcal{RT}$ around $\alpha = 1.0$ if the network jitter is particularly high.

CHAPTER 7

Conclusion

7.1 Findings

We presented the problem of a mobile unit with a connection link to a data source, where the link is of variable quality (volatile network condition). In our work, we discussed that knowledge of the underlying link and its quality – especially the change of its bandwidth, or effective byte rate, over time – can be used to enhance the Quality of Experience as perceived by the user. The main aspect of this improvement is the fetching of data ahead of time (prefetching) in situations where a substantial degradation of link quality is expected or predicted.

Our main presupposition is that for each point in time in the future, information about the amount of data transmittable per given time span is known to some degree. While this assumption is a rather strong one – it is difficult enough to measure the current link bandwidth, and predicting it depending on time and location can certainly not be regarded as an easier task –, it is generally possible to predict rough tendencies stemming from types of network coverage (2G versus 3G or 4G); furthermore, in our experiments analysing the exact behaviour of the improvement provided by prefetching for different degrees of prediction accuracy, we learned that even with inexact prediction, prefetching provides an improvement of Quality of Experience by reducing the average user response time ($\mathcal{RT}$) – see sections 6.5.2 through 6.5.5.

We therefore state that it is possible to improve user experience even with rough knowledge of network quality. To do so, one needs to employ prefetching as an inherent characteristic of the data service application. This is possible only by establishing another presupposition, namely that service requests are known in advance. While this is generally easier to manage, since applications can register scheduled requests ahead of time (for example through a prefetching middleware API), one still has to take into account possibilities of unplanned requests, for instance when the user requests data intermittently, but also when the context changes, be it because the user has diverged

from the planned route and a new route has been calculated. Still it can be established that at least some of the data can be fetched ahead of time given knowledge of its schedule.

Assuming that knowledge about the link quality and future requests is available, we proposed an algorithm for scheduling fetches of request-based data; in other words, a strategy for reaching prefetching decisions based on the assumed knowledge. The core idea of the algorithm is based on function integration and is discussed in section 4.5.1. We then describe how to apply this concept in our scenario in section 4.5.3 and present a reference Java implementation of our algorithm in section 4.6.

We defined performance criteria (target variables) for a scheduling strategy and built a framework for simulating certain scenarios in order to evaluate performance strategies. The framework allows for representing two communication parties and a link with controllable, varying byte rate, and is discussed in sections 5.4 and 5.5.

Subsequently, we performed testing and analysis by selecting three candidate strategies (section 6.1) and discussing an exemplary simulation using these three strategies section 6.2). We then presented our main hypothesis (namely that prefetching with regards to the context, in other words, the available link speed, enhances user experience; see section 6.3) regarding algorithm performance and ran multiple regression tests using different variables (section 6.5). Our results show that even under sub-par network conditions (high jitter, imprecise prediction), the presented algorithm performs well and provides the lowest response time compared to the other strategies (one of which is the strategy of performing no prefetching at all, in other words, *fetch on demand*). We have seen that while using a no-prefetching strategy in our exemplary use cases almost consistently yielded a user-perceived waiting time of $35\,\mathcal{T}$, employing prefetching caused the waiting time to generally stay well below $10\,\mathcal{T}$, which is a reduction of around $70\,\%$.

It can therefore be stated that fetching data ahead of time is generally a very promising way of enhancing user experience by reducing waiting times. Prefetching without knowledge of link quality is by itself providing good performance (significantly better than no prefetching, at rather low costs[1]), and given knowledge about the predicted link quality allows for further optimisation of the fetching schedule.

7.2 Outlook, Future Work

We have established a basic understanding of how prefetching is applicable in the context of mobile users, and how certain prefetching mechanisms influence the outcome variables. In this section, we discuss the outlook derived from our results and identify fields of possible future work.

[1] Costs in this context are increased data age and the need to cache data between the fetch and the actual usage.

7.2.1 Real-World Evaluation

Despite attempts of designing the setup as realistically as possible, our described scenario is still merely a theoretical model, an approximation of a certain situation. Transferring knowledge gained from this model and its simulations towards applications in the real world, in other words, using the prefetching strategy in actual network service applications, is a challenge that is still to be undertaken. We have identified several challenges to this task, which we will discuss in this section.

Predicting Link Quality

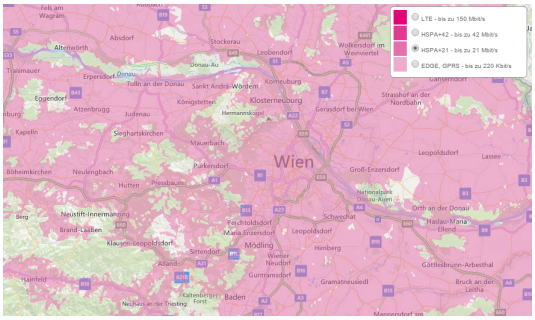
Probably the most challenging task in applying the described model in the real world is the matter of predicting link quality (in other words, providing B_{link} to the algorithm). We propose to further research several methods of doing so:

Network Coverage Maps Many mobile service operators provide maps of service coverage, often differentiating between mobile network standards (2G, 3G or 4G) or even between transfer rates available in different areas. While these maps are often rough and only accurate to some degree, they provide a valuable source of information for link quality prediction. Such maps usually stem from computer models, which calculate from a topographical map and positions and attributes of broadcast transmitters an approximated map of reception, determining whether network coverage is likely at a given point and what network quality is to be expected. Examples of such maps are shown in figure 7.1.

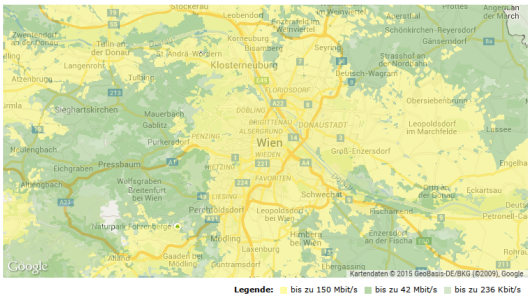
Empirical Measurement Determining the link speed empirically, for instance, by driving along roads and measuring network speed, is more precise than using network coverage maps, yet requires an order of magnitude more work to achieve. Advantages include the fact that the actual network speed is measured, regardless of the network standard (2G, 3G or 4G) used, as the fact that 4G is available at a certain location does not necessarily imply any certain byte rate. Such measurements, however, would have to be repeated periodically to be precise, since network coverage not only changes with varying conditions (weather, network load caused by events gathering many users, etc.) but also over time, when providers extend or modify their infrastructure. A downside, apart from the effort necessary to drive along roads, is the fact that in order to measure network speed, a certain amount of data must be transferred.[2]

Crowdsourcing An approach similar to empirical measurement would be to employ crowdsourcing to aggregate network reception information. This can be done either passively, by monitoring the network standard type and reception quality and sending periodic reports together with the current location, or actively, by employing the same methods as described in empirical measurement to determine

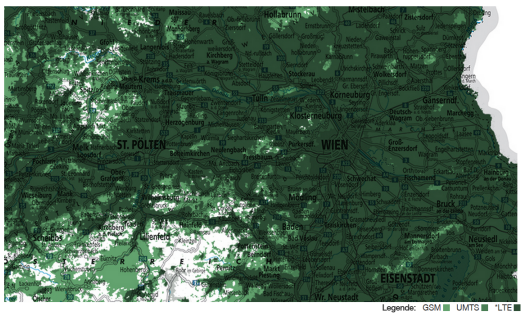
[2] See [JD02] for an approach of measuring the available link speed with minimal intrusive methods.



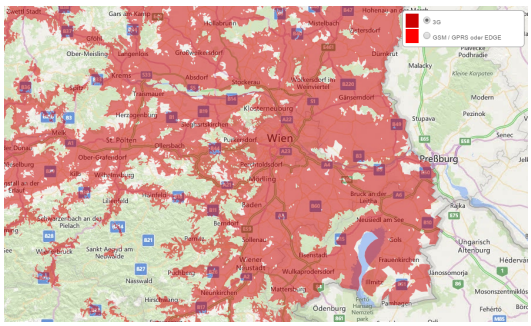
(a) T-Mobile (source: [T-M15b])



(b) A1 (source: [A1 15])



(c) Drei (source: [Hut15])



(d) tele.ring (source: [T-M15a])

Figure 7.1: Coverage maps of different Austrian mobile network providers, showing the available degree of granularity and bandwidth description (Details of the Viennese urban region)

the exact byte rate and report it. This is similar to what Google Inc. is using to provide live traffic information (see [Bar09]).

Predicting Future Requests

As a second kind of prediction necessary to plan prefetching, knowledge about future requests is necessary. While this kind of prediction is intuitively easier to make, one still requires to take into account this requirement.

To construct a method of satisfying this requirement, we propose to employ an API between the user code (software containing the business logic) and the middleware responsible for prefetching. This interface enables the user code to register planned requests ahead of time, similar to how this is achieved in our simulation: The code would notify the middleware of the deadline, amount of data and available resource bandwidth. In a real-world use case scenario, the user code could, for example, register a callback method (or interface) for perform the actual fetch. The middleware, in return, would schedule planned prefetch times, and invoke those callback methods at the respective time.

7.2.2 Cross-Relationships of Influencing Variables

We have performed regression experiments for our influencing variables in section 6.5, however, it is possible (and likely) for inter-variable relationships to exist; these correlations might have a significant impact on the algorithm performance. One possible domain of future work is to establish these relations and analyse their impact.

7.2.3 Fine-Grained Network Quality Evaluation

For simplicity, we have only taken into account the byte rate (data speed) of the communication link and analysed its impact on network services; we have stated that the impact of delay and jitter is ultimately comparable to a reduced bandwidth. This aspect is one which allows for deep subsequent research; factors influencing the transmission quality include:

- Latency
- Bandwidth
- Frequency of Disconnections
- Jitter in Latency and Bandwidth

7.2.4 Optimisation using Self-Learning Algorithms

In our current implementation, we use a constant factor (α) as a mean of correcting inexact predictions. One way of enhancing this implementation would be to employ a self-learning algorithm instead of a constant factor (possibly coupled to the crowdsourcing approach described in section 7.2.1); the algorithm could then adapt to the scale of precision of the link bandwidth estimation.

7.2.5 Optimisation for Data Streams

The model presented in our work is heavily request-oriented; a request for data is a rather short-term transmission of data with a given data amount and bandwidth. This means that streams of data are modelled in our domain as several small requests, with no semantic connection between them. It could help the scheduling algorithm to take into account the fact that those requests pose coherent streams of data – the Fuzzy Adaptive Buffering algorithm (FAB) presented in [Bag11] could for example be the subject of investigating further optimisation possibilities.

APPENDIX A

Analysis Experiment Reproduction

All references to files or directories, unless otherwise noted, relates to the code base of the prefetch simulation environment; this source code, information and documentation is available at `https://github.com/michael-borkowski/prefetch-simulation` or `http://www.borkowski.at/prefetch-simulation`. The version of the project `prefetch-simulation` used in our analysis was 0.0.2.

The prerequisites for all of the processes described in this appendix are a UNIX-like environment[1], a Java 8 run-time environment installation as well as Apache Maven 3.0.4 or later, which we use as a build tool, and a working Internet connection. Furthermore, in order for the running scripts to generate PDF files from the generated TEX sources, a TEX installation is required on the target system.

For processing our simulations, we use two bash scripts: `perform` and `perform_abc`. While `perform` is responsible for running a single simulation configuration, the script `perform_abc` is used to process a single configuration using three strategies (A, B and C, as described in section 6.1).

[1] Execution on other platforms should be possible in a similar way due to the platform-independent nature of Java; however, the exact syntax of the steps may differ from the ones described here.

A.1 Environment Preparation

To perform simulations, we need to prepare the environment by cloning the git repository containing the simulation source code and checking out the version used in this thesis:

```
~ $ git clone git@github.com:michael-borkowski/prefetch-simulation.git
~ $ cd prefetch-simulation
~/prefetch-simulation $ git checkout 0.0.2
```

This provides us with the code base used to derive the results presented in our work. We now compile and package the simulation software using Apache Maven:

```
~/prefetch-simulation $ cd prefetch-simulation
~/prefetch-simulation/prefetch-simulation $ mvn package
```

Note that the presented command fetches the entire dependency tree, which may take some time, depending on the Internet connection speed. Ultimately, the output of the `mvn` command should show a success message:

```
~/prefetch-simulation/prefetch-simulation $ mvn package
(... output left out ...)
[INFO] Reactor Summary:
[INFO]
[INFO] Prefetch Simulation ............................... SUCCESS [  0.005 s]
[INFO] Prefetch Simulation Components .................... SUCCESS [ 28.521 s]
[INFO] Configuration Materialiser ........................ SUCCESS [ 12.050 s]
[INFO] Simulation Runner ................................. SUCCESS [  0.553 s]
[INFO] Visualiser Components ............................. SUCCESS [  0.831 s]
[INFO] Regression Runner ................................. SUCCESS [  0.627 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 42.741 s
[INFO] Finished at: 2015-04-27T18:08:26+02:00
[INFO] Final Memory: 32M/231M
[INFO] ------------------------------------------------------------------------
~/prefetch-simulation/prefetch-simulation $
```

We can verify that runnable JAR files have been created:

```
~/prefetch-simulation/prefetch-simulation $ find . -name "*dependencies.jar"
./regression/target/regression-0.0.2-jar-with-dependencies.jar
./runner/target/runner-0.0.2-jar-with-dependencies.jar
./materialiser/target/materialiser-0.0.2-jar-with-dependencies.jar
./visualiser/target/visualiser-0.0.2-jar-with-dependencies.jar
```

With the environment set up and the framework built, our experiments can be run.

A.2 Exemplary Simulation

To run the exemplary simulation, the simulation framework must be set up and compiled according to section A.1. The simulation, which is in detail presented in section 6.2 is defined in the file `simulations/01_detail/configuration`:

```
seed 26031991001

ticks 3600
byterate u/1000/5000
slot-length 180
network-uptime 0.90
relative-jitter 0
absolute-jitter 0
relative-prediction-time-error 0
relative-prediction-amplitude-error ~/0/0.01
absolute-prediction-time-error ~/0/10
absolute-prediction-amplitude-error 0
look-ahead 3600

algorithm-parameter alpha 0.9

request-series interval u/400/600 size ~/500000/60000 byterate ~/3800/100 start 1000 end
    3000
request tick 3400 byterate 3600 data 480000
```

To execute the simulation using the three strategies (A, B and C, as described in section 6.1), we use the script `perform_abc`, as shown in the following:

```
~/prefetch-simulation $ cd simulations
~/prefetch-simulation/simulations $ ./perform_abc 01_detail
RT:          count    3 range [    115.00 ..     340.00] avg     200.67 median     147.00
DA:          count    3 range [    115.00 ..     340.00] avg     200.67 median     147.00
DV:          count    3 range [ 411404.00 ..  560520.00] avg  485924.67 median  485850.00
Hit Rate:    0 / 4 (0.0)
RT:          count    4 range [      0.00 ..       0.00] avg       0.00 median       0.00
DA:          count    4 range [    128.00 ..     174.00] avg     151.50 median     152.00
DV:          count    4 range [ 411404.00 ..  560520.00] avg  484443.50 median  482925.00
Hit Rate:    4 / 4 (100.0)
RT:          count    4 range [      0.00 ..       0.00] avg       0.00 median       0.00
DA:          count    4 range [    117.00 ..     173.00] avg     146.25 median     147.50
DV:          count    4 range [ 411404.00 ..  560520.00] avg  484443.50 median  482925.00
Hit Rate:    4 / 4 (100.0)
```

The directory `01_detail` now contains several files, instead of only the configuration file. Firstly, genesis files have been created for all three strategies, called `genesis_A`, `genesis_B` and `genesis_C`. In addition, a genesis PDF file has been created[2], along with the source TEX file, showing the graphical representation of the genesis. Furthermore, the results of the three strategies' simulations are stored in `result-summary_*.txt` files, where the asterisk ($\star$) is either A, B or C. The contents of these files is exactly the same as what is printed to the console during the simulation. Finally, the graphical representation of the results is saved to `result-timeline_*.pdf`, also along with the corresponding TEX source file.

Upon inspection, all of these files can be found to be the exact same results as presented in section 6.2. Also, see appendix C for a listing of checksum fixities to verify the integrity of the results.

A.3 Regression Simulations

The regression simulations can be run using the `regression` executable JAR, found in `regression/target/regression-0.0.2-jar-with-dependencies.jar` in the `prefetch-simulation` directory. The executable accepts zero or one parameters; if none are specified, all regression simulation experiments are run according to section 6.5. If a parameter is specified, it is used as the descriptor of the simulation to run (for example, the parameter `s1` will run the regression simulation for the variable S1 – the relative jitter.

[2]Since the only difference in genesis for the three strategies is the scheduling algorithm, which is not visible in the graphical representation, only one PDF file is created.

Note that executing a regression simulation experiment takes a rather large amount of time since a large amount of single simulations is performed. In the current implementation as of writing of this thesis, no multi-threading is performed.

An example for invoking a regression experiment is shown in the following, where the variable S4 – the look-ahead time t_{horizon} – is tested:

```
~/prefetch-simulation $ java -jar prefetch-simulation/regression/target/regression-0.0.2-
    jar-with-dependencies.jar s4 | tee s4.out
```

This command performs the experiments for variable S4 and saves them in the file `s4.out`. The file contains the results in CSV (comma-separated value) format, exactly the way they were used for this thesis. Furthermore, the results are listed numerically in appendix B, and in appendix C a listing of checksum fixities is provided to verify the integrity of the results.

APPENDIX B

Regression Analysis Results

This appendix shows the numeric results of our regression analysis simulations. Decimal numbers are rounded for typesetting purposes. Units for all $\mathcal{RT}$, and $\mathcal{DA}$ variables as well as t_{horizon} are $\mathcal{T}$, HR variables are ratios (their standard deviations being ratio points, in other words, their units are 100 percentage points). The measured hit rate for Strategy A (HR_A) and its standard deviation are not listed since those values are always zero by design (Strategy A performs no prefetching and thus cannot yield any cache hits).

The following tables show the results for each iteration of the sweep (independent) variable. Note that as described in section 6.5, for each variable value, 500 different iterations (with different seeds) have been performed and the mean values are shown in the tables below. The results include the basic metrics ($\mathcal{RT}$ and $\mathcal{DA}$ as well as the hit rate HR) for each strategy (A, B and C); their mean value and standard deviation (σ) are provided.

The last four columns, $t_{\mathcal{RT}_{AC}}$, $t_{\mathcal{RT}_{BC}}$, $t_{\mathcal{DA}_{AC}}$ and $t_{\mathcal{DA}_{BC}}$ show the *t-test result value* for $\mathcal{RT}$ and for $\mathcal{DA}$, when comparing the results of Strategy A with Strategy C, and those of Strategy B with Strategy C. The value determines whether there is a statistically significant difference between those values; the critical value for our simulation setup is **1.962**[1] – in all rows where the t-test result value is lower than this critical value, the respective values show no significant difference. In other words, values lower than the critical value determine that the two algorithms yield comparable performance.

For example, in section B.4, the thirds row shows the results for $t_{\text{horizon}} = 10\,\mathcal{T}$. $\mathcal{RT}$ shows values of $35.2\,\mathcal{T}$, $26.3\,\mathcal{T}$ and $26.3\,\mathcal{T}$, for A, B and C, respectively. The column $t_{\mathcal{RT}_{AC}}$ displays a value of 24.12 (higher than the critical value), indicating that Strategy A and Strategy C did not perform in a comparable manner regarding $\mathcal{RT}$ (Strategy C performed significantly better since its $\mathcal{RT}$ is lower). $t_{\mathcal{DA}_{AC}}$, however, only has a value of 0.15, indicating that regarding $\mathcal{DA}$, Strategy A and Strategy C were comparable. Looking

[1] Assuming $\alpha = 0.05$ and $\text{df} = 8$ with $n = 500$.

down the table, we see that in this case, the t values increase, thus the algorithms' performance diverges.

B.1 Relative Jitter (S1)

S1	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
0.00	35.1	5.8	7.0	4.9	0.6	1.7	28.4	3.1	30.0	3.3	31.9	3.2	0.604	0.11	0.984	0.03	128.16	27.68	17.87	9.10
0.04	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.1	3.4	31.9	3.2	0.598	0.11	0.983	0.03	128.39	27.94	17.03	8.44
0.08	35.2	5.8	7.2	4.9	0.6	1.7	28.6	3.2	30.2	3.4	31.9	3.2	0.595	0.11	0.982	0.03	128.95	28.23	16.54	8.04
0.12	35.4	5.7	7.2	4.9	0.6	1.8	28.7	3.1	30.3	3.4	31.9	3.2	0.589	0.11	0.973	0.04	129.21	28.26	16.05	7.62
0.16	35.5	5.7	7.3	4.9	0.7	1.9	28.9	3.2	30.4	3.4	32.0	3.2	0.584	0.11	0.953	0.05	128.53	27.87	15.48	7.41
0.20	35.7	5.8	7.4	4.9	0.8	1.9	29.1	3.2	30.6	3.4	32.1	3.2	0.575	0.12	0.923	0.06	128.53	27.84	14.55	7.14
0.24	35.9	5.8	7.6	4.9	1.0	2.0	29.4	3.2	30.8	3.4	32.3	3.2	0.561	0.11	0.885	0.07	128.04	27.51	14.10	7.08
0.28	36.2	5.8	7.8	5.0	1.3	2.1	29.7	3.3	31.0	3.4	32.5	3.3	0.546	0.12	0.846	0.08	126.83	26.75	13.44	7.03
0.32	36.6	5.8	8.1	5.0	1.7	2.2	30.1	3.3	31.3	3.5	32.8	3.3	0.528	0.12	0.803	0.09	126.06	26.21	12.75	6.83
0.36	37.0	5.9	8.4	5.0	2.1	2.4	30.5	3.4	31.7	3.6	33.1	3.4	0.506	0.11	0.762	0.10	123.76	25.37	12.12	6.36
0.40	37.4	5.9	8.7	5.1	2.5	2.5	31.0	3.6	32.1	3.7	33.5	3.5	0.486	0.12	0.724	0.10	121.86	24.29	11.42	6.04
0.44	37.9	6.0	9.1	5.2	3.1	2.7	31.4	3.7	32.6	3.9	34.0	3.6	0.466	0.11	0.686	0.10	119.15	23.16	11.13	5.98
0.48	38.5	6.0	9.5	5.3	3.6	2.8	31.9	3.7	33.0	3.9	34.4	3.6	0.445	0.11	0.654	0.11	116.97	22.20	10.78	6.01
0.52	39.0	6.1	10.0	5.3	4.1	3.0	32.4	3.8	33.4	4.1	34.8	3.7	0.427	0.11	0.622	0.11	115.04	21.41	10.32	5.87
0.56	39.5	6.1	10.5	5.4	4.7	3.2	32.9	4.0	33.8	4.2	35.3	3.7	0.409	0.11	0.596	0.11	112.73	20.45	9.75	5.68
0.60	40.1	6.2	10.9	5.4	5.3	3.3	33.4	4.2	34.3	4.4	35.7	3.9	0.391	0.11	0.573	0.11	110.05	19.78	8.96	5.38
0.64	40.6	6.3	11.4	5.5	6.0	3.6	33.9	4.3	34.7	4.5	36.1	4.0	0.377	0.11	0.552	0.11	106.75	18.65	8.55	5.37
0.68	41.2	6.4	11.9	5.5	6.6	3.8	34.3	4.5	35.1	4.6	36.5	4.1	0.363	0.11	0.532	0.11	104.52	17.89	8.06	5.35
0.72	41.7	6.5	12.4	5.6	7.2	3.9	34.8	4.8	35.5	4.7	36.9	4.2	0.351	0.11	0.515	0.11	102.03	17.08	7.44	5.16
0.76	42.2	6.6	12.8	5.6	7.7	4.0	35.2	4.9	35.9	4.8	37.3	4.3	0.341	0.11	0.499	0.11	100.48	16.77	7.11	4.88
0.80	42.6	6.6	13.3	5.7	8.2	4.0	35.6	4.9	36.2	4.9	37.6	4.4	0.330	0.11	0.485	0.11	99.26	16.33	6.87	4.66
0.84	43.1	6.7	13.7	5.7	8.6	4.1	36.0	5.1	36.6	5.0	37.9	4.5	0.321	0.11	0.473	0.11	98.27	16.01	6.44	4.50
0.88	43.6	6.8	14.1	5.8	9.1	4.2	36.3	5.2	36.8	5.1	38.2	4.5	0.313	0.10	0.460	0.11	96.84	15.75	6.26	4.47
0.92	44.0	6.8	14.5	5.8	9.6	4.4	36.6	5.3	37.1	5.2	38.6	4.7	0.306	0.10	0.449	0.11	94.86	15.04	6.31	4.79
0.96	44.5	6.9	15.0	5.9	10.1	4.5	36.9	5.4	37.4	5.3	38.9	4.8	0.299	0.10	0.439	0.11	93.63	14.78	6.19	4.76
1.00	44.9	7.0	15.4	6.0	10.5	4.5	37.3	5.5	37.7	5.3	39.2	4.9	0.292	0.10	0.431	0.11	92.96	14.73	5.94	4.76
1.04	45.3	7.0	15.8	6.0	10.9	4.6	37.6	5.6	38.0	5.5	39.6	5.0	0.287	0.10	0.426	0.11	91.40	14.40	5.81	4.62
1.08	45.7	7.1	16.2	6.1	11.3	4.7	37.9	5.7	38.3	5.5	39.8	5.1	0.281	0.10	0.419	0.11	90.27	14.14	5.68	4.52
1.12	46.1	7.2	16.5	6.2	11.7	4.7	38.1	5.8	38.6	5.6	40.1	5.2	0.275	0.10	0.410	0.10	89.04	14.00	5.61	4.46
1.16	46.4	7.3	16.9	6.3	12.0	4.8	38.4	5.9	38.8	5.7	40.3	5.3	0.269	0.10	0.405	0.11	88.20	13.77	5.42	4.19
1.20	46.7	7.3	17.2	6.3	12.4	4.9	38.6	6.0	39.1	5.8	40.5	5.3	0.266	0.10	0.399	0.11	87.53	13.51	5.26	4.10
1.24	47.1	7.4	17.5	6.4	12.7	4.9	38.9	6.1	39.3	5.9	40.7	5.4	0.262	0.09	0.395	0.10	86.71	13.43	5.12	4.07
1.28	47.4	7.4	17.8	6.5	13.0	5.0	39.1	6.2	39.6	6.0	41.0	5.5	0.258	0.09	0.390	0.10	85.76	13.29	5.04	3.83
1.32	47.7	7.5	18.1	6.5	13.3	5.1	39.4	6.2	39.8	6.1	41.2	5.5	0.254	0.09	0.385	0.10	84.96	13.09	4.84	3.67
1.36	48.0	7.6	18.4	6.5	13.6	5.1	39.6	6.4	40.1	6.1	41.4	5.5	0.251	0.09	0.381	0.10	84.23	12.96	4.82	3.52
1.40	48.2	7.6	18.7	6.6	13.9	5.2	39.8	6.5	40.3	6.2	41.7	5.6	0.249	0.09	0.377	0.10	83.39	12.77	4.72	3.55
1.44	48.5	7.6	18.9	6.6	14.2	5.2	40.1	6.6	40.5	6.1	41.8	5.7	0.247	0.09	0.374	0.10	82.91	12.60	4.62	3.53
1.48	48.7	7.7	19.2	6.7	14.4	5.3	40.3	6.7	40.8	6.2	42.0	5.7	0.244	0.09	0.371	0.10	82.24	12.53	4.44	3.39
1.52	49.0	7.7	19.4	6.7	14.7	5.3	40.5	6.7	40.9	6.3	42.2	5.8	0.242	0.09	0.368	0.10	81.71	12.41	4.42	3.41
1.56	49.2	7.8	19.6	6.8	14.9	5.4	40.7	6.7	41.2	6.3	42.4	5.9	0.241	0.09	0.365	0.10	81.18	12.25	4.34	3.34
1.60	49.5	7.8	19.9	6.8	15.2	5.4	40.9	6.9	41.3	6.4	42.7	6.0	0.240	0.09	0.361	0.10	80.60	12.12	4.22	3.39
1.64	49.7	7.8	20.1	6.9	15.4	5.5	41.1	7.0	41.5	6.5	42.9	6.1	0.238	0.09	0.358	0.10	80.35	12.03	4.18	3.35
1.68	49.9	7.9	20.4	6.9	15.7	5.6	41.4	7.0	41.8	6.6	43.1	6.1	0.236	0.09	0.355	0.10	79.47	11.88	4.10	3.21
1.72	50.1	7.9	20.6	6.9	15.9	5.6	41.5	7.1	42.0	6.7	43.2	6.2	0.234	0.09	0.353	0.10	79.11	11.87	4.07	3.10
1.76	50.3	8.0	20.8	6.9	16.1	5.6	41.7	7.1	42.1	6.7	43.4	6.3	0.233	0.09	0.350	0.10	78.48	11.84	3.93	3.00
1.80	50.5	8.0	21.0	7.0	16.3	5.7	41.9	7.2	42.3	6.8	43.5	6.3	0.231	0.09	0.347	0.10	77.94	11.75	3.83	2.90
1.84	50.7	8.1	21.2	7.0	16.5	5.7	42.1	7.3	42.5	6.9	43.7	6.4	0.229	0.09	0.344	0.10	77.62	11.70	3.79	2.85
1.88	50.9	8.1	21.4	7.0	16.7	5.7	42.2	7.3	42.7	7.0	43.8	6.4	0.228	0.09	0.342	0.10	77.17	11.62	3.71	2.78
1.92	51.1	8.1	21.6	7.1	16.8	5.8	42.4	7.4	42.8	7.0	44.0	6.4	0.226	0.09	0.339	0.10	76.88	11.52	3.70	2.81
1.96	51.3	8.1	21.7	7.1	17.0	5.8	42.4	7.4	42.9	7.1	44.1	6.5	0.225	0.09	0.337	0.10	76.87	11.47	3.75	2.82
2.00	51.4	8.2	21.9	7.1	17.2	5.8	42.6	7.5	43.0	7.1	44.2	6.5	0.224	0.09	0.335	0.10	76.07	11.20	3.67	2.85
2.04	51.6	8.2	22.0	7.2	17.4	5.9	42.7	7.5	43.1	7.1	44.3	6.5	0.222	0.09	0.334	0.10	75.88	11.12	3.59	2.79
2.08	51.8	8.2	22.2	7.2	17.6	5.9	42.9	7.5	43.2	7.2	44.4	6.6	0.221	0.09	0.332	0.10	75.24	10.95	3.49	2.71
2.12	51.9	8.3	22.4	7.3	17.8	6.0	43.0	7.6	43.4	7.2	44.5	6.6	0.221	0.09	0.330	0.10	74.75	10.93	3.36	2.61
2.16	52.1	8.3	22.6	7.3	17.9	6.0	43.2	7.7	43.5	7.2	44.6	6.7	0.219	0.09	0.329	0.10	74.50	10.92	3.20	2.57
2.20	52.3	8.3	22.7	7.3	18.1	6.0	43.3	7.7	43.6	7.3	44.7	6.8	0.219	0.09	0.327	0.10	74.32	10.88	3.21	2.60
2.24	52.4	8.4	22.8	7.3	18.2	6.1	43.4	7.7	43.7	7.3	44.9	6.8	0.218	0.09	0.326	0.10	74.06	10.80	3.25	2.64
2.28	52.6	8.5	23.0	7.3	18.4	6.1	43.4	7.7	43.8	7.4	45.0	6.8	0.217	0.09	0.325	0.10	73.38	10.73	3.31	2.61
2.32	52.7	8.5	23.1	7.3	18.5	6.1	43.6	7.7	43.9	7.4	45.1	6.9	0.216	0.09	0.323	0.10	73.13	10.70	3.21	2.53
2.36	52.9	8.5	23.2	7.4	18.7	6.1	43.7	7.7	44.0	7.4	45.1	6.9	0.215	0.09	0.321	0.10	72.94	10.72	3.19	2.47
2.40	53.0	8.5	23.4	7.4	18.8	6.1	43.8	7.8	44.1	7.5	45.2	6.9	0.213	0.09	0.320	0.10	72.93	10.68	3.11	2.41

2.44	53.2	8.6	23.5	7.4	18.9	6.1	43.9	7.8	44.2	7.5	45.3	6.9	0.212	0.09	0.319	0.10	72.76	10.72	3.09	2.38
2.48	53.3	8.6	23.6	7.4	19.0	6.2	43.9	7.8	44.3	7.6	45.4	7.0	0.211	0.09	0.317	0.10	72.61	10.70	3.08	2.33
2.52	53.4	8.6	23.7	7.4	19.1	6.2	44.1	7.8	44.4	7.6	45.5	7.0	0.211	0.09	0.316	0.10	72.46	10.64	3.02	2.34
2.56	53.5	8.6	23.9	7.5	19.3	6.2	44.2	7.8	44.5	7.6	45.5	7.0	0.210	0.09	0.316	0.10	72.16	10.55	2.90	2.32
2.60	53.6	8.6	24.0	7.5	19.4	6.2	44.3	7.9	44.6	7.7	45.6	7.0	0.209	0.09	0.315	0.10	72.08	10.55	2.90	2.15
2.64	53.8	8.6	24.1	7.5	19.5	6.2	44.4	7.9	44.7	7.7	45.7	7.1	0.209	0.09	0.314	0.10	71.86	10.53	2.78	2.09
2.68	53.9	8.7	24.2	7.5	19.6	6.2	44.5	7.9	44.8	7.8	45.8	7.1	0.209	0.09	0.313	0.10	71.60	10.50	2.80	2.13
2.72	54.0	8.7	24.3	7.5	19.8	6.2	44.6	7.9	44.9	7.8	45.8	7.1	0.208	0.09	0.313	0.10	71.40	10.52	2.68	1.99
2.76	54.1	8.7	24.5	7.5	19.9	6.3	44.7	8.0	45.0	7.8	45.9	7.1	0.208	0.09	0.312	0.10	71.18	10.48	2.52	1.87
2.80	54.2	8.7	24.6	7.5	20.0	6.3	44.8	8.0	45.1	7.8	46.0	7.2	0.208	0.09	0.311	0.10	70.92	10.43	2.59	1.89
2.84	54.3	8.8	24.6	7.5	20.1	6.4	44.8	8.0	45.2	7.9	46.1	7.2	0.207	0.09	0.310	0.10	70.66	10.30	2.58	1.79
2.88	54.4	8.8	24.7	7.5	20.2	6.4	44.9	8.0	45.3	7.9	46.2	7.2	0.206	0.09	0.310	0.10	70.30	10.26	2.66	1.89
2.92	54.5	8.8	24.8	7.5	20.3	6.4	45.0	8.0	45.3	7.9	46.3	7.3	0.206	0.09	0.309	0.10	70.22	10.27	2.63	1.98
2.96	54.6	8.8	24.9	7.5	20.4	6.4	45.1	8.1	45.4	8.0	46.3	7.2	0.205	0.09	0.309	0.10	70.07	10.22	2.57	2.00
3.00	54.7	8.8	25.0	7.5	20.5	6.5	45.2	8.0	45.4	7.9	46.4	7.3	0.204	0.09	0.307	0.10	69.88	10.11	2.66	2.06
3.04	54.8	8.9	25.1	7.6	20.6	6.5	45.2	8.1	45.6	8.0	46.5	7.3	0.204	0.09	0.306	0.10	69.40	10.06	2.70	1.95
3.08	54.9	8.9	25.2	7.6	20.7	6.5	45.3	8.1	45.6	8.0	46.6	7.3	0.204	0.09	0.306	0.10	69.28	10.04	2.67	1.93
3.12	55.0	8.9	25.3	7.6	20.8	6.6	45.3	8.1	45.7	8.0	46.6	7.3	0.203	0.09	0.305	0.10	69.13	10.03	2.65	1.82
3.16	55.0	8.9	25.3	7.6	20.8	6.6	45.4	8.2	45.8	8.1	46.7	7.3	0.203	0.09	0.305	0.10	69.04	10.03	2.59	1.89
3.20	55.1	8.9	25.4	7.6	20.9	6.6	45.5	8.2	45.8	8.1	46.7	7.3	0.203	0.09	0.305	0.10	68.81	9.94	2.59	1.82
3.24	55.2	8.9	25.5	7.6	21.0	6.6	45.5	8.2	45.9	8.1	46.8	7.3	0.203	0.09	0.304	0.10	68.55	9.94	2.61	1.83
3.28	55.3	8.9	25.6	7.6	21.1	6.7	45.6	8.2	46.0	8.1	46.8	7.3	0.202	0.09	0.303	0.10	68.49	9.90	2.58	1.81
3.32	55.3	9.0	25.7	7.7	21.2	6.7	45.6	8.2	46.0	8.1	46.9	7.3	0.202	0.09	0.303	0.10	68.38	9.92	2.65	1.82
3.36	55.4	9.0	25.8	7.7	21.3	6.7	45.6	8.2	46.1	8.1	47.0	7.3	0.202	0.09	0.302	0.10	68.24	9.90	2.70	1.83
3.40	55.5	9.0	25.9	7.7	21.3	6.7	45.7	8.2	46.1	8.1	47.0	7.3	0.202	0.09	0.302	0.10	68.24	9.92	2.59	1.86
3.44	55.6	8.9	25.9	7.7	21.4	6.7	45.7	8.2	46.1	8.1	47.0	7.3	0.202	0.09	0.301	0.10	68.34	9.93	2.56	1.78
3.48	55.6	8.9	26.0	7.7	21.5	6.7	45.8	8.3	46.2	8.1	47.1	7.4	0.202	0.09	0.301	0.10	68.37	9.90	2.57	1.83
3.52	55.7	9.0	26.1	7.7	21.5	6.7	45.8	8.3	46.2	8.1	47.1	7.4	0.202	0.09	0.300	0.10	68.13	9.87	2.60	1.89
3.56	55.8	9.0	26.1	7.7	21.6	6.8	45.9	8.4	46.3	8.2	47.2	7.4	0.202	0.09	0.299	0.10	67.74	9.78	2.57	1.92
3.60	55.8	9.0	26.2	7.7	21.7	6.8	46.0	8.4	46.3	8.2	47.2	7.4	0.202	0.09	0.299	0.10	67.67	9.79	2.51	1.89
3.64	55.9	9.0	26.2	7.7	21.8	6.8	46.0	8.4	46.3	8.2	47.2	7.4	0.201	0.09	0.298	0.10	67.40	9.72	2.45	1.86
3.68	56.0	9.0	26.3	7.7	21.8	6.8	46.1	8.4	46.4	8.2	47.3	7.4	0.201	0.09	0.298	0.10	67.53	9.74	2.46	1.83
3.72	56.0	9.0	26.3	7.7	21.9	6.8	46.1	8.4	46.4	8.2	47.3	7.4	0.201	0.09	0.297	0.10	67.57	9.72	2.54	1.82
3.76	56.1	9.0	26.4	7.7	21.9	6.8	46.1	8.4	46.5	8.2	47.4	7.4	0.200	0.09	0.296	0.10	67.46	9.71	2.48	1.79
3.80	56.2	9.1	26.5	7.7	22.0	6.8	46.2	8.4	46.5	8.2	47.4	7.5	0.200	0.09	0.296	0.10	67.34	9.69	2.50	1.78
3.84	56.2	9.1	26.5	7.7	22.0	6.8	46.2	8.4	46.6	8.2	47.4	7.5	0.200	0.09	0.296	0.10	67.36	9.72	2.48	1.71
3.88	56.3	9.1	26.6	7.7	22.1	6.9	46.2	8.4	46.6	8.2	47.5	7.5	0.200	0.09	0.295	0.10	67.27	9.76	2.46	1.67
3.92	56.4	9.1	26.7	7.7	22.1	6.9	46.2	8.4	46.7	8.2	47.5	7.5	0.199	0.09	0.294	0.10	67.20	9.75	2.54	1.72
3.96	56.4	9.1	26.7	7.7	22.2	6.9	46.3	8.4	46.7	8.3	47.6	7.5	0.199	0.09	0.294	0.10	67.02	9.72	2.57	1.65
4.00	56.5	9.1	26.8	7.7	22.3	6.9	46.3	8.4	46.8	8.4	47.6	7.5	0.199	0.09	0.293	0.10	66.95	9.67	2.52	1.63

B.2 Prediction Amplitude Error Deviation σ (S2)

S2	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
0.00	35.2	5.8	7.1	4.9	0.6	1.8	28.5	3.1	30.2	3.4	31.7	3.2	0.596	0.11	0.983	0.03	128.22	27.91	16.25	7.58
0.05	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.983	0.03	128.44	28.00	16.95	8.31
0.10	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.978	0.03	128.38	27.94	17.89	9.32
0.15	35.2	5.8	7.1	4.9	0.7	1.7	28.5	3.1	30.2	3.4	32.5	3.4	0.596	0.11	0.961	0.04	128.07	27.69	19.13	10.70
0.20	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	32.9	3.6	0.596	0.11	0.941	0.05	127.42	27.21	20.86	12.58
0.25	35.2	5.8	7.1	4.9	0.8	1.8	28.5	3.1	30.2	3.4	33.5	3.8	0.596	0.11	0.923	0.06	127.02	26.84	22.71	14.70
0.30	35.2	5.8	7.1	4.9	0.9	1.9	28.5	3.1	30.2	3.4	34.1	4.0	0.596	0.11	0.912	0.06	126.26	26.31	24.88	17.02
0.35	35.2	5.8	7.1	4.9	1.0	2.0	28.5	3.1	30.2	3.4	34.9	4.3	0.596	0.11	0.901	0.06	125.33	25.82	26.71	19.20
0.40	35.2	5.8	7.1	4.9	1.1	2.0	28.5	3.1	30.2	3.4	35.6	4.6	0.596	0.11	0.892	0.07	124.77	25.37	28.57	21.32
0.45	35.2	5.8	7.1	4.9	1.1	2.0	28.5	3.1	30.2	3.4	36.2	4.8	0.596	0.11	0.884	0.07	125.01	25.37	29.91	22.90
0.50	35.2	5.8	7.1	4.9	1.2	2.0	28.5	3.1	30.2	3.4	36.8	5.1	0.596	0.11	0.879	0.07	124.68	25.10	31.08	24.31
0.55	35.2	5.8	7.1	4.9	1.2	2.0	28.5	3.1	30.2	3.4	37.5	5.5	0.596	0.11	0.873	0.07	124.48	24.90	31.79	25.41
0.60	35.2	5.8	7.1	4.9	1.3	2.0	28.5	3.1	30.2	3.4	38.1	5.8	0.596	0.11	0.868	0.07	124.16	24.64	32.58	26.45
0.65	35.2	5.8	7.1	4.9	1.3	2.0	28.5	3.1	30.2	3.4	38.7	6.1	0.596	0.11	0.864	0.07	124.09	24.52	33.21	27.34
0.70	35.2	5.8	7.1	4.9	1.3	2.0	28.5	3.1	30.2	3.4	39.4	6.5	0.596	0.11	0.860	0.07	123.94	24.38	33.81	28.22
0.75	35.2	5.8	7.1	4.9	1.4	2.0	28.5	3.1	30.2	3.4	40.0	6.9	0.596	0.11	0.857	0.07	123.80	24.23	33.99	28.69
0.80	35.2	5.8	7.1	4.9	1.4	2.0	28.5	3.1	30.2	3.4	40.6	7.3	0.596	0.11	0.855	0.07	123.58	24.08	34.14	29.08
0.85	35.2	5.8	7.1	4.9	1.4	2.0	28.5	3.1	30.2	3.4	41.3	7.8	0.596	0.11	0.852	0.07	123.47	23.97	33.83	29.10
0.90	35.2	5.8	7.1	4.9	1.4	2.0	28.5	3.1	30.2	3.4	41.9	8.3	0.596	0.11	0.850	0.08	123.26	23.81	33.57	29.10
0.95	35.2	5.8	7.1	4.9	1.5	2.1	28.5	3.1	30.2	3.4	42.5	8.8	0.596	0.11	0.848	0.08	123.09	23.68	33.37	29.13
1.00	35.2	5.8	7.1	4.9	1.5	2.1	28.5	3.1	30.2	3.4	43.0	9.3	0.596	0.11	0.846	0.08	122.93	23.55	33.24	29.19
1.05	35.2	5.8	7.1	4.9	1.5	2.1	28.5	3.1	30.2	3.4	43.6	9.7	0.596	0.11	0.845	0.08	122.83	23.45	33.24	29.37
1.10	35.2	5.8	7.1	4.9	1.5	2.1	28.5	3.1	30.2	3.4	44.2	10.0	0.596	0.11	0.843	0.08	122.72	23.36	33.43	29.67
1.15	35.2	5.8	7.1	4.9	1.6	2.1	28.5	3.1	30.2	3.4	44.7	10.4	0.596	0.11	0.841	0.08	122.64	23.27	33.47	29.85

1.20	35.2	5.8	7.1	4.9	1.6	2.1	28.5	3.1	30.2	3.4	45.3	10.9	0.596	0.11	0.841	0.08	122.56	23.20	33.29	29.81
1.25	35.2	5.8	7.1	4.9	1.6	2.1	28.5	3.1	30.2	3.4	45.9	11.4	0.596	0.11	0.838	0.08	122.47	23.12	33.04	29.72
1.30	35.2	5.8	7.1	4.9	1.6	2.1	28.5	3.1	30.2	3.4	46.4	11.7	0.596	0.11	0.837	0.08	122.37	23.04	33.04	29.81
1.35	35.2	5.8	7.1	4.9	1.6	2.1	28.5	3.1	30.2	3.4	47.0	12.3	0.596	0.11	0.837	0.08	122.19	22.91	32.56	29.49
1.40	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	47.5	12.7	0.596	0.11	0.836	0.08	122.08	22.83	32.52	29.53
1.45	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	47.9	13.1	0.596	0.11	0.835	0.08	122.03	22.78	32.27	29.37
1.50	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	48.3	13.4	0.596	0.11	0.834	0.08	121.97	22.73	32.19	29.36
1.55	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	48.7	13.8	0.596	0.11	0.834	0.08	121.91	22.67	31.91	29.16
1.60	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	49.1	14.1	0.596	0.11	0.833	0.08	121.83	22.61	31.79	29.11
1.65	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	49.6	14.5	0.596	0.11	0.832	0.08	121.78	22.56	31.65	29.04
1.70	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	50.0	14.8	0.596	0.11	0.831	0.08	121.74	22.52	31.64	29.08
1.75	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	50.4	15.1	0.596	0.11	0.830	0.08	121.71	22.49	31.64	29.13
1.80	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	50.8	15.5	0.596	0.11	0.830	0.08	121.65	22.45	31.53	29.08
1.85	35.2	5.8	7.1	4.9	1.7	2.1	28.5	3.1	30.2	3.4	51.2	15.9	0.596	0.11	0.830	0.08	121.61	22.41	31.33	28.94
1.90	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	51.6	16.3	0.596	0.11	0.830	0.08	121.56	22.37	31.03	28.71
1.95	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	52.0	16.8	0.596	0.11	0.829	0.08	121.54	22.35	30.70	28.44
2.00	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	52.5	17.7	0.596	0.11	0.828	0.08	121.39	22.23	29.92	27.78
2.05	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	53.0	18.2	0.596	0.11	0.828	0.08	121.28	22.15	29.63	27.55
2.10	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	53.4	18.5	0.596	0.11	0.828	0.08	121.26	22.12	29.67	27.62
2.15	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	53.8	18.8	0.596	0.11	0.827	0.08	121.20	22.09	29.64	27.62
2.20	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	54.2	19.4	0.596	0.11	0.827	0.08	121.18	22.07	29.25	27.29
2.25	35.2	5.8	7.1	4.9	1.8	2.1	28.5	3.1	30.2	3.4	54.6	19.7	0.596	0.11	0.826	0.08	121.16	22.05	29.31	27.38
2.30	35.2	5.8	7.1	4.9	1.8	2.2	28.5	3.1	30.2	3.4	55.0	19.9	0.596	0.11	0.826	0.08	121.12	22.02	29.32	27.42
2.35	35.2	5.8	7.1	4.9	1.8	2.2	28.5	3.1	30.2	3.4	55.3	20.2	0.596	0.11	0.825	0.08	121.09	21.99	29.37	27.49
2.40	35.2	5.8	7.1	4.9	1.8	2.2	28.5	3.1	30.2	3.4	55.7	20.5	0.596	0.11	0.825	0.08	121.06	21.97	29.30	27.44
2.45	35.2	5.8	7.1	4.9	1.8	2.2	28.5	3.1	30.2	3.4	56.1	21.1	0.596	0.11	0.824	0.08	121.04	21.95	28.89	27.09
2.50	35.2	5.8	7.1	4.9	1.8	2.2	28.5	3.1	30.2	3.4	56.5	21.4	0.596	0.11	0.824	0.08	121.00	21.92	28.87	27.10
2.55	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	56.8	21.8	0.596	0.11	0.823	0.08	120.98	21.90	28.70	26.96
2.60	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	57.3	22.4	0.596	0.11	0.823	0.08	120.96	21.88	28.45	26.75
2.65	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	57.6	22.9	0.596	0.11	0.823	0.08	120.94	21.87	28.23	26.57
2.70	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	58.0	23.4	0.596	0.11	0.823	0.08	120.92	21.85	27.98	26.35
2.75	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	58.3	23.7	0.596	0.11	0.822	0.08	120.90	21.83	27.89	26.29
2.80	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	58.6	24.0	0.596	0.11	0.822	0.08	120.87	21.81	27.84	26.26
2.85	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	58.8	24.1	0.596	0.11	0.822	0.08	120.86	21.79	27.89	26.31
2.90	35.2	5.8	7.1	4.9	1.9	2.2	28.5	3.1	30.2	3.4	59.1	24.4	0.596	0.11	0.821	0.08	120.82	21.77	27.85	26.29
2.95	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	59.3	24.5	0.596	0.11	0.821	0.08	121.94	22.11	27.91	26.36
3.00	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	59.5	24.6	0.596	0.11	0.821	0.08	121.92	22.10	27.89	26.35
3.05	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	59.7	24.8	0.596	0.11	0.821	0.08	121.91	22.09	27.91	26.38
3.10	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	59.9	24.9	0.596	0.11	0.821	0.08	121.90	22.07	27.92	26.40
3.15	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	60.1	25.0	0.596	0.11	0.821	0.08	121.88	22.06	27.96	26.45
3.20	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	60.4	25.2	0.596	0.11	0.821	0.08	121.84	22.02	28.00	26.50
3.25	35.2	5.8	7.1	4.9	1.9	2.0	28.5	3.1	30.2	3.4	60.6	25.4	0.596	0.11	0.820	0.08	121.73	21.95	28.04	26.55
3.30	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	60.9	25.7	0.596	0.11	0.820	0.08	121.44	21.82	27.92	26.45
3.35	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	61.0	25.9	0.596	0.11	0.820	0.08	121.42	21.81	27.92	26.46
3.40	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	61.3	26.1	0.596	0.11	0.820	0.08	121.42	21.80	27.83	26.38
3.45	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	61.5	26.3	0.596	0.11	0.819	0.08	121.44	21.82	27.79	26.35
3.50	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	61.7	26.5	0.596	0.11	0.819	0.08	121.27	21.74	27.80	26.38
3.55	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	62.0	27.1	0.596	0.11	0.819	0.08	121.25	21.72	27.50	26.10
3.60	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	62.2	27.2	0.596	0.11	0.819	0.08	121.24	21.71	27.51	26.12
3.65	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	62.4	27.4	0.596	0.11	0.818	0.08	121.23	21.70	27.51	26.13
3.70	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	62.5	27.4	0.596	0.11	0.818	0.08	121.21	21.68	27.57	26.19
3.75	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	62.7	27.5	0.596	0.11	0.818	0.08	121.20	21.67	27.61	26.23
3.80	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	63.0	27.8	0.596	0.11	0.818	0.08	121.19	21.67	27.53	26.17
3.85	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	63.1	27.9	0.596	0.11	0.818	0.08	121.18	21.66	27.57	26.21
3.90	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	63.3	28.0	0.596	0.11	0.818	0.08	121.18	21.65	27.59	26.24
3.95	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	63.5	28.1	0.596	0.11	0.818	0.08	121.16	21.64	27.67	26.32
4.00	35.2	5.8	7.1	4.9	1.9	2.1	28.5	3.1	30.2	3.4	63.7	28.2	0.596	0.11	0.818	0.08	121.15	21.63	27.75	26.40

B.3 Prediction Time Error Deviation σ (S3)

S3	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
0.00	35.2	5.8	7.1	4.9	0.2	0.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.992	0.02	134.29	31.17	16.78	8.11
0.02	35.2	5.8	7.1	4.9	0.2	1.0	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.992	0.02	133.44	30.74	16.84	8.22
0.04	35.2	5.8	7.1	4.9	0.5	1.6	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.986	0.03	129.83	28.79	16.91	8.27
0.06	35.2	5.8	7.1	4.9	0.7	1.9	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.09	27.24	17.05	8.41
0.08	35.2	5.8	7.1	4.9	0.9	2.0	28.5	3.1	30.2	3.4	32.0	3.2	0.596	0.11	0.976	0.04	125.50	26.30	17.23	8.61
0.10	35.2	5.8	7.1	4.9	1.0	2.2	28.5	3.1	30.2	3.4	32.0	3.2	0.596	0.11	0.972	0.04	123.78	25.29	17.32	8.70
0.12	35.2	5.8	7.1	4.9	1.1	2.3	28.5	3.1	30.2	3.4	32.0	3.3	0.596	0.11	0.967	0.04	122.22	24.50	17.37	8.79

0.14	35.2	5.8	7.1	4.9	1.2	2.4	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.963	0.04	121.19	23.92	17.50	8.94
0.16	35.2	5.8	7.1	4.9	1.4	2.5	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.959	0.04	120.05	23.29	17.55	9.03
0.18	35.2	5.8	7.1	4.9	1.5	2.6	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.955	0.05	119.07	22.68	17.73	9.19
0.20	35.2	5.8	7.1	4.9	1.5	2.6	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.952	0.05	118.67	22.43	17.93	9.41
0.22	35.2	5.8	7.1	4.9	1.6	2.7	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.949	0.05	118.13	22.11	18.20	9.66
0.24	35.2	5.8	7.1	4.9	1.6	2.7	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.945	0.05	117.81	21.90	18.30	9.78
0.26	35.2	5.8	7.1	4.9	1.7	2.8	28.5	3.1	30.2	3.4	32.3	3.3	0.596	0.11	0.940	0.05	116.95	21.40	18.49	9.97
0.28	35.2	5.8	7.1	4.9	1.8	2.8	28.5	3.1	30.2	3.4	32.3	3.3	0.596	0.11	0.936	0.05	116.49	21.09	18.73	10.21
0.30	35.2	5.8	7.1	4.9	1.9	2.9	28.5	3.1	30.2	3.4	32.4	3.4	0.596	0.11	0.933	0.05	115.80	20.69	18.87	10.38
0.32	35.2	5.8	7.1	4.9	1.9	2.9	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.929	0.06	115.26	20.33	19.26	10.74
0.34	35.2	5.8	7.1	4.9	2.0	3.0	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.926	0.06	114.21	19.84	19.45	10.91
0.36	35.2	5.8	7.1	4.9	2.1	3.1	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.922	0.06	113.40	19.46	19.58	11.00
0.38	35.2	5.8	7.1	4.9	2.1	3.1	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.920	0.06	113.28	19.36	19.79	11.18
0.40	35.2	5.8	7.1	4.9	2.2	3.1	28.5	3.1	30.2	3.4	32.6	3.3	0.596	0.11	0.917	0.06	112.84	19.03	20.15	11.57
0.42	35.2	5.8	7.1	4.9	2.2	3.2	28.5	3.1	30.2	3.4	32.7	3.3	0.596	0.11	0.914	0.06	112.00	18.61	20.40	11.80
0.44	35.2	5.8	7.1	4.9	2.3	3.2	28.5	3.1	30.2	3.4	32.7	3.3	0.596	0.11	0.911	0.06	111.43	18.25	20.56	11.96
0.46	35.2	5.8	7.1	4.9	2.4	3.2	28.5	3.1	30.2	3.4	32.7	3.4	0.596	0.11	0.908	0.06	111.24	18.12	20.58	12.06
0.48	35.2	5.8	7.1	4.9	2.4	3.2	28.5	3.1	30.2	3.4	32.8	3.4	0.596	0.11	0.905	0.06	110.88	17.87	20.70	12.23
0.50	35.2	5.8	7.1	4.9	2.5	3.2	28.5	3.1	30.2	3.4	32.8	3.4	0.596	0.11	0.903	0.06	110.75	17.72	20.95	12.46
0.52	35.2	5.8	7.1	4.9	2.6	3.3	28.5	3.1	30.2	3.4	32.9	3.4	0.596	0.11	0.900	0.07	109.55	17.17	21.10	12.60
0.54	35.2	5.8	7.1	4.9	2.6	3.3	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.898	0.07	109.40	17.04	21.32	12.89
0.56	35.2	5.8	7.1	4.9	2.6	3.4	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.898	0.07	109.02	16.81	21.39	12.98
0.58	35.2	5.8	7.1	4.9	2.7	3.4	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.895	0.07	108.28	16.43	21.47	13.05
0.60	35.2	5.8	7.1	4.9	2.8	3.5	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.892	0.07	107.73	16.07	21.67	13.27
0.62	35.2	5.8	7.1	4.9	2.8	3.5	28.5	3.1	30.2	3.4	33.1	3.5	0.596	0.11	0.891	0.07	107.47	15.89	21.80	13.35
0.64	35.2	5.8	7.1	4.9	2.8	3.5	28.5	3.1	30.2	3.4	33.1	3.5	0.596	0.11	0.888	0.07	107.19	15.80	21.83	13.42
0.66	35.2	5.8	7.1	4.9	2.9	3.5	28.5	3.1	30.2	3.4	33.1	3.5	0.596	0.11	0.887	0.07	106.69	15.61	22.05	13.65
0.68	35.2	5.8	7.1	4.9	2.9	3.6	28.5	3.1	30.2	3.4	33.2	3.5	0.596	0.11	0.885	0.07	106.14	15.36	22.56	14.13
0.70	35.2	5.8	7.1	4.9	3.0	3.6	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.882	0.07	105.84	15.12	22.89	14.47
0.72	35.2	5.8	7.1	4.9	3.1	3.6	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.879	0.07	105.34	14.80	22.91	14.45
0.74	35.2	5.8	7.1	4.9	3.1	3.6	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.877	0.07	105.20	14.62	23.01	14.61
0.76	35.2	5.8	7.1	4.9	3.2	3.7	28.5	3.1	30.2	3.4	33.4	3.6	0.596	0.11	0.875	0.07	104.79	14.37	23.24	14.90
0.78	35.2	5.8	7.1	4.9	3.2	3.7	28.5	3.1	30.2	3.4	33.4	3.5	0.596	0.11	0.873	0.08	104.50	14.26	23.24	14.87
0.80	35.2	5.8	7.1	4.9	3.2	3.7	28.5	3.1	30.2	3.4	33.5	3.5	0.596	0.11	0.872	0.08	104.04	14.06	23.45	15.06
0.82	35.2	5.8	7.1	4.9	3.3	3.7	28.5	3.1	30.2	3.4	33.5	3.5	0.596	0.11	0.871	0.08	104.17	14.01	23.57	15.20
0.84	35.2	5.8	7.1	4.9	3.3	3.7	28.5	3.1	30.2	3.4	33.5	3.6	0.596	0.11	0.868	0.08	104.17	13.99	23.67	15.30
0.86	35.2	5.8	7.1	4.9	3.3	3.7	28.5	3.1	30.2	3.4	33.5	3.6	0.596	0.11	0.867	0.08	103.92	13.78	23.63	15.26
0.88	35.2	5.8	7.1	4.9	3.3	3.8	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.865	0.08	103.42	13.61	23.90	15.53
0.90	35.2	5.8	7.1	4.9	3.4	3.8	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.864	0.08	103.13	13.48	23.98	15.62
0.92	35.2	5.8	7.1	4.9	3.4	3.8	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.863	0.08	102.51	13.31	23.99	15.66
0.94	35.2	5.8	7.1	4.9	3.4	3.8	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.861	0.08	102.49	13.21	23.97	15.63
0.96	35.2	5.8	7.1	4.9	3.5	3.8	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.859	0.08	102.39	13.11	24.08	15.74
0.98	35.2	5.8	7.1	4.9	3.5	3.9	28.5	3.1	30.2	3.4	33.6	3.6	0.596	0.11	0.858	0.08	101.88	12.95	24.05	15.74
1.00	35.2	5.8	7.1	4.9	3.5	3.9	28.5	3.1	30.2	3.4	33.7	3.6	0.596	0.11	0.857	0.08	101.80	12.81	24.23	15.91

B.4 Look-Ahead Time t_{horizon} (S4)

S4	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
0	35.2	5.8	35.2	5.8	35.2	5.8	28.5	3.1	28.5	3.1	28.5	3.1	0.000	0.00	0.000	0.00	0.00	0.00	0.00	0.00
5	35.2	5.8	31.2	5.8	31.2	5.8	28.5	3.1	28.5	3.1	28.5	3.1	0.000	0.00	0.000	0.00	10.91	0.00	0.08	0.00
10	35.2	5.8	26.3	5.9	26.3	5.9	28.5	3.1	28.5	3.1	28.5	3.1	0.000	0.00	0.000	0.00	24.12	0.00	0.15	0.00
15	35.2	5.8	21.3	5.8	21.3	5.8	28.5	3.1	28.5	3.1	28.5	3.1	0.000	0.00	0.000	0.00	37.97	0.00	0.15	0.00
20	35.2	5.8	16.6	5.8	16.3	5.8	28.5	3.1	28.4	3.1	28.4	3.1	0.098	0.07	0.123	0.07	51.67	0.73	0.32	0.01
25	35.2	5.8	13.6	5.6	12.4	5.5	28.5	3.1	28.5	3.0	28.5	3.0	0.229	0.09	0.357	0.10	63.84	3.51	0.10	0.03
30	35.2	5.8	11.7	5.5	9.6	5.4	28.5	3.1	29.2	2.6	29.7	2.1	0.325	0.10	0.548	0.10	72.60	6.12	7.17	3.39
35	35.2	5.8	10.3	5.3	7.6	5.1	28.5	3.1	30.0	3.0	31.4	2.5	0.394	0.11	0.680	0.10	80.41	8.22	16.34	8.37
40	35.2	5.8	9.3	5.2	6.2	4.9	28.5	3.1	30.2	3.3	31.9	3.1	0.442	0.11	0.762	0.09	85.72	9.70	17.01	8.35
45	35.2	5.8	8.5	5.1	5.2	4.6	28.5	3.1	30.2	3.4	31.9	3.2	0.482	0.11	0.819	0.09	90.89	10.91	16.95	8.31
50	35.2	5.8	8.0	5.0	4.5	4.4	28.5	3.1	30.2	3.4	31.9	3.2	0.516	0.11	0.862	0.08	94.66	11.85	16.95	8.31
55	35.2	5.8	7.6	5.0	3.8	4.1	28.5	3.1	30.2	3.4	31.9	3.2	0.538	0.11	0.888	0.07	98.83	13.08	16.95	8.31
60	35.2	5.8	7.3	4.9	3.5	4.0	28.5	3.1	30.2	3.4	31.9	3.2	0.561	0.11	0.912	0.07	100.99	13.63	16.95	8.31
65	35.2	5.8	7.2	4.9	3.1	3.8	28.5	3.1	30.2	3.4	31.9	3.2	0.580	0.11	0.933	0.06	103.59	14.55	16.95	8.31
70	35.2	5.8	7.1	4.9	2.9	3.7	28.5	3.1	30.2	3.4	31.9	3.2	0.591	0.11	0.946	0.05	105.16	15.33	16.95	8.31
75	35.2	5.8	7.1	4.9	2.7	3.5	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.953	0.05	107.62	16.41	16.95	8.31
80	35.2	5.8	7.1	4.9	2.5	3.3	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.955	0.05	109.86	17.47	16.95	8.31
85	35.2	5.8	7.1	4.9	2.3	3.1	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.956	0.04	112.14	18.39	16.95	8.31
90	35.2	5.8	7.1	4.9	2.2	3.0	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.957	0.04	113.56	19.17	16.95	8.31
95	35.2	5.8	7.1	4.9	2.0	2.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.959	0.04	115.47	20.10	16.95	8.31

100	35.2	5.8	7.1	4.9	1.9	2.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.960	0.04	116.45	20.68	16.95	8.31
105	35.2	5.8	7.1	4.9	1.8	2.6	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.961	0.04	117.62	21.31	16.95	8.31
110	35.2	5.8	7.1	4.9	1.7	2.5	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.963	0.04	118.81	21.99	16.95	8.31
115	35.2	5.8	7.1	4.9	1.5	2.5	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.965	0.04	119.80	22.62	16.95	8.31
120	35.2	5.8	7.1	4.9	1.4	2.4	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.966	0.04	120.60	23.14	16.95	8.31
125	35.2	5.8	7.1	4.9	1.4	2.4	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.967	0.04	121.30	23.58	16.95	8.31
130	35.2	5.8	7.1	4.9	1.3	2.3	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.969	0.04	122.14	24.14	16.95	8.31
135	35.2	5.8	7.1	4.9	1.2	2.3	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.970	0.04	122.64	24.44	16.95	8.31
140	35.2	5.8	7.1	4.9	1.1	2.2	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.971	0.04	123.09	24.72	16.95	8.31
145	35.2	5.8	7.1	4.9	1.1	2.2	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.972	0.04	123.90	25.23	16.95	8.31
150	35.2	5.8	7.1	4.9	1.0	2.1	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.974	0.04	124.32	25.50	16.95	8.31
155	35.2	5.8	7.1	4.9	1.0	2.1	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.975	0.04	124.77	25.78	16.95	8.31
160	35.2	5.8	7.1	4.9	0.9	2.0	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.976	0.03	125.26	26.07	16.95	8.31
165	35.2	5.8	7.1	4.9	0.9	2.0	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.976	0.03	125.78	26.36	16.95	8.31
170	35.2	5.8	7.1	4.9	0.8	2.0	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.977	0.03	126.01	26.51	16.95	8.31
175	35.2	5.8	7.1	4.9	0.8	1.9	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.978	0.03	126.29	26.69	16.95	8.31
180	35.2	5.8	7.1	4.9	0.8	1.9	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.979	0.03	126.50	26.83	16.95	8.31
185	35.2	5.8	7.1	4.9	0.8	1.9	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.980	0.03	126.84	27.02	16.95	8.31
190	35.2	5.8	7.1	4.9	0.7	1.9	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.980	0.03	127.13	27.20	16.95	8.31
195	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.980	0.03	127.27	27.29	16.95	8.31
200	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.32	27.32	16.95	8.31
205	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.36	27.35	16.95	8.31
210	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.39	27.37	16.95	8.31
215	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.50	27.42	16.95	8.31
220	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.57	27.46	16.95	8.31
225	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.981	0.03	127.64	27.51	16.95	8.31
230	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.982	0.03	127.70	27.53	16.95	8.31
235	35.2	5.8	7.1	4.9	0.6	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.982	0.03	127.93	27.67	16.95	8.31
240	35.2	5.8	7.1	4.9	0.6	1.8	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.982	0.03	128.12	27.76	16.95	8.31
245	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.982	0.03	128.14	27.78	16.95	8.31
250	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.982	0.03	128.17	27.79	16.95	8.31

B.5 Error Correction Factor α (S5)

S5	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
0.24	35.2	5.8	1.7	2.6	0.0	0.3	28.5	3.1	81.3	12.9	82.9	7.9	0.968	0.04	1.000	0.00	136.09	14.87	143.12	2.30
0.26	35.2	5.8	1.9	2.7	0.0	0.3	28.5	3.1	75.6	12.1	77.0	7.4	0.964	0.04	0.999	0.01	136.08	15.65	135.53	2.23
0.28	35.2	5.8	2.2	3.1	0.0	0.3	28.5	3.1	70.8	11.4	72.1	6.9	0.961	0.04	0.999	0.01	136.04	15.26	129.19	2.23
0.30	35.2	5.8	2.3	3.3	0.1	0.6	28.5	3.1	66.5	10.7	67.8	6.3	0.956	0.04	0.999	0.01	135.52	15.34	124.93	2.26
0.32	35.2	5.8	2.4	3.3	0.1	0.6	28.5	3.1	62.7	10.1	64.3	5.9	0.953	0.05	0.998	0.01	135.34	15.53	119.04	3.09
0.34	35.2	5.8	2.6	3.5	0.1	0.6	28.5	3.1	59.5	9.5	61.2	5.7	0.947	0.05	0.998	0.01	135.33	16.23	112.79	3.35
0.36	35.2	5.8	2.8	3.5	0.1	0.7	28.5	3.1	56.6	9.0	58.5	5.5	0.941	0.05	0.998	0.01	135.05	16.80	105.88	3.92
0.38	35.2	5.8	2.9	3.6	0.1	1.0	28.5	3.1	53.9	8.6	56.0	5.3	0.933	0.05	0.997	0.01	134.01	16.80	99.71	4.73
0.40	35.2	5.8	3.1	3.7	0.1	1.0	28.5	3.1	51.6	8.1	53.7	5.1	0.925	0.06	0.997	0.01	133.80	17.26	94.45	4.93
0.42	35.2	5.8	3.3	3.8	0.2	1.0	28.5	3.1	49.5	7.7	51.8	4.9	0.917	0.06	0.997	0.01	133.59	17.42	89.99	5.77
0.44	35.2	5.8	3.4	3.9	0.2	1.1	28.5	3.1	47.6	7.4	50.1	4.7	0.909	0.06	0.996	0.01	133.38	17.90	84.98	6.24
0.46	35.2	5.8	3.6	4.0	0.2	1.1	28.5	3.1	45.9	7.0	48.5	4.6	0.900	0.07	0.996	0.01	133.37	18.39	80.50	6.80
0.48	35.2	5.8	3.7	4.1	0.2	1.1	28.5	3.1	44.5	6.7	47.0	4.4	0.890	0.07	0.996	0.01	133.19	18.68	76.25	7.01
0.50	35.2	5.8	3.9	4.2	0.2	1.1	28.5	3.1	43.1	6.4	45.5	4.3	0.878	0.07	0.996	0.01	133.17	19.15	70.98	7.17
0.52	35.2	5.8	4.0	4.3	0.2	1.1	28.5	3.1	41.9	6.1	44.5	4.3	0.867	0.08	0.996	0.01	133.13	19.54	67.24	7.81
0.54	35.2	5.8	4.2	4.4	0.2	1.1	28.5	3.1	40.6	5.8	43.4	4.2	0.855	0.08	0.995	0.02	132.91	19.85	63.21	8.52
0.56	35.2	5.8	4.4	4.4	0.2	1.2	28.5	3.1	39.6	5.6	42.3	4.1	0.844	0.08	0.995	0.02	132.62	20.43	59.60	8.81
0.58	35.2	5.8	4.6	4.5	0.3	1.2	28.5	3.1	38.6	5.3	41.4	4.0	0.831	0.08	0.994	0.02	132.58	20.89	56.24	9.29
0.60	35.2	5.8	4.7	4.5	0.3	1.3	28.5	3.1	37.7	5.2	40.4	3.9	0.817	0.09	0.994	0.02	132.12	21.26	52.70	9.08
0.62	35.2	5.8	4.9	4.6	0.3	1.3	28.5	3.1	36.9	5.0	39.6	3.9	0.806	0.09	0.993	0.02	131.88	21.76	49.62	9.47
0.64	35.2	5.8	5.1	4.6	0.3	1.3	28.5	3.1	36.1	4.8	38.8	3.8	0.790	0.09	0.993	0.02	131.85	22.38	46.88	9.66
0.66	35.2	5.8	5.2	4.6	0.3	1.3	28.5	3.1	35.4	4.7	38.0	3.7	0.777	0.09	0.992	0.02	131.82	22.75	43.94	9.67
0.68	35.2	5.8	5.4	4.7	0.3	1.3	28.5	3.1	34.8	4.5	37.3	3.6	0.762	0.10	0.992	0.02	131.79	23.28	41.12	9.68
0.70	35.2	5.8	5.5	4.7	0.4	1.5	28.5	3.1	34.2	4.3	36.7	3.6	0.747	0.10	0.991	0.02	130.81	23.50	38.50	9.76
0.72	35.2	5.8	5.7	4.7	0.4	1.5	28.5	3.1	33.7	4.2	36.0	3.6	0.732	0.10	0.991	0.02	130.76	24.05	35.48	9.55
0.74	35.2	5.8	5.9	4.7	0.4	1.6	28.5	3.1	33.1	4.1	35.4	3.6	0.716	0.10	0.991	0.02	130.01	24.39	32.62	9.47
0.76	35.2	5.8	6.0	4.8	0.4	1.6	28.5	3.1	32.7	4.0	34.9	3.5	0.703	0.10	0.990	0.02	129.84	24.78	30.32	9.44
0.78	35.2	5.8	6.2	4.8	0.4	1.6	28.5	3.1	32.2	3.9	34.4	3.5	0.689	0.11	0.989	0.02	129.79	25.34	28.31	9.35
0.80	35.2	5.8	6.4	4.8	0.5	1.6	28.5	3.1	31.8	3.8	33.9	3.4	0.672	0.11	0.989	0.02	129.67	25.98	25.94	9.09
0.82	35.2	5.8	6.5	4.8	0.5	1.6	28.5	3.1	31.4	3.7	33.5	3.4	0.657	0.11	0.988	0.02	129.42	26.36	24.04	9.21
0.84	35.2	5.8	6.7	4.9	0.5	1.6	28.5	3.1	31.1	3.6	33.0	3.3	0.642	0.11	0.987	0.03	129.37	26.99	22.25	8.89
0.86	35.2	5.8	6.8	4.9	0.5	1.6	28.5	3.1	30.8	3.5	32.6	3.3	0.625	0.11	0.986	0.03	129.22	27.48	20.47	8.79
0.88	35.2	5.8	7.0	4.9	0.6	1.7	28.5	3.1	30.4	3.4	32.2	3.2	0.610	0.11	0.985	0.03	128.86	27.76	18.62	8.52

0.90	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.983	0.03	128.44	28.00	16.95	8.31
0.92	35.2	5.8	7.3	4.9	0.6	1.7	28.5	3.1	29.9	3.3	31.6	3.2	0.580	0.11	0.982	0.03	128.21	28.56	15.37	8.26
0.94	35.2	5.8	7.4	4.9	0.6	1.8	28.5	3.1	29.6	3.2	31.3	3.1	0.566	0.11	0.979	0.03	127.94	28.97	13.95	8.23
0.96	35.2	5.8	7.6	5.0	0.7	1.8	28.5	3.1	29.3	3.2	31.0	3.1	0.554	0.11	0.976	0.03	127.57	29.15	12.47	8.19
0.98	35.2	5.8	7.7	5.0	0.7	1.9	28.5	3.1	29.1	3.2	30.7	3.1	0.542	0.11	0.971	0.04	126.81	29.23	11.06	7.90
1.00	35.2	5.8	7.8	5.0	0.8	1.9	28.5	3.1	28.9	3.1	30.4	3.1	0.525	0.11	0.957	0.05	126.55	29.62	9.58	7.79

B.6 Prediction Amplitude Error Mean μ (S7)

S7	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
−0.72	35.2	5.8	7.1	4.9	0.0	0.3	28.5	3.1	30.2	3.4	81.6	8.4	0.596	0.11	0.999	0.01	136.01	32.17	133.06	127.76
−0.70	35.2	5.8	7.1	4.9	0.0	0.4	28.5	3.1	30.2	3.4	76.4	7.6	0.596	0.11	0.999	0.01	135.96	32.13	130.23	124.41
−0.68	35.2	5.8	7.1	4.9	0.0	0.4	28.5	3.1	30.2	3.4	71.9	7.1	0.596	0.11	0.999	0.01	135.95	32.12	124.65	118.49
−0.66	35.2	5.8	7.1	4.9	0.0	0.4	28.5	3.1	30.2	3.4	68.0	6.6	0.596	0.11	0.999	0.01	135.83	32.05	120.63	114.05
−0.64	35.2	5.8	7.1	4.9	0.1	0.5	28.5	3.1	30.2	3.4	64.8	6.3	0.596	0.11	0.998	0.01	135.50	31.88	115.37	108.55
−0.62	35.2	5.8	7.1	4.9	0.1	0.7	28.5	3.1	30.2	3.4	61.9	5.9	0.596	0.11	0.998	0.01	135.06	31.67	111.75	104.54
−0.60	35.2	5.8	7.1	4.9	0.1	0.7	28.5	3.1	30.2	3.4	59.3	5.6	0.596	0.11	0.998	0.01	135.04	31.66	107.45	99.94
−0.58	35.2	5.8	7.1	4.9	0.1	0.7	28.5	3.1	30.2	3.4	57.0	5.5	0.596	0.11	0.998	0.01	134.95	31.60	101.09	93.52
−0.56	35.2	5.8	7.1	4.9	0.1	0.9	28.5	3.1	30.2	3.4	54.9	5.3	0.596	0.11	0.998	0.01	134.21	31.28	95.98	88.28
−0.54	35.2	5.8	7.1	4.9	0.1	0.9	28.5	3.1	30.2	3.4	53.0	5.1	0.596	0.11	0.997	0.01	134.02	31.16	91.50	83.64
−0.52	35.2	5.8	7.1	4.9	0.2	1.0	28.5	3.1	30.2	3.4	51.2	4.9	0.596	0.11	0.996	0.01	133.56	30.92	87.35	79.31
−0.50	35.2	5.8	7.1	4.9	0.2	1.0	28.5	3.1	30.2	3.4	49.6	4.7	0.596	0.11	0.996	0.01	133.55	30.91	83.68	75.43
−0.48	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	48.2	4.5	0.596	0.11	0.996	0.01	133.36	30.81	79.81	71.44
−0.46	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	46.9	4.5	0.596	0.11	0.996	0.01	133.19	30.70	74.98	66.63
−0.44	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	45.7	4.4	0.596	0.11	0.996	0.01	133.17	30.70	71.07	62.70
−0.42	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	44.6	4.4	0.596	0.11	0.996	0.01	133.14	30.67	67.07	58.71
−0.40	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	43.6	4.3	0.596	0.11	0.995	0.02	133.09	30.63	63.72	55.29
−0.38	35.2	5.8	7.1	4.9	0.2	1.2	28.5	3.1	30.2	3.4	42.6	4.2	0.596	0.11	0.995	0.02	132.65	30.39	60.31	51.86
−0.36	35.2	5.8	7.1	4.9	0.3	1.2	28.5	3.1	30.2	3.4	41.7	4.1	0.596	0.11	0.994	0.02	132.38	30.22	57.33	48.81
−0.34	35.2	5.8	7.1	4.9	0.3	1.2	28.5	3.1	30.2	3.4	40.9	4.0	0.596	0.11	0.994	0.02	132.37	30.21	54.22	45.68
−0.32	35.2	5.8	7.1	4.9	0.3	1.3	28.5	3.1	30.2	3.4	40.1	3.9	0.596	0.11	0.994	0.02	132.10	30.05	51.44	42.83
−0.30	35.2	5.8	7.1	4.9	0.3	1.3	28.5	3.1	30.2	3.4	39.3	3.9	0.596	0.11	0.993	0.02	131.96	29.97	48.64	40.00
−0.28	35.2	5.8	7.1	4.9	0.3	1.3	28.5	3.1	30.2	3.4	38.6	3.8	0.596	0.11	0.993	0.02	131.82	29.88	46.23	37.51
−0.26	35.2	5.8	7.1	4.9	0.3	1.3	28.5	3.1	30.2	3.4	37.9	3.7	0.596	0.11	0.992	0.02	131.82	29.87	43.52	34.79
−0.24	35.2	5.8	7.1	4.9	0.4	1.4	28.5	3.1	30.2	3.4	37.3	3.6	0.596	0.11	0.992	0.02	131.08	29.53	40.79	32.07
−0.22	35.2	5.8	7.1	4.9	0.4	1.5	28.5	3.1	30.2	3.4	36.7	3.6	0.596	0.11	0.991	0.02	130.81	29.39	38.21	29.50
−0.20	35.2	5.8	7.1	4.9	0.4	1.5	28.5	3.1	30.2	3.4	36.1	3.5	0.596	0.11	0.991	0.02	130.68	29.31	35.87	27.14
−0.18	35.2	5.8	7.1	4.9	0.4	1.6	28.5	3.1	30.2	3.4	35.6	3.6	0.596	0.11	0.990	0.02	130.00	28.98	33.27	24.63
−0.16	35.2	5.8	7.1	4.9	0.4	1.6	28.5	3.1	30.2	3.4	35.1	3.5	0.596	0.11	0.990	0.02	129.85	28.90	30.97	22.38
−0.14	35.2	5.8	7.1	4.9	0.4	1.6	28.5	3.1	30.2	3.4	34.6	3.5	0.596	0.11	0.990	0.02	129.82	28.88	28.97	20.36
−0.12	35.2	5.8	7.1	4.9	0.5	1.6	28.5	3.1	30.2	3.4	34.1	3.4	0.596	0.11	0.989	0.02	129.69	28.79	27.08	18.46
−0.10	35.2	5.8	7.1	4.9	0.5	1.6	28.5	3.1	30.2	3.4	33.7	3.4	0.596	0.11	0.988	0.02	129.49	28.65	25.04	16.43
−0.08	35.2	5.8	7.1	4.9	0.5	1.6	28.5	3.1	30.2	3.4	33.3	3.3	0.596	0.11	0.988	0.02	129.39	28.58	23.57	14.87
−0.06	35.2	5.8	7.1	4.9	0.5	1.6	28.5	3.1	30.2	3.4	32.9	3.3	0.596	0.11	0.987	0.03	129.36	28.55	21.70	13.06
−0.04	35.2	5.8	7.1	4.9	0.5	1.7	28.5	3.1	30.2	3.4	32.6	3.2	0.596	0.11	0.986	0.03	129.06	28.36	20.21	11.52
−0.02	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	32.2	3.2	0.596	0.11	0.984	0.03	128.86	28.23	18.40	9.75
0.00	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.983	0.03	128.44	28.00	16.95	8.31
0.02	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.6	3.2	0.596	0.11	0.982	0.03	128.20	27.84	15.47	6.84
0.04	35.2	5.8	7.1	4.9	0.6	1.8	28.5	3.1	30.2	3.4	31.3	3.1	0.596	0.11	0.980	0.03	128.05	27.75	14.17	5.54
0.06	35.2	5.8	7.1	4.9	0.7	1.8	28.5	3.1	30.2	3.4	31.0	3.1	0.596	0.11	0.978	0.03	127.73	27.56	12.79	4.20
0.08	35.2	5.8	7.1	4.9	0.7	1.9	28.5	3.1	30.2	3.4	30.8	3.1	0.596	0.11	0.974	0.04	126.86	27.11	11.62	3.05
0.10	35.2	5.8	7.1	4.9	0.7	1.9	28.5	3.1	30.2	3.4	30.6	3.1	0.596	0.11	0.967	0.04	126.76	27.04	10.37	1.82
0.12	35.2	5.8	7.1	4.9	0.8	1.9	28.5	3.1	30.2	3.4	30.3	3.1	0.596	0.11	0.958	0.05	126.53	26.87	9.33	0.80
0.14	35.2	5.8	7.1	4.9	0.8	1.9	28.5	3.1	30.2	3.4	30.1	3.1	0.596	0.11	0.944	0.05	126.17	26.61	8.22	0.26
0.16	35.2	5.8	7.1	4.9	0.9	2.1	28.5	3.1	30.2	3.4	30.0	3.1	0.596	0.11	0.927	0.06	124.56	25.76	7.38	1.08
0.18	35.2	5.8	7.1	4.9	1.0	2.2	28.5	3.1	30.2	3.4	29.8	3.1	0.596	0.11	0.904	0.07	123.87	25.30	6.61	1.83
0.20	35.2	5.8	7.1	4.9	1.1	2.2	28.5	3.1	30.2	3.4	29.7	3.1	0.596	0.11	0.877	0.07	123.17	24.84	5.96	2.45
0.22	35.2	5.8	7.1	4.9	1.2	2.2	28.5	3.1	30.2	3.4	29.6	3.1	0.596	0.11	0.847	0.08	122.66	24.41	5.48	2.92
0.24	35.2	5.8	7.1	4.9	1.3	2.3	28.5	3.1	30.2	3.4	29.5	3.1	0.596	0.11	0.814	0.08	122.15	23.95	5.03	3.34
0.26	35.2	5.8	7.1	4.9	1.4	2.3	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.782	0.09	121.75	23.55	4.72	3.65
0.28	35.2	5.8	7.1	4.9	1.5	2.3	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.751	0.10	121.23	23.07	4.45	3.91
0.30	35.2	5.8	7.1	4.9	1.6	2.3	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.723	0.10	120.73	22.57	4.29	4.07
0.32	35.2	5.8	7.1	4.9	1.7	2.3	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.696	0.10	120.11	22.08	4.21	4.15
0.34	35.2	5.8	7.1	4.9	1.9	2.4	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.674	0.10	119.38	21.44	4.12	4.22
0.36	35.2	5.8	7.1	4.9	2.0	2.4	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.657	0.11	118.65	20.90	4.12	4.20
0.38	35.2	5.8	7.1	4.9	2.1	2.4	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.643	0.11	118.27	20.50	4.08	4.24
0.40	35.2	5.8	7.1	4.9	2.2	2.4	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.634	0.11	117.81	20.08	4.10	4.22
0.42	35.2	5.8	7.1	4.9	2.3	2.5	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.625	0.11	117.27	19.58	4.11	4.21

0.44	35.2	5.8	7.1	4.9	2.4	2.5	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.621	0.11	116.88	19.19	4.18	4.16
0.46	35.2	5.8	7.1	4.9	2.5	2.5	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.616	0.11	116.52	18.84	4.19	4.14
0.48	35.2	5.8	7.1	4.9	2.6	2.5	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.613	0.11	116.24	18.53	4.18	4.15
0.50	35.2	5.8	7.1	4.9	2.6	2.6	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.611	0.11	115.18	18.01	4.15	4.18
0.52	35.2	5.8	7.1	4.9	2.7	2.6	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.609	0.11	114.81	17.66	4.17	4.16
0.54	35.2	5.8	7.1	4.9	2.8	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.609	0.11	113.96	17.20	4.22	4.10
0.56	35.2	5.8	7.1	4.9	2.9	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.608	0.11	113.66	16.90	4.23	4.09
0.58	35.2	5.8	7.1	4.9	3.0	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.607	0.11	113.29	16.58	4.25	4.07
0.60	35.2	5.8	7.1	4.9	3.1	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.606	0.11	112.80	16.16	4.23	4.10
0.62	35.2	5.8	7.1	4.9	3.1	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.606	0.11	112.49	15.88	4.25	4.07
0.64	35.2	5.8	7.1	4.9	3.2	2.7	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.605	0.11	111.95	15.57	4.23	4.09
0.66	35.2	5.8	7.1	4.9	3.2	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	111.68	15.33	4.24	4.08
0.68	35.2	5.8	7.1	4.9	3.3	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	111.46	15.14	4.26	4.06
0.70	35.2	5.8	7.1	4.9	3.3	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	111.26	14.95	4.26	4.07
0.72	35.2	5.8	7.1	4.9	3.4	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	110.88	14.69	4.27	4.07
0.74	35.2	5.8	7.1	4.9	3.4	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	110.66	14.49	4.28	4.06
0.76	35.2	5.8	7.1	4.9	3.5	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.604	0.11	110.41	14.31	4.28	4.06
0.78	35.2	5.8	7.1	4.9	3.5	2.8	28.5	3.1	30.2	3.4	29.3	3.1	0.596	0.11	0.603	0.11	110.22	14.15	4.27	4.06
0.80	35.2	5.8	7.1	4.9	3.6	2.8	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.603	0.11	110.04	14.00	4.30	4.04
0.82	35.2	5.8	7.1	4.9	3.6	2.8	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.603	0.11	109.85	13.85	4.31	4.03
0.84	35.2	5.8	7.1	4.9	3.6	2.8	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.603	0.11	109.72	13.71	4.33	4.01
0.86	35.2	5.8	7.1	4.9	3.7	2.8	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.603	0.11	109.61	13.60	4.30	4.04
0.88	35.2	5.8	7.1	4.9	3.7	2.8	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	109.47	13.48	4.32	4.02
0.90	35.2	5.8	7.1	4.9	3.7	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	109.26	13.33	4.32	4.02
0.92	35.2	5.8	7.1	4.9	3.7	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	109.14	13.22	4.32	4.02
0.94	35.2	5.8	7.1	4.9	3.8	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	109.04	13.12	4.32	4.01
0.96	35.2	5.8	7.1	4.9	3.8	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	108.92	13.01	4.32	4.01
0.98	35.2	5.8	7.1	4.9	3.8	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	108.79	12.91	4.31	4.02
1.00	35.2	5.8	7.1	4.9	3.8	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.602	0.11	108.69	12.81	4.33	4.00
1.02	35.2	5.8	7.1	4.9	3.9	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	108.60	12.72	4.35	3.98
1.04	35.2	5.8	7.1	4.9	3.9	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	108.12	12.42	4.36	3.97
1.06	35.2	5.8	7.1	4.9	4.0	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	108.01	12.34	4.37	3.96
1.08	35.2	5.8	7.1	4.9	4.0	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.93	12.27	4.38	3.95
1.10	35.2	5.8	7.1	4.9	4.0	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.84	12.20	4.38	3.95
1.12	35.2	5.8	7.1	4.9	4.0	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.61	12.07	4.39	3.93
1.14	35.2	5.8	7.1	4.9	4.0	2.9	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.56	12.01	4.39	3.94
1.16	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.39	11.90	4.39	3.94
1.18	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.32	11.84	4.39	3.94
1.20	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.25	11.79	4.37	3.96
1.22	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.17	11.72	4.36	3.97
1.24	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.10	11.66	4.37	3.96
1.26	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	107.00	11.59	4.36	3.97
1.28	35.2	5.8	7.1	4.9	4.1	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	106.95	11.55	4.35	3.98
1.30	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.601	0.11	106.89	11.51	4.36	3.97
1.32	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.85	11.46	4.35	3.98
1.34	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.79	11.42	4.36	3.97
1.36	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.76	11.39	4.35	3.98
1.38	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.72	11.35	4.35	3.98
1.40	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.65	11.31	4.34	3.99
1.42	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.61	11.27	4.34	3.99
1.44	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.58	11.24	4.35	3.98
1.46	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.55	11.21	4.34	3.99
1.48	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.50	11.18	4.34	3.99
1.50	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.46	11.15	4.35	3.98
1.52	35.2	5.8	7.1	4.9	4.2	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.44	11.12	4.35	3.98
1.54	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.40	11.10	4.35	3.98
1.56	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.37	11.07	4.35	3.98
1.58	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.34	11.04	4.35	3.98
1.60	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.31	11.02	4.34	3.99
1.62	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.29	11.00	4.34	3.99
1.64	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.26	10.97	4.34	3.99
1.66	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.22	10.95	4.34	3.99
1.68	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.19	10.93	4.34	3.99
1.70	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.17	10.91	4.35	3.98
1.72	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.15	10.89	4.35	3.98
1.74	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.14	10.88	4.35	3.98
1.76	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.11	10.86	4.33	4.00
1.78	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.09	10.84	4.33	4.00
1.80	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.07	10.83	4.33	3.99
1.82	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.05	10.81	4.33	3.99
1.84	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.03	10.80	4.32	4.00
1.86	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.01	10.79	4.32	4.00
1.88	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	106.00	10.78	4.32	4.00

1.90	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.98	10.76	4.32	4.00
1.92	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.97	10.75	4.32	4.00
1.94	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.96	10.74	4.32	4.00
1.96	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.94	10.73	4.32	4.00
1.98	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.93	10.72	4.32	4.00
2.00	35.2	5.8	7.1	4.9	4.3	3.0	28.5	3.1	30.2	3.4	29.4	3.1	0.596	0.11	0.600	0.11	105.93	10.71	4.32	4.00

B.7 Prediction Time Error Mean μ (S8)

S8	$\mathcal{RT}_A$	σ	$\mathcal{RT}_B$	σ	$\mathcal{RT}_C$	σ	$\mathcal{DA}_A$	σ	$\mathcal{DA}_B$	σ	$\mathcal{DA}_C$	σ	HR_B	σ	HR_C	σ	$t_{\mathcal{RT}_{AC}}$	$t_{\mathcal{RT}_{BC}}$	$t_{\mathcal{DA}_{AC}}$	$t_{\mathcal{DA}_{BC}}$
−1.00	35.2	5.8	7.1	4.9	3.2	2.9	28.5	3.1	30.2	3.4	34.1	3.4	0.596	0.11	0.825	0.08	110.38	15.18	26.67	18.05
−0.96	35.2	5.8	7.1	4.9	3.1	2.8	28.5	3.1	30.2	3.4	34.0	3.4	0.596	0.11	0.829	0.08	111.82	15.87	26.22	17.63
−0.92	35.2	5.8	7.1	4.9	2.9	2.7	28.5	3.1	30.2	3.4	33.9	3.4	0.596	0.11	0.833	0.08	113.29	16.71	25.76	17.18
−0.88	35.2	5.8	7.1	4.9	2.8	2.6	28.5	3.1	30.2	3.4	33.8	3.4	0.596	0.11	0.839	0.08	114.78	17.55	25.33	16.76
−0.84	35.2	5.8	7.1	4.9	2.5	2.4	28.5	3.1	30.2	3.4	33.7	3.5	0.596	0.11	0.845	0.08	117.07	18.93	25.03	16.50
−0.80	35.2	5.8	7.1	4.9	2.3	2.2	28.5	3.1	30.2	3.4	33.7	3.4	0.596	0.11	0.852	0.08	119.40	20.12	24.82	16.25
−0.76	35.2	5.8	7.1	4.9	2.1	2.0	28.5	3.1	30.2	3.4	33.6	3.4	0.596	0.11	0.858	0.08	120.89	21.02	24.36	15.80
−0.72	35.2	5.8	7.1	4.9	2.0	2.0	28.5	3.1	30.2	3.4	33.5	3.4	0.596	0.11	0.865	0.08	121.85	21.67	23.90	15.37
−0.68	35.2	5.8	7.1	4.9	1.9	1.9	28.5	3.1	30.2	3.4	33.4	3.5	0.596	0.11	0.871	0.07	122.98	22.38	23.47	14.99
−0.64	35.2	5.8	7.1	4.9	1.7	1.7	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.878	0.07	124.49	23.29	23.07	14.59
−0.60	35.2	5.8	7.1	4.9	1.6	1.6	28.5	3.1	30.2	3.4	33.2	3.4	0.596	0.11	0.885	0.07	125.38	23.95	22.73	14.23
−0.56	35.2	5.8	7.1	4.9	1.4	1.5	28.5	3.1	30.2	3.4	33.2	3.5	0.596	0.11	0.892	0.07	127.03	25.00	22.34	13.89
−0.52	35.2	5.8	7.1	4.9	1.2	1.4	28.5	3.1	30.2	3.4	33.1	3.5	0.596	0.11	0.900	0.07	128.00	25.73	21.94	13.49
−0.48	35.2	5.8	7.1	4.9	1.1	1.2	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.907	0.06	129.14	26.49	21.60	13.18
−0.44	35.2	5.8	7.1	4.9	1.0	1.1	28.5	3.1	30.2	3.4	32.9	3.4	0.596	0.11	0.912	0.06	130.05	27.15	21.22	12.75
−0.40	35.2	5.8	7.1	4.9	0.8	1.0	28.5	3.1	30.2	3.4	32.8	3.4	0.596	0.11	0.920	0.06	131.14	27.96	20.58	12.12
−0.36	35.2	5.8	7.1	4.9	0.7	0.9	28.5	3.1	30.2	3.4	32.7	3.4	0.596	0.11	0.926	0.06	132.00	28.58	20.07	11.60
−0.32	35.2	5.8	7.1	4.9	0.6	0.8	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.934	0.05	132.81	29.23	19.81	11.23
−0.28	35.2	5.8	7.1	4.9	0.5	0.7	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.940	0.05	133.36	29.68	19.50	10.92
−0.24	35.2	5.8	7.1	4.9	0.4	0.5	28.5	3.1	30.2	3.4	32.4	3.3	0.596	0.11	0.948	0.05	134.19	30.33	19.10	10.51
−0.20	35.2	5.8	7.1	4.9	0.3	0.5	28.5	3.1	30.2	3.4	32.3	3.3	0.596	0.11	0.956	0.05	134.74	30.84	18.66	10.07
−0.16	35.2	5.8	7.1	4.9	0.2	0.4	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.964	0.04	135.19	31.24	18.29	9.69
−0.12	35.2	5.8	7.1	4.9	0.2	0.3	28.5	3.1	30.2	3.4	32.1	3.2	0.596	0.11	0.972	0.04	135.51	31.54	17.98	9.31
−0.08	35.2	5.8	7.1	4.9	0.1	0.4	28.5	3.1	30.2	3.4	32.1	3.2	0.596	0.11	0.979	0.03	135.59	31.70	17.66	9.04
−0.04	35.2	5.8	7.1	4.9	0.2	1.1	28.5	3.1	30.2	3.4	32.0	3.2	0.596	0.11	0.984	0.03	133.03	30.52	17.27	8.66
0.00	35.2	5.8	7.1	4.9	0.6	1.7	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.983	0.03	128.44	28.00	16.95	8.31
0.04	35.2	5.8	7.1	4.9	1.4	2.6	28.5	3.1	30.2	3.4	31.9	3.2	0.596	0.11	0.974	0.03	119.67	23.18	17.04	8.41
0.08	35.2	5.8	7.1	4.9	2.1	3.2	28.5	3.1	30.2	3.4	32.0	3.3	0.596	0.11	0.959	0.05	112.26	19.13	17.44	8.87
0.12	35.2	5.8	7.1	4.9	2.9	3.8	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.941	0.05	104.25	15.12	17.77	9.20
0.16	35.2	5.8	7.1	4.9	3.4	4.1	28.5	3.1	30.2	3.4	32.1	3.3	0.596	0.11	0.928	0.06	100.24	12.94	17.89	9.34
0.20	35.2	5.8	7.1	4.9	3.8	4.4	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.917	0.06	97.02	11.34	17.87	9.37
0.24	35.2	5.8	7.1	4.9	4.0	4.5	28.5	3.1	30.2	3.4	32.2	3.3	0.596	0.11	0.908	0.07	95.37	10.43	18.18	9.64
0.28	35.2	5.8	7.1	4.9	4.2	4.6	28.5	3.1	30.2	3.4	32.3	3.3	0.596	0.11	0.900	0.07	94.25	9.81	18.65	10.06
0.32	35.2	5.8	7.1	4.9	4.3	4.6	28.5	3.1	30.2	3.4	32.4	3.3	0.596	0.11	0.892	0.07	93.47	9.35	19.13	10.54
0.36	35.2	5.8	7.1	4.9	4.4	4.7	28.5	3.1	30.2	3.4	32.5	3.3	0.596	0.11	0.885	0.07	92.87	8.99	19.47	10.89
0.40	35.2	5.8	7.1	4.9	4.5	4.7	28.5	3.1	30.2	3.4	32.6	3.3	0.596	0.11	0.877	0.07	92.43	8.70	19.90	11.37
0.44	35.2	5.8	7.1	4.9	4.5	4.7	28.5	3.1	30.2	3.4	32.7	3.4	0.596	0.11	0.872	0.08	92.17	8.50	20.38	11.86
0.48	35.2	5.8	7.1	4.9	4.6	4.7	28.5	3.1	30.2	3.4	32.8	3.4	0.596	0.11	0.867	0.08	91.99	8.31	20.83	12.37
0.52	35.2	5.8	7.1	4.9	4.6	4.7	28.5	3.1	30.2	3.4	32.9	3.5	0.596	0.11	0.861	0.08	91.84	8.21	20.93	12.57
0.56	35.2	5.8	7.1	4.9	4.6	4.7	28.5	3.1	30.2	3.4	33.0	3.5	0.596	0.11	0.855	0.08	91.74	8.10	21.38	13.03
0.60	35.2	5.8	7.1	4.9	4.7	4.7	28.5	3.1	30.2	3.4	33.1	3.5	0.596	0.11	0.848	0.08	91.42	7.93	21.89	13.48
0.64	35.2	5.8	7.1	4.9	4.7	4.7	28.5	3.1	30.2	3.4	33.2	3.5	0.596	0.11	0.842	0.08	91.29	7.82	22.28	13.86
0.68	35.2	5.8	7.1	4.9	4.8	4.7	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.836	0.08	91.18	7.72	22.60	14.22
0.72	35.2	5.8	7.1	4.9	4.8	4.7	28.5	3.1	30.2	3.4	33.3	3.5	0.596	0.11	0.832	0.08	91.07	7.62	22.86	14.50
0.76	35.2	5.8	7.1	4.9	4.8	4.7	28.5	3.1	30.2	3.4	33.4	3.6	0.596	0.11	0.827	0.08	90.97	7.55	23.10	14.75
0.80	35.2	5.8	7.1	4.9	4.8	4.7	28.5	3.1	30.2	3.4	33.5	3.5	0.596	0.11	0.821	0.09	90.94	7.50	23.67	15.29
0.84	35.2	5.8	7.1	4.9	4.8	4.7	28.5	3.1	30.2	3.4	33.7	3.6	0.596	0.11	0.816	0.09	90.85	7.41	24.13	15.82
0.88	35.2	5.8	7.1	4.9	4.9	4.7	28.5	3.1	30.2	3.4	33.8	3.7	0.596	0.11	0.812	0.09	90.75	7.30	24.58	16.32
0.92	35.2	5.8	7.1	4.9	4.9	4.7	28.5	3.1	30.2	3.4	33.9	3.7	0.596	0.11	0.807	0.09	90.72	7.22	24.84	16.58
0.96	35.2	5.8	7.1	4.9	4.9	4.7	28.5	3.1	30.2	3.4	33.9	3.7	0.596	0.11	0.805	0.09	90.64	7.18	25.15	16.89
1.00	35.2	5.8	7.1	4.9	4.9	4.7	28.5	3.1	30.2	3.4	34.0	3.7	0.596	0.11	0.803	0.09	90.60	7.12	25.61	17.33

APPENDIX C

Fixity Assurance

All fixity check values are provided as SHA-1 checksums, either of the git commits of the respective repository, or of the specified file. Files not present in the repositories are yielded by running the simulations as described in appendix A. In the table, the location `$A` refers to `simulations/01_detail`, and `$B` refers to the `prefetch-simulation` directory, where the regression results are stored, as described in section A.3; the short forms are used for typesetting reasons.

Fixity	Location	Value
Commit of spiceJ	git (tag 0.0.10)	ddef963c278b291387f351de30825f5b6c708215
Commit of scovilleJ	git (tag 0.0.7)	178f66738bec5bc771579228b528949ee7304e78
Commit of prefetch-simulation	git (tag 0.0.2)	484b02c5ab39e202cfc1fa10b0c9e52568627cfc
Exemplary Configuration	$A/configuration	3c5334b656673a07d2a09e1e79b48db081abc89c
Exemplary Genesis (TEX)	$A/genesis.tex	ad6beab64a5ce1e36f07718dedc18a0bff5eca33
Exemplary Genesis A	$A/genesis_A	d21a27219bb46bf9e26a30be8d19d2ea774db790
Exemplary Genesis B	$A/genesis_B	4cbd4538459f389b483ff4fed53f9500879cbd39
Exemplary Genesis C	$A/genesis_C	1aba5b6094af8c4bfd8281f95ab5bd78b9eaf2bc
Exemplary Result A (TEX)	$A/result-timeline_A.tex	2bae972f308cbf7bae2c764820aa431335f00da2
Exemplary Result B (TEX)	$A/result-timeline_B.tex	0251abfdf656808c1764850ce639350d6e17b9d9
Exemplary Result C (TEX)	$A/result-timeline_C.tex	52a30976b0057a0c8aa128c85cc7fbdec5909223
Exemplary Result A (TXT)	$A/result-summary_A.txt	59133d226d298911598cd34b8dbaa89c4e01450d
Exemplary Result B (TXT)	$A/result-summary_B.txt	c3d841fe31fd73bfea7982c238986b3589f10c3b
Exemplary Result C (TXT)	$A/result-summary_C.txt	91136c05cd61e5ffd50ed729af67fe6fd03d5383
Regression Results for S1	$B/s1.out	641bb8e44aab36d1468bfb326476ef47a6b11217
Regression Results for S2	$B/s2.out	180b09e5fc5d3c9ce1bff53e262bab42506fa18a
Regression Results for S3	$B/s3.out	5d8a3f7f198e019967e068abbcf8d3f571c9a73b
Regression Results for S4	$B/s4.out	1c14cca86541fac29b2dea91098ade623324c935
Regression Results for S5	$B/s5.out	735578c01ac23e92c706749f8c835ba315d4707a
Regression Results for S7	$B/s7.out	439fa0ccc60e95d33fa50adffbbd0939d4addfd3
Regression Results for S8	$B/s8.out	72dd00fa199fa10a199183c3ba57acb2e4d20fe0

Bibliography

[Bab+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. „Models and Issues in Data Stream Systems“. In: PODS '02. Madison, Wisconsin: ACM, 2002, pp. 1–16. ISBN: 1-58113-507-6.

[Bag11] S. Bagchi. „A Fuzzy Algorithm for Dynamically Adaptive Multimedia Streaming“. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 2 (Mar. 2011), 11:1–11:26. ISSN: 1551-6857.

[Bao+11] P. Bao, J. Pierce, S. Whittaker, and S. Zhai. „Smart Phone Use by Non-mobile Business Users“. In: MobileHCI '11. Stockholm, Sweden: ACM, 2011, pp. 445–454. ISBN: 978-1-4503-0541-9.

[Bar09] D. Barth. *Google Official Blog: The bright side of sitting in traffic: Crowdsourcing road congestion data.* `http://googleblog.blogspot.ca/2009/08/bright-side-of-sitting-in-traffic.html`. Accessed: 2015-04-23. 2009.

[BH10] P. Brooks and B. Hestnes. „User measures of quality of experience: why being objective and quantitative is important“. In: *Network, IEEE* 2 (Mar. 2010), pp. 8–13. ISSN: 0890-8044.

[BI95] D. Barbará and T. Imieliński. „Sleepers and workaholics: Caching strategies in mobile environments (Extended version)“. In: *The VLDB Journal* 4 (1995), pp. 567–602. ISSN: 1066-8888.

[Cao02] G. Cao. „Proactive power-aware cache management for mobile computing systems“. In: *Computers, IEEE Transactions on* 6 (June 2002), pp. 608–621. ISSN: 0018-9340.

[Cha99] X. Chang. „Network Simulations with OPNET“. In: WSC '99. Phoenix, Arizona, USA: ACM, 1999, pp. 307–314. ISBN: 0-7803-5780-9.

[Chu+11] A. Y. K. Chua, R. S. Balkunje, and D. H.-L. Goh. „Fulfilling Mobile Information Needs: A Study on the Use of Mobile Phones“. In: ICUIMC '11. Seoul, Korea: ACM, 2011, 92:1–92:7. ISBN: 978-1-4503-0571-6.

[GB02] S. Gitzenis and N. Bambos. „Power-controlled data prefetching/caching in wireless packet networks“. In: 2002, 1405–1414 vol.3.

[Han+13] P. Han, L. Chen, and J. Wu. „A Semantic-Based Dual Caching System for Nomadic Web Service“. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 511–521. ISBN: 978-3-642-40318-7.

[Hen+08] T. R. Henderson, M. Lacage, G. F. Riley, C Dowell, and J. Kopena. „Network simulations with the ns-3 simulator“. In: *SIGCOMM demonstration* (2008).

[Hum+14] W. Hummer, S. Schulte, P. Hoenisch, and S. Dustdar. „Context-Aware Data Prefetching in Mobile Service Environments“. In: 2014.

[Jai04] R. Jain. „Quality of Experience“. In: *IEEE MultiMedia* 1 (Jan. 2004), pp. 96–95. ISSN: 1070-986X.

[JD02] M. Jain and C. Dovrolis. „Pathload: A measurement tool for end-to-end available bandwidth“. In: Citeseer. 2002.

[JV12] J. Jayasudha and G. G. Vijayan. „Latency Reduction in Mobile Environment“. In: *Procedia Engineering* 0 (2012), pp. 2949 –2955. ISSN: 1877-7058.

[KL05] S. Kim and C. Leem. „Implementation of the Security System for Instant Messengers“. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 739–744. ISBN: 978-3-540-24127-0.

[Knu97] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0-201-89684-2.

[Law+06] C.-F. Law, X. Zhang, S.-M. Chan, and C.-L. Wang. „Smart Instant Messenger in Pervasive Computing Environments“. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-33809-3.

[LT98] S. C. Liew and D. C.-y. Tse. „A Control-theoretic Approach to Adapting VBR Compressed Video for Transport over a CBR Communications Channel“. In: *IEEE/ACM Trans. Netw.* 1 (Feb. 1998), pp. 42–55. ISSN: 1063-6692.

[Mok+11] R. Mok, E. Chan, and R. Chang. „Measuring the quality of experience of HTTP video streaming“. In: May 2011, pp. 485–492.

[Pal+10] E. Palme, C.-H. Tan, J. Sutanto, and C. W. Phang. „Choosing the Smart Phone Operating System for Developing Mobile Applications“. In: ICEC '10. Honolulu, Hawaii, USA: ACM, 2010, pp. 146–152. ISBN: 978-1-4503-1427-5.

[Par+11] S. Park, D. Oh, and B. Lee. „Analyzing User Satisfaction Factors for Instant Messenger-Based Mobile SNS“. In: Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, pp. 280–287. ISBN: 978-3-642-22308-2.

[RD00] Q. Ren and M. H. Dunham. „Using Semantic Caching to Manage Location Dependent Data in Mobile Computing“. In: MobiCom '00. Boston, Massachusetts, USA: ACM, 2000, pp. 210–221. ISBN: 1-58113-197-6.

[Sch+11] D. Schreiber, A. Göb, E. Aitenbichler, and M. Mühlhäuser. „Reducing User Perceived Latency with a Proactive Prefetching Middleware for Mobile SOA Access“. In: *Int. J. Web Serv. Res.* 1 (Jan. 2011), pp. 68–85. ISSN: 1545-7362.

[She+05] H. Shen, M. Kumar, S. K. Das, and Z. Wang. „Energy-efficient Data Caching and Prefetching for Mobile Devices Based on Utility“. In: *Mob. Netw. Appl.* 4 (Aug. 2005), pp. 475–486. ISSN: 1383-469X.

[Tab+12] K. Tabassum, M. Q. Syed, and A. Damodaram. „A Location Dependent Semantic Cache Replacement Strategy in Mobile Environment“. In: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2012, pp. 213–224. ISBN: 978-3-642-27298-1.

[Tua+98] N. J. Tuah, M. Kumar, and S. Venkatesh. „Investigation of a Prefetch Model for Low Bandwidth Networks“. In: WOWMOM '98. Dallas, Texas, USA: ACM, 1998, pp. 38–47. ISBN: 1-58113-093-7.

[Ver06] H. Verkasalo. „Empirical Observations on the Emergence of Mobile Multimedia Services and Applications in the U.S. And Europe“. In: MUM '06. Stanford, California, USA: ACM, 2006. ISBN: 1-59593-607-6.

[Yin+02] L. Yin, G. Cao, C. Das, and A. Ashraf. „Power-aware prefetch in mobile environments“. In: 2002, pp. 571–578.

[Zag+12] Q. Zagarese, G. Canfora, E. Zimeo, and F. Baude. „Enabling Advanced Loading Strategies for Data Intensive Web Services“. In: June 2012.

[Zha+13] W. Zhang, Y. Wen, Z. Chen, and A. Khisti. „QoE-Driven Cache Management for HTTP Adaptive Bit Rate Streaming Over Wireless Networks“. In: *Multimedia, IEEE Transactions on* 6 (Oct. 2013), pp. 1431–1445. ISSN: 1520-9210.

[A1 15] A1 Telekom Austria AG, Lasallestraße 9, 1020 Vienna, Austria. *Das Giganetz von A1 | A1.net*. `http://www.a1.net/hilfe-support/netzabdeckung/`. Accessed: 2015-04-23. 2015.

[Hut15] Hutchison Drei Austria GmbH, Brünner Straße 52, 1210 Vienna, Austria. *Abdeckungskarte*. `https://www.drei.at/portal/de/bottomnavi/kontakt-und-hilfe/netzabdeckung/`. Accessed: 2015-04-23. 2015.

[T-M15a] T-Mobile Austria GmbH, Rennweg 97-99, 1030 Vienna, Austria. *Das tele.ring Netz – tele.ring*. `http://www.telering.at/Content.Node2/service/netzabdeckung.php`. Accessed: 2015-04-23. 2015.

[T-M15b] T-Mobile Austria GmbH, Rennweg 97-99, 1030 Vienna, Austria. *T-Mobile Österreich: Netzabdeckung*. `https://www.t-mobile.at/netz/Das_Netz_der_Zukunft.php`. Accessed: 2015-04-23. 2015.