# Adaptive Server Control for Low-Latency Applications over the Cellular Network

Johan Siwerson

Samuel Gunnarsson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6175
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden


Printed in Sweden by Tryckeriet i E-huset
Lund 2022

# Abstract

Queue latency is a fundamental part of end-to-end latency and ensuring low end-to-end latency is crucial for a good quality of service in the upcoming 5G and 6G applications. In this work we present a model capturing the essential dynamics of an end-to-end cellular system upon which we derive a control structure that is suitable for tractable analysis, seeking to mitigate queue latency and provide stability guarantees. We do this by analysing the model and leveraging classic control theory. The controller is evaluated both in a self-built Julia simulation framework and a state-of-the-art network simulator where it also is compared to state-of-the-art congestion control algorithms. The results are promising and show that the proposed control structure performs on par with the compared congestion controllers despite its simple nature. The most prominent area of improvement is in signal estimation, where we believe that the proposed controller could gain in performance and yield better results.

## Keywords

Congestion control, Adaptive control, Control theory, Latency

# Acknowledgements

A special thanks goes to our industrial supervisor at Ericsson, Victor Millnert, Ph.D, who has not only constantly supported and helped us in our work but also led the thesis work in a well balanced manner and provided invaluable feedback along the way. We would also like to thank the other Ericsson employees at the S&T department in Lund for being very welcoming, for sharing their knowledge within communication systems and for the pleasant conversations at the office.

We would also like to express our gratitude to the Department of Automatic Control at LTH for sparking our interest in this field, which we did not know about three years ago. We also thank Adj. Prof. Johan Eker and Prof. Karl-Erik Årzén for their feedback and for volunteering to be the project's academic supervisor and examiner, respectively. This marks the end of our studies at LTH and we are both very thankful for the years we have spent studying here and proud of this academic contribution that marks the end of them.

Finally, I, Johan, would like to thank my father, who passed away during my studies at LTH, for sparking my interest in technology and for his unconditional devotion and support to me and my siblings.

# Contents

# List of Acronyms

**AIMD** Additive Increase Multiplicative Decrease.

**APCC** Adaptive P Congestion Controller.

**AQM** Active Queue Management.

**AS** Application Server.

**BS** Base Station.

**CWND** Congestion Window.

**DCTCP** Data Center TCP.

**ECN** Explicit Congestion Notification.

**IP** Internet Protocol.

**L4S** Low Latency Low Loss Scalable Throughput.

**LAN** Local Area Network.

**RED** Random Early Discard.

**RTP** Real-time Transport Protocol.

**RTT** Round Trip Time.

**SCReAM** Self-Clocked Rate Adaptation for Multimedia.

**SIMO** Single Input, Multiple Output.

**SINR** Signal-to-interference-plus-noise ratio.

**TCP** Transport Control Protocol.

**UE** User Equipment.

**WAN** Wide Area Network.

**XR** Extended Reality.

# 1

# Introduction

Society is changing and new fields are evolving when transmission rates on the internet increase and companies are looking at new ways of running their businesses. With the upcoming 5G and 6G cellular networks, high-rate latency-critical applications such as Extended Reality (XR) gaming, massive multi-player cloud gaming and remote-controlled robotics are expected to become more widespread and used. This new class of applications will require a low and predictable end-to-end latency to ensure a good quality of service [Schulz et al., 2017]. One of the main hurdles to achieve this is queue buildup due to network congestion and the increased latency that it brings.

Congestion has historically been mitigated by dropping incoming packets to network nodes when the queue in the node overflows and by letting the sender interpret the dropped packet as a sign to reduce its transmission rate. To increase the throughput and reduce the number of dropped packets, larger queues were introduced. However, this solution came with an increased end-to-end latency due to the increased queue delays that the larger queues led to. A new congestion paradigm, based on marking packets instead of dropping them, has emerged with the goal of reducing the queuing delay caused by large queues while maintaining a high throughput. This is possible due to having smaller queues, marking packets before the queues overflow and having senders that adapt their transmission rates based on marked packets instead of dropped packets.

What makes this problem difficult is that all users of a cellular network have to share the available capacity that constantly changes. The available capacity of a cellular network is affected by users that move around, new users that enter and old users that leave a cellular network, to name some examples. High-rate latency-critical applications naturally want to transmit a large amount of data while maintaining a low latency, and in order to do so, they must adapt their transmission rate as the available capacity in the cellular network changes. The idea is that these applications should adapt their sending rate based on the marked packets that occur when the network is about to become congested and thus mitigate an increase in the end-to-end latency.

There exist protocols today that react to marked packets and obtain lower end-to-end latency compared to protocols that react to dropped packets. However, both types of protocols usually contain lots of tunable parameters that make them difficult to understand how changes to the parameters will affect the protocol's performance.

## 1.1 Purpose, method and goals

The purpose of this thesis is to investigate how applications should react to marked packets using a control-theoretic approach. To do this, a mathematical model of the end-to-end system will be constructed and a controller will be derived where it is possible to reason about stability requirements. In order to aid in model and control synthesis, a simulation framework will be developed in a suitable programming language. The control algorithm will then be evaluated in Ericsson's network simulator and compared to state-of-the-art congestion controllers. Ideally, the controller proposed in this thesis should perform on par with current state-of-the-art algorithms while still being suitable for tractable analysis.

## 1.2 Delimitations

As mentioned before, the focus of this thesis is to derive a model for a simple server-client system which sends traffic via the cellular network and then utilize this model to synthesize an application throughput controller. Considering the limited time period it is necessary to limit the scope of the thesis. The thesis will therefore not focus on the details regarding different signal estimation techniques. Instead, it will assume that the necessary signals, discussed in Section 3.2, can be properly estimated and are available. For the same reason, the mechanics needed for a safe deployment over the Internet, such as re-transmission policy, protocol headers and handshakes, are not considered. Furthermore, the thesis will also limit its scope by only considering one sender and one receiver with a single path between them, where information is sent from the sender to the receiver and then back to the sender, as opposed to having multiple senders, receivers and paths as in a full network. This is further discussed in Section 3.2.

## 1.3 Thesis outline

The thesis is structured in the following way. Chapter 2 provides background information along with some related work needed to gain a deeper understanding of the problem treated in this thesis. Chapter 3 presents the derived system model along with an analysis of it. This is then followed by a presentation and analysis of the control strategy in Chapter 4. Following this, in Chapter 5, the model and control structure is evaluated in a self-developed Julia simulation and the results from the

Julia simulation are then validated in Ericsson's proprietary network simulator in Chapter 6. A discussion and analysis of the simulation results are presented in Chapter 7 along with some overall conclusions. Finally, in Chapter 8 a discussion about possible directions for future work and improvements to the thesis is presented.

## 1.4 Individual contributions

We have worked at Ericsson two days a week and three days from home. We have had re-occurring meetings with our industrial supervisor Victor Millnert twice a week. The vast majority of the work has been done by both students together. However, Johan has taken a bit more responsibility in the work of the background section and the generation of plots, whereas Samuel has contributed a bit more to the Julia simulation and the analysis. All in all, we are both very satisfied with each other and the time and effort we have put into this project.

# 2

# Background

The overall theme of this chapter is latency and particularly ways to achieve low end-to-end latency. An important component of end-to-end latency is queue delay, which in this thesis is the main subject to be mitigated. There are a lot of ways to reduce queue latency, however, only a few that reduce it without significantly affecting the throughput performance at the same time.

This chapter explains concepts necessary to grasp and understand the thesis in its entirety, such as an introduction to communication networks in Section 2.1, Congestion control in Section 2.2, Explicit Congestion Notification (ECN) in Section 2.3, Active Queue Management (AQM) in Section 2.4 and Low Latency Low Loss Scalable Throughput (L4S) in Section 2.5. Thereafter, the complications of communication in cellular networks are introduced in Section 2.6 and state-of-the-art congestion control algorithms are discussed in Section 2.7. Finally, the problem formulation of the thesis is presented in Section 2.8.

## 2.1 Communication networks

A network consists of a set of computers sharing resources via network nodes such as switches, routers and similar gateways. The computers use communication protocols accepted by the network in order to communicate with each other. The nodes of the network are interconnected based on physically wired, optical and wireless radio-frequency methods in various arrangements of network topologies. These networks can vary in size, and if the network is confined to a relatively small area, it is called Local Area Network (LAN) as opposed to Wide Area Network (WAN) which connects networks in larger geographic areas such as Lund, Sweden or the world. An example of a WAN is the Internet, which is considered the largest WAN in the world.

In communication networks, messages (packets containing information or data) are sent from a sender to a receiver according to certain rules called communication protocols. There are numerous communication protocols deployed on the Internet today, and many of them are intended for different stages of the communication

path. The Transport Control Protocol (TCP) is of particular interest in this thesis as it regulates how the packages should be transported. TCP is a connection-oriented protocol where a connection between sender and receiver has to be established before data can be sent. To transport the packages in a reliable manner, mechanisms such as acknowledgements, re-transmission and error detection are used, which adds to the reliability of the protocol but lengthens the latency.

These networks and communication protocols make it possible to transmit data reliably over various distances, but in order to do so effectively, some additions are needed.

## 2.2 Congestion control

The Internet has never been as large and important as today, nor have as many demands on it been made. In the past, the emphasis was for packets to reach the destination and concerns such as latency came second to delivering the packets safe and sound. As the Internet expanded in size and reach, it eventually became more congested due to the traffic growth entailed by the expansion. To remedy such congestions and to avoid a potential congestive collapse, various congestion avoidance schemes were added to the TCP, such as an AQM (discussed in Section 2.4), ECN (discussed in Section 2.3) and Congestion Window (CWND).

The CWND determines the number of bytes that can be sent out at any time and is calculated by estimating how much congestion there is on the link. The window is maintained by the sender as a means of stopping a node between sender and receiver from becoming overloaded. In most TCP variants, the CWND is driven by an Additive Increase Multiplicative Decrease (AIMD) approach, meaning that the CWND additively increases by a set constant when there is no congestion and multiplicatively decreases by a set factor when congestion is detected.

In order to achieve low latency in networks, it is beneficial to detect congestion before it causes buffer overflow and thus packet drop since packet drop triggers retransmission which in turn leads to high latency. This idea is directly in line with the idea of L4S (further discussed in Section 2.5), which utilize ECN for this purpose.

## 2.3 Explicit Congestion Notification

ECN is a mechanism allowing for end-to-end notification of network congestion with the purpose of reducing packet drops. It is intended to reduce end-to-end latency by giving a faster and explicit signal of congestion. ECN utilizes two bits in the Internet Protocol (IP) header to notify the presence of congestion. With these bits it is possible to map four different codepoints, explained in Table 2.1, informing network nodes if the packet support ECN and if congestion has been experienced.

For ECN to be deployed in a network it is required that all network nodes are ECN capable, which means that they are able to inspect the ECN codepoint and

| Codepoint name | Binary codepoint | Meaning |
|---|---|---|
| Not-ECT | 00 | Not ECN-capable transport |
| ECT(0) | 01 | ECN-capable transport |
| ECT(1) | 10 | ECN-capable transport |
| CE | 11 | Congestion experienced |

**Table 2.1** Definition of ECN codepoints and their meaning.

change it by setting the CE codepoint in case of congestion. In most variants of TCP, a marked packet has the same meaning as a lost packet at the endpoints of the network (sender and receiver). However, this interpretation is not unanimous for all TCP as it is up to each protocol to define it. A benefit of the explicit notification from ECN is that the packet drop probability can be reduced, which by avoiding a re-transmission, improves throughput and end-to-end latency and reduces jitter. Further improvement can be achieved if the notification is sent at an early stage in the congestion. This idea is further discussed in Section 2.4 below.

## 2.4 Active Queue Management

AQM is a policy for managing buffers in various nodes of a network. The purpose of the AQM is to reduce network congestion by notifying the sender through congestion signalling, using ECN. The idea of AQM is to mark or drop packets before buffers become full since a full buffer requires incoming packets to be discarded which causes large delays. Enqueuing a packet only if the queue is shorter than its maximum size (measured in bytes or latency) and dropping it otherwise, is a discipline known as *drop-tail*. Queues using the drop-tail discipline have a tendency to considerably increase latency and penalise bursty flows [Floyd and Jacobson, 1993]. Many AQM disciplines try to avoid these issues by dropping or marking packets according to a probability function, $p(t)$, in advance of a possible future congestion scenario.

Dropping a packet means that the node discards the packet. The receiver then notifies the sender that a packet has been lost, which understands that congestion has happened. Marking a packet instead exploits the two ECN bits in the IP header used to signal congestion. By using ECN, the receiver can immediately notify the sender that congestion has occurred, without the sender having to understand it itself with a dropped packet, which would cause delay.

The algorithms used in AQM for marking or dropping packets vary depending on what the network, in which it is used, is designed for. An example of an AQM based on Random Early Discard (RED) involves setting a *min threshold* and a *max threshold*, neither predefined and can be adjusted and vary in different nodes of the network. The thresholds are used to determine if an incoming packet should be enqueued or not. When a new packet reaches a node implementing RED, it is always

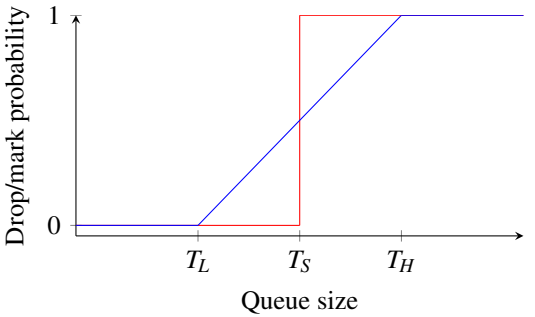



**Figure 2.1** Example of two different possible mark probability functions that could be used in various AQM algorithms. The step function (in red) can be varied by the parameter $T_S$ and the ramp function (in blue) can be varied by the parameters $T_L$ and $T_H$. A queue length below $T_L$ would never result in a dropped or marked packet while a queue length above $T_H$ would result in all packets being dropped or marked for the blue ramp function.

queued if the queue length, measured in bytes, is below the minimum threshold. If the queue length is in between the thresholds, the packet drop is determined by a probability function, $p(t)$, that depends on the queue length. In the final case, where the queue length is above the max threshold, an incoming packet is always dropped. A visual representation of this technique can be seen in the blue line in Figure 2.1, where in this case the queue size is measured in bytes.

Although the mark probability function used in different AQM's varies [Leung and Muppala, 2001], all of them are non-decreasing functions defined on $t \in \mathbb{R}^+$ and $p(t) \in [0,1]$. Two alternatives of a mark probability function are visualized in Figure 2.1.

## 2.5 L4S

L4S is a technology intended to ensure low latency and low packet loss for all Internet traffic without much reduction of throughput [Briscoe et al., 2022b]. L4S is based on ECN and the idea is to signal congestion early when the queue latency in a network node starts growing above a defined threshold. With this new approach, it is possible to keep a low queue delay and thus, a lower end-to-end latency while at the same time maintaining sufficient link utilization.

The process of congestion signalling to sender rate adaptation in L4S is described in the following way. The sender, through the use of a specific ECN codepoint (explained in Table 2.2) signals that the packet supports L4S. The nodes in the network will then recognize the packet as an L4S packet and if congestion occurs change the ECN bits to indicate that congestion has happened. When the packet

| Codepoint name | Binary codepoint | Meaning |
|---|---|---|
| Not-ECT | 00 | Not ECN-capable transport |
| ECT(0) | 01 | Not L4S-capable transport (Classic) |
| ECT(1) | 10 | L4S-capable transport (L4S) |
| CE | 11 | Congestion experienced |

**Table 2.2** Definition of L4S codepoints and their meaning.

reaches the receiver, and if CE is indicated somewhere along the path, the receiver notifies the sender about the congestion. Finally, the sender, notified of the congestion, adapts its sending rate with its congestion control algorithm.

In order for L4S to function properly, it is required that some conditions are met. The first regards the endpoints, where the receiver must be able to detect any CE codepoint and notify the sender of the experienced congestion. The sender must in the case of congestion reduce its sending rate. The last requirement regards the various network nodes where congestion can occur, such as routers, switches and Base Station (BS)s. These nodes should be able to distinguish an L4S packet from a classic packet[1] and treat them differently, to effectively manage congestion. A traffic flow of classic packets should be treated normally and sent to the classic queue, while L4S traffic have a separate reserved L4S queue. Therefore, two different queues are present in the L4S system.

In order to be able to distinguish L4S packets from classic packets, it is required that some ECN codepoints are interpreted differently compared to the interpretation presented in Table 2.1. The L4S codepoints and their meaning are presented in Table 2.2.

As can be seen in Table 2.1 and 2.2, codepoints ECT(0) and ECT(1) have different meaning in the tables. In the standard implementation of ECN, codepoints ECT(0) and ECT(1) are both interpreted as ECN-capable transport. However, in L4S the codepoints, aside from indicating that the packet is ECN-capable, also indicate what type of support the packet has, either L4S or Classic. As mentioned above, this allows for different treatment and queueing, where packets with the ECT(0) codepoint should be sent to the classic queue and packets that are L4S-capable should be sent to the L4S queue.

There is, however, a coupling between the two queues and by adjusting it in a clever way the queues will behave as if there only exists one queue from a network perspective. The coupling also guarantees that both queues share the available capacity equally. This means that neither the L4S traffic nor the classic traffic will starve the other one out. The details of the coupling are outside the scope of this thesis, however, the interested reader can find the details of the coupling in [Briscoe, 2019] and get an understanding of how L4S can be implemented in Figure 2.2.

Historically, ECN marked packets were interpreted as packet loss and the sender

[1] A classic packet has no L4S support and is not L4S-capable, as explained in Table 2.2

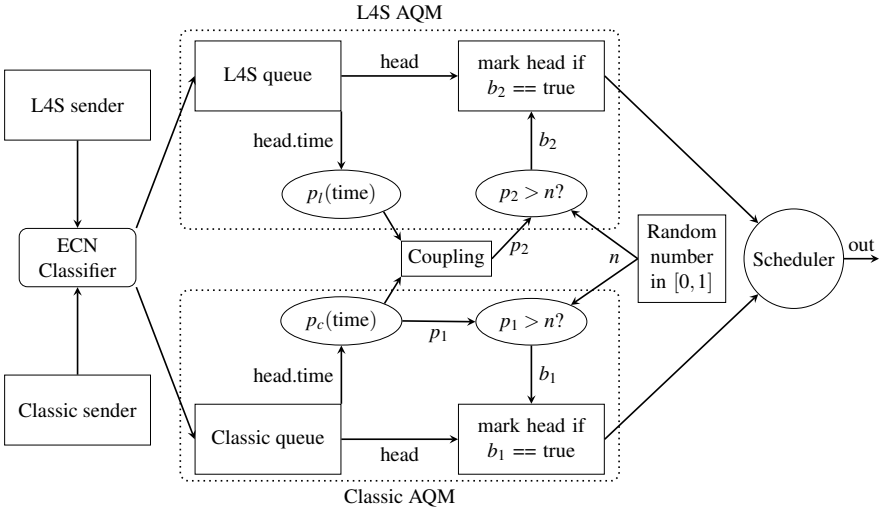



**Figure 2.2** Simplified overview of the structure in a network node that implements L4S. The incoming traffic can either come from an L4S or a Classic sender and by inspecting the ECN bits, it is possible to distinguish the two possible senders. Naturally, L4S traffic is sent to the L4S AQM while classic traffic is sent to the Classic AQM. The two AQMs use the queue latencies in combination with the chosen mark probability functions, $p_c(t)$ and $p_l(t)$ to determine if the packet should be marked or not. Finally, the packet gets scheduled and leaves the network node. By coupling the two queues in a clever way, which is outside the scope of this thesis, L4S guarantees that L4S traffic and classic traffic share the available link capacity equally while ensuring low latency for both types of traffic.

decreased its sending rate accordingly. However, with L4S, ECN marks are not treated as packet loss and instead have a much lower impact on the sending rate reduction [Briscoe et al., 2018]. This interpretation allows for more gentle congestion control and even throughput performance.

## 2.6 Cellular networks

In cellular networks, the amount of information you can send between a User Equipment (UE) and a BS is dictated by the link quality between these two entities. Usually, the UE is a mobile phone and the transmitted signals from the BS are affected by obstacles such as buildings and trees before it arrives at the UE. This, among other things, causes variation in the link quality and the maximum information one can send with the same signal power. A decrease in link quality can be counteracted with an increase in signal power by the BS. However, because of the physical limitations on antenna sizes and other regulations, the signal power can not increase

indefinitely, and with many users connected to the same BS, the resources in the BS have to be shared. The number of UE that wish to transmit data might also limit the available bandwidth for a user. The details of the BS are not the focus of this thesis and will not be discussed in more detail but later on in the thesis, it will be of great use to know that the BS have to degrade and determine the performance of the users to serve traffic from all the connected users.

## 2.7 Current state-of-the-art

### SCReAM

Real-time media streaming and other interactive services play a prominent role in today's internet. Such communication is often latency-critical and imposes a lot of requirements on transport. A protocol to tackle these requirements is Self-Clocked Rate Adaptation for Multimedia (SCReAM) [Johansson and Sarker, 2017], a window-based congestion control algorithm intended for Real-time Transport Protocol (RTP) traffic. SCReAM is based on the self-clocking principle of TCP and implements a similar technique to what is proposed in [Shalunov et al., 2012] to estimate network queue delay. The self-clocking principle is in essence a convention to only send new data upon receiving acknowledgements of previously sent packages. SCReAM also supports L4S, which makes it suitable for comparison in this master thesis.

The SCReAM algorithm mainly consists of three parts: network congestion control, sender transmission control and media rate control. All of which reside on the sender side of the algorithm. Since the algorithm is intended for RTP, extra blocks such as a media encoder and an internal queue for RTP packets are also present in the implementation. SCReAM is open source and thus its implementation details can be found online[2].

### DCTCP

Another state-of-the-art TCP algorithm is Data Center TCP (DCTCP) [Bensley et al., 2017], proposed for deployment in data centres. [Alizadeh et al., 2010] writes of a trend in data centre design to build highly available, highly performant computing and storage infrastructure using low-cost commodity switches. However, such low-cost switches tend to have limited queue capacities and thus be more susceptible to packet loss caused by congestion [Bensley et al., 2017]. DCTCP tries to solve this by using ECN, already available in commodity switches, combined with a control scheme at the source in a smart way.

The algorithm involves three main components:

**Marking at the switch:** For marking at the switch a mark probability function is needed in the AQM. Initially, the red step function in Figure 2.1 was used, where an

[2] `https://github.com/EricssonResearch/scream`

arriving packet is marked with the CE codepoint if the queue size is greater than $T_S$ upon its arrival and not marked otherwise. However, DCTCP also works with other mark probability functions.

**ECN-echo at the receiver:** The DCTCP receiver echoes back the congestion information to the sender using the ECN-echo flag in the TCP header.

**Controller at the sender:** An estimate of the fraction of packets that are marked, called $\alpha$, is maintained by the sender and updated once for each sending window of data according to:

$$\alpha \leftarrow (1-g) \cdot \alpha + g \cdot F, \tag{2.1}$$

where $F$ is the fraction of packets that were marked in the last sending window and $g \in [0,1]$ is a weight, deciding how $F$ affects $\alpha$ in the new estimate.

Although DCTCP is designed to only be used in data centres, it is a promising development for algorithms in wide area networks as well, thus suitable for comparison in this master thesis. DCTCP was one of the first TCP algorithms developed for L4S and laid the foundation for L4S-based protocols.

### TCP Prague

In a meeting in Prague in July 2015, members of the Internet Engineering Task Force (IETF) agreed to a list of modifications to DCTCP to make it safe to use over the public Internet. It was suggested that this list could be called "the Prague requirements" [Briscoe et al., 2018] and that the variant of DCTCP implementing the requirements could be called TCP Prague [Schepper et al., 2021] to distinguish it from DCTCP.

Since TCP Prague stems from DCTCP, the core scalable congestion control algorithm is the same. However, in order to make TCP Prague safe for deployment over the public Internet, it adopts new developments such as accurate ECN TCP feedback [Briscoe et al., 2022a], recent acknowledgement (RACK) [Cheng et al., 2021] and Paced Chirping [Misund and Briscoe, 2019]. In the start-up phase of the algorithm, the delay-based approach of Paced Chirping aims to increase the sending rate rapidly and reduce any overshoot. This allows TCP Prague to quickly utilize the available link capacity both in start-up and when the available capacity increases, e.g. other flows depart or the link capacity increases. A full and more detailed description of TCP Prague and its benefits and drawbacks can be found in [Schepper et al., 2021].

### BBR

Since its release in 2016 by Google, BBR [Cardwell et al., 2016] has gained much attention. It was first deployed in its B4 network and YouTube, after which several papers [Hock et al., 2017; Scholz et al., 2018; Jain et al., 2018] analysed its behaviour in simulations and real network testbeds and concluded that BBR can substantially improve throughput in these experiments.

The core idea of BBR is to estimate the bandwidth of the bottleneck (*BtlBW*), measure the Round Trip Time (RTT), calculate a bandwidth-delay product (*BDP*) and control the throughput based on these. There are four control states in BBR: StartUp, Drain, ProbeBW and ProbeRTT.

Similar to the start-up phase in classic TCP, StartUp is essentially a state where the sender increases its transmission rate with a fixed gain, for each RTT to probe for the maximum available bandwidth and build up a queue in the bottleneck. When the queue is consistently non-empty for three RTTs the algorithm transitions to the Drain state. In this state, a different gain parameter is used to reduce the sending rate below *BtlBW*. When the inflight packets match *BDP* the state machine transitions to ProbeBW.

In ProbeBW, the algorithm probes for available bandwidth during one RTT with a gain larger than one followed by a phase of reducing the sending rate with a gain less than one for one RTT. The gain is thereafter set to one for six RTTs after which the cycle restarts. Whenever the lowest RTT has not been sampled for ten seconds, BBR enters the ProbeRTT state to correctly estimate the lowest possible delay between the entities. This is done by reducing the inflight packets to four for at least one RTT and then returning to ProbeBW.

It was reported in [Hock et al., 2017; Ma et al., 2017] that the first version of BBR (BBRv1, with its core algorithm described above) suffers from RTT unfairness towards loss-based congestion controls and overloads the bottleneck link when co-existing with other flows. This caused substantial packet loss in links with shallow buffers. Since then, Google has updated BBR in BBRv2 [3] with the goal of solving these issues and to better coexist with classic TCP such as TCP Cubic [Ha et al., 2008] flows. In BBRv2, there is also support for ECN and BBRv2 uses a DCTCP-inspired way of adapting its sending rate to ECN-markings [Pan et al., 2022] similar to Equation (2.1).

## 2.8 Problem formulation

Finally, the components explained in this chapter should be tied up in a formulation of the problem addressed by this thesis. The problem formulation is best described by the following sentence: To derive a congestion controller for low latency applications in an L4S system with performance on par with these state-of-the-art algorithms, yet simple enough to do tractable analysis on.

[3] `https://github.com/google/bbr/blob/v2alpha/README.md`

# 3

# Model

This chapter presents the approach for modelling a system in a cellular network utilizing the mechanics explained in Chapter 2. The derived model is a simplified representation of the real system in order for it to be analysable, yet capture enough dynamics for it to be representable. The terminology used in the model approach is presented in Section 3.1, the model itself is presented in Section 3.2 and its dynamics in Section 3.3. Finally, the model is analysed in Section 3.4.

## 3.1 Terminology and physical representation

In the coming sections, the following terminology is used which is graphically summarized in Figure 3.1.

An *Application Server (AS)* is an adaptive sender that reduces its sending rate when it receives congestion signals in the incoming packets to the AS. The AS controls the sending rate of the traffic going down to the user equipment. There are many possible implementations of an AS but one could think of this as a cloud gaming server that generates the graphics for all users.

*User equipment* is the final destination of the traffic, for example, a mobile phone, that receives the packets generated in the AS and uses it in some way, for example by displaying the information on the screen if the packets generated on the AS is a video stream.

A *non-bottlenecked resource* is a network node that does not induce any significant queuing delays and can thus be seen as it mainly forwards packets. Many routers, switches and gateways on the internet can be modelled this way.

A *bottlenecked resource* is the network node where persistent queues are built up and can be viewed as the slowest link of the system. The maximum data that can be sent between the AS and UE is limited by the bottleneck.

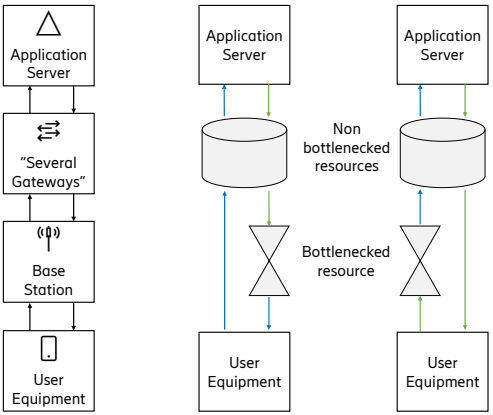



**Figure 3.1** To the left: Simplified overview of how an AS and a UE communicate with each other. In the middle: A model of the network when the bottleneck appears before the UE. To the right: A model of the network when the bottleneck appears after the UE. The green arrows are link delays before the bottleneck and the blue arrows are link delays after the bottleneck.

## 3.2 Model approach

Network packets travelling from an AS to the UE pass through several network nodes before it arrives. Any one of the network nodes can theoretically be the bottleneck resource, i.e. the slowest link, and determine the maximum available sending rate before queues are built up. However, the bottleneck of a network is usually known and engineered to be the access link [Briscoe et al., 2018] and in the cellular network domain, this corresponds to the BS. By implementing L4S in the bottleneck, the coupling between the AQM's, as discussed in Section 2.4, will make the bottleneck behave as a single queue from a network perspective even though two queues are present in the implementation of L4S.

Physical processes, such as an inverted pendulum or a water tank, adhere to certain mathematical expressions that are possible to derive in many cases. A complex combination of hardware and software, however, is unfeasible to model precisely and a simple model that captures the essentials between inputs and outputs is therefore often desirable when modelling software systems [Nylander et al., 2018]. The same reasoning can also be applied in this thesis and from a modelling and control perspective, the bottleneck resource is where the most important dynamics of the

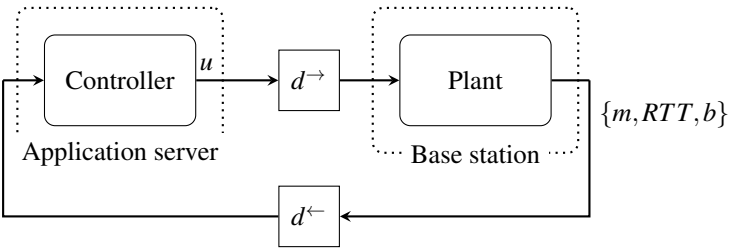



**Figure 3.2** Model of traffic from the AS to the bottleneck resource and back to the AS. The delays before the bottleneck are included in $d^{\rightarrow}$ while delays after the bottleneck are included in $d^{\leftarrow}$. The rate that leaves the AS is denoted by $u$.

system lie and is, therefore, the focus when deriving the model presented here.

The proposed model, visualized in Figure 3.2, consists of an AS, a BS and link delays, $d^{\rightarrow}$ and $d^{\leftarrow}$, between them. Inside the AS lies the controller regulating the sending rate, $u$. The acknowledgements arriving at the AS can be used to obtain three relevant signals. The first part is an ECN marking, denoted by $m$, notifying the sender of network congestion and described in Section 2.3. The second part is the $RTT$ of the acknowledged packet that just arrived at the AS, and the third part is the packet size and the inter-packet spacing used to determine the departure rate of the BS denoted by $b$.

### Converting ECN-markings to a mark probability

Reacting to ECN markings highly depends on the incoming packet rate and the current RTT. For instance, receiving three ECN markings out of ten packets arriving in a time window is very different to receiving the same number of markings out of a hundred packets in the same time window. In the former, a substantial reduction of throughput might be necessary to prevent a large end-to-end latency, while in the latter no reduction at all might due.

This phenomenon is something classic TCP suffers from, where the size of the sending window is cut by a factor of 2 when it receives an ECN notification. In effect, classic TCP reacts to the presence of congestion, not to its *extent*. In turn, this often leads to buffer underflows and loss of throughput [Briscoe et al., 2018].

More information, allowing for a better control of throughput, can be found in the *mark probability* $p(t) \in [0,1]$. Essentially, a number between (and including) 0 and 1, indicating the fraction of packets that are ECN marked in the queue of the BS. The marking strategy varies in different implementation scenarios and BSs and a cellular user might in fact utilise multiple different BSs and naturally experience different marking strategies. However, in this thesis, a certain $p(t)$ will be assumed in order to describe and analyze the system. The way different mark probability functions impact the system is discussed in Section 7.1.

The mark probability at any time can accurately be estimated over time using

for instance a moving average filter, a first-order discrete filter or by calculating the fraction between marked and unmarked numbers of bytes in a RTT. The latter of which is implemented in SCReAM [Johansson and Sarker, 2017]. Since the modelling and control part is the sole focus of this work, and to make the master thesis more applicable, it is assumed that the mark probability, $p \in [0,1]$ is present in the feedback signal instead of binary marks, $m \in \{0,1\}$ as would be the case in an actual implementation. Filtering or smoothing $m$ to get $p$ on the sender side is a design choice of L4S to ensure that congestion signals arrive back to the sender as fast as possible [Briscoe, 2019]. Due to time constraints, the best way to obtain $p$ from $m$ is not evaluated and left for future work and mentioned in Section 8.1.

## 3.3 Model dynamics

As discussed earlier, the approach used in this thesis is to model the bottleneck resource as a queue and treat other resources along the path (from AS to UE and back to the AS) as nodes that only contribute to the static link delays, $d^{\rightarrow}$ and $d^{\leftarrow}$, and not as nodes that limit the maximum sending rate. Nodes existing on the forward path, i.e. between the AS and the queue, contributes to $d^{\rightarrow}$ while nodes that exist on the backward path, i.e. between the queue and the AS, contributes to $d^{\leftarrow}$. Next, the dynamics for the two types of bottlenecks studied in this thesis will be presented: i) a physical bottleneck, and ii) a virtual bottleneck.

### Physical bottleneck dynamics

In a physical bottleneck, the dynamics can be described by denoting $u(t)$ as the sending rate from the AS and introduce the *incoming rate* to the queue as a time delayed version, $a(t) = u(t - d^{\rightarrow})$ and $b(t)$ as the *departure rate* of the queue. The queue growth, $\dot{q}(t)$ can then be expressed as

$$\dot{q}(t) = a(t) - b(t)$$

and the length of the queue, $q(t)$, at time $t$ can be found by integrating $\dot{q}$ as

$$q(t) = \int_0^t \dot{q}(x)\,dx.$$

The departure rate of the queue is described as

$$b(t) = \begin{cases} b^{\max}(t) & \text{if} \quad q(t) \neq 0 \\ \min\left(a(t), b^{\max}(t)\right) & \text{if} \quad q(t) = 0 \end{cases}$$

where $b^{\max}(t)$ denotes the *available link capacity* at time $t$. Exceeding $b^{\max}(t)$ with $a(t)$ will lead to queue build up and an increase in queue latency. To fully utilize the system and have minimum queue latency, $a(t)$ should be equal to $b^{\max}(t)$ with

$q(t) = 0$. If the maximum link capacity of the queue, $b^{\max}(t)$, is constant, the queue latency, $l(t)$, can easily be expressed as

$$l(t) = \frac{q(t)}{b^{\max}(t)}$$

and the mark probability function used in the AQM, $p^{\text{aqm}}(t)$, as

$$p^{\text{aqm}}(t) = \left[\frac{l(t) - T_L}{T_H - T_L}\right]_0^1$$

where $T_L$ denotes the minimum queue latency before any packets are marked, $T_H$ denotes the queue latency at which all packets are marked and $T_H - T_L$ determines the slope of the line. The specific mark probability function considered in this thesis, $p^{\text{aqm}}(t)$, is the blue function in Figure 2.1.

The three signals arriving back to the AS in this model are

$$p(t) = p^{\text{aqm}}\left(t - d^{\leftarrow}\right)$$

$$RTT(t) = d^{\rightarrow} + d^{\leftarrow} + l(t - d^{\leftarrow})$$

$$c(t) = b(t - d^{\leftarrow})$$

where $p(t)$ is a time-delayed version of $p^{\text{aqm}}(t)$, $RTT(t)$ is the current round trip time and $c(t)$ is a time-delayed version of the amount of data leaving the queue, obtained by observing the rate of acknowledgements currently being received.

### Virtual bottleneck dynamics

One way of obtaining lower queue latencies for a traffic flow is to assign priority to a traffic flow passing through the BS. This will lead to the traffic being prioritized over lower-priority traffic and thus reduce the physical queue and its associated latency (as long as the traffic does not exceed the physical capacity of the bottleneck, of course). However, there might still be a maximum capacity associated with such a high-priority flow. In a virtual bottleneck this constraint is not a physical limitation but a virtual capacity limit, hence, the name of "virtual bottleneck".

In a virtual bottleneck with priority there exists two limits, $b^{\max}(t)$ which is the physical upper limit on how much data the user can send and $b^{\max}_{\text{virt}}(t)$ which is the virtual upper limit that is assigned to the user. A necessary requirement is that $b^{\max}_{\text{virt}}(t) < b^{\max}(t)$ (because otherwise the physical capacity will be a limiting factor, and it will be a physical bottleneck as presented before). Naturally, in a virtual bottleneck system, there is a physical queue (as before) $q(t)$, but unique to the virtual bottleneck is that there is also a *virtual queue* $q_{\text{virt}}(t)$. The physical queue follows the same dynamics as before, but the virtual queue follows the dynamics presented in the equations below. In other words, if the user sends more than $b^{\max}_{\text{virt}}(t)$ bps, a virtual queue will build up and eventually result in marked packets since the virtual

queue latency will increase. The rate that leaves the physical queue, however, can get as large as $b^{\mathsf{max}}(t)$ before a physical queue starts growing and it is, therefore, possible to obtain a lower latency in systems with virtual queues compared to only using physical queues [Briscoe, 2019].

If the bottleneck is implemented as a virtual bottleneck with priority, its queue dynamics instead becomes

$$\dot{q}_{\mathsf{virt}}(t) = a(t) - b_{\mathsf{virt}}(t) \quad \dot{q}(t) = a(t) - b(t)$$

$$q_{\mathsf{virt}}(t) = \int_0^t \dot{q}_{\mathsf{virt}}(x)\,dx \quad q(t) = \int_0^t \dot{q}(x)\,dx$$

with the output from the virtual queue being

$$b_{\mathsf{virt}}(t) = \begin{cases} b_{\mathsf{virt}}^{\mathsf{max}}(t) & \text{if} \quad q_{\mathsf{virt}}(t) \neq 0 \\ \min\left(a(t), b_{\mathsf{virt}}^{\mathsf{max}}(t)\right) & \text{if} \quad q_{\mathsf{virt}}(t) = 0 \end{cases}$$

and the output from the physical queue being

$$b(t) = \begin{cases} b^{\mathsf{max}}(t) & \text{if} \quad q(t) \neq 0 \\ \min\left(a(t), b^{\mathsf{max}}(t)\right) & \text{if} \quad q(t) = 0 \end{cases}$$

which in essence describes that the output of the physical queue can exceed the virtual limit, $b_{\mathsf{virt}}^{\mathsf{max}}(t)$ when $a(t) > b_{\mathsf{virt}}^{\mathsf{max}}(t)$ by borrowing resources in the BS from the non-prioritized users. The physical limit $b^{\mathsf{max}}(t)$, can however not be exceeded. Assuming that $b_{\mathsf{virt}}^{\mathsf{max}}(t)$ and $b^{\mathsf{max}}(t)$ are constant, the virtual and physical queue latencies are described by

$$l_{\mathsf{virt}}(t) = \frac{q_{\mathsf{virt}}(t)}{b_{\mathsf{virt}}^{\mathsf{max}}(t)} \quad l(t) = \frac{q(t)}{b^{\mathsf{max}}(t)}$$

and the mark probability function is given by

$$p_{\mathsf{virt}}^{\mathsf{aqm}} = \left[\frac{l_{\mathsf{virt}}(t) - T_L}{T_H - T_L}\right]_0^1.$$

The outputs arriving back to the AS with a virtual bottleneck are

$$p(t) = p_{\mathsf{virt}}^{\mathsf{aqm}}(t - d^{\leftarrow}) \tag{3.1}$$

$$RTT(t) = d^{\rightarrow} + d^{\leftarrow} + l(t - d^{\leftarrow}) \tag{3.2}$$

$$c(t) = b(t - d^{\leftarrow}). \tag{3.3}$$

Although the dynamics of a physical bottleneck and a virtual bottleneck with priority are similar to each other, there are some key differences that make them behave differently.

First of all, $p(t)$ is a function of $l(t)$ in the physical bottleneck and instead a function of $l_{\mathsf{virt}}(t)$ in the virtual bottleneck. In a virtual bottleneck, this allows the sender to reduce its sending rate when a virtual queue latency is present but before a physical queue latency is formed. This can also be seen by looking at the output $RTT(t)$, which is identical in a virtual and physical bottleneck. In a virtual bottleneck, it is possible to obtain $p(t) > 0$, signaling that $u(t)$ should be lowered, while having $l(t) = 0$. This is not possible in a physical queue where having $p(t) > 0$ implies that $l(t) > 0$. This discrepancy makes $RTT(t)$ difficult to use in combination with $p(t)$ if the queue type is unknown.

The second thing to note is that the output, $b(t)$ are identical in both implementations. In neither queue types, it is not possible that $b(t)$ is greater than the maximum rate that can leave the queue, i.e. $b(t) \not> b^{\mathsf{max}}(t)$. In a virtual queue on the other hand, it is possible that $b(t)$ is greater than the rate that leaves the *virtual queue*, i.e. $b(t) > b^{\mathsf{max}}_{\mathsf{virt}}(t)$. This means that the system will behave differently depending on if the queue is physical or virtual.

The fact that the system behaves differently depending on what kind of bottleneck makes it more problematic to design a good controller for the system. Another circumstance that has to be considered is the fact that the type of bottleneck might even shift during a session if the UE connects to another BS which alters the feedback signals behaviour. This is discussed in more detail in Chapter 4 where a control structure that functions with both physical and virtual bottlenecks is presented, analysed and discussed.

## 3.4 Model analysis

Before digging into the control aspects of the time-varying and non-linear process, which dynamics are described above, some fundamental limitations are studied by linearizing the system with physical queues around two different equilibrium points. In this analysis, time delays in the process are left out and the only output that is considered is a non-saturated version of the mark probability, $p(t)$ with $T_L = 0$. This results in a significant aid in notation while preserving the fundamental dynamics of the system. The system with these relaxations is then described by

$$\dot{q}(t) = u(t) - b(t) \tag{3.4}$$

$$b(t) = \begin{cases} b^{\mathsf{max}}(t) & \text{if} \quad q(t) \neq 0 \\ \min(u(t),\, b^{\mathsf{max}}(t)) & \text{if} \quad q(t) = 0 \end{cases} \tag{3.5}$$

and the output of the system as

$$p(t) = \frac{q(t)}{T_H b^{\mathsf{max}}(t)}. \tag{3.6}$$

### Linearization around a non-empty queue

Observing the system in equilibrium where $q_0 \neq 0$ and $b^{\max}$ do not change, the time-varying and non-linear system described in Section 3.3 boils down to a time-invariant and linear system that is easier to analyze. The departure rate from the queue under these conditions is described by $b(t) = b^{\max}$ and Equation (3.4) becomes

$$\dot{q}(t) = u(t) - b^{\max}.$$

Introducing the state vector $x = \begin{bmatrix} q & b^{\max} \end{bmatrix}^T$ and choosing $u(t) = b^{\max}$ results in the equilibrium point $(q, b^{\max}, u, p) = (q_0, b_0^{\max}, u_0, p_0)$. The dynamics around this point is given by

$$\begin{cases} f_1(x,u) = \dot{q} = u - b^{\max} \\ f_2(x,u) = \dot{b}^{\max} = 0 \\ g(x,u) = p(x,u) = \frac{q}{T_H b^{\max}} \end{cases}$$

and their partial derivatives evaluated in the equilibrium point are

$$\begin{pmatrix} \frac{\partial f_1}{\partial q} & \frac{\partial f_1}{\partial b^{\max}} \\ \frac{\partial f_2}{\partial q} & \frac{\partial f_2}{\partial b^{\max}} \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_2}{\partial u} \end{pmatrix}^T = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$$

$$\begin{pmatrix} \frac{\partial g}{\partial q} & \frac{\partial g}{\partial b^{\max}} \end{pmatrix} = \begin{pmatrix} \frac{1}{T_H b_0^{\max}} & \frac{-q_0}{T_H (b_0^{\max})^2} \end{pmatrix}.$$

By introducing the new variables, $\Delta q = q - q_0$, $\Delta b^{\max} = b^{\max} - b_0^{\max}$, $\Delta u = u - u_0$ and $\Delta p = p - p_0$, the linearized system can be expressed as

$$\begin{pmatrix} \Delta\dot{q} \\ \Delta\dot{b}^{\max} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}}_{\text{A}} \begin{pmatrix} \Delta q \\ \Delta b^{\max} \end{pmatrix} + \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\text{B}} u$$

$$\Delta p = \underbrace{\begin{pmatrix} \frac{1}{T_H b_0^{\max}} & \frac{-q_0}{T_H (b_0^{\max})^2} \end{pmatrix}}_{\text{C}} \begin{pmatrix} \Delta q \\ \Delta b^{\max} \end{pmatrix}.$$

The controllability matrix, $\zeta$, can then be expressed as

$$\zeta = \begin{pmatrix} B & AB \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

which has rank 1 and the only state that is controllable is $q$.

The observability matrix, $Q$, can now also be expressed in the following way

$$Q = \begin{pmatrix} C \\ CA \end{pmatrix} = \begin{pmatrix} \frac{1}{T_H b_0^{\max}} & \frac{-q_0}{T_H (b_0^{\max})^2} \\ 0 & \frac{-1}{T_H b_0^{\max}} \end{pmatrix}.$$

which has rank 2 and indicates that both $q$ and $b^{\max}$ are observable.

In summary, having a queue results in a system where both the states $q$ and $b^{\max}$ are observable and $q$ is the only controllable state. The following section describes how this changes when the system operates around an equilibrium without a queue.

### Linearization around an empty queue

Similarly to the previous section, the goal is to find an equilibrium point and analyze the system around that point. The departure rate from the empty queue when $u(t) < b^{\max}(t)$ is the current sending rate, denoted by $u_0$. Equation (3.4) becomes

$$\dot{q} = u(t) - u_0.$$

With the same state vector as the previous section, $x = \begin{bmatrix} q & b^{\max} \end{bmatrix}^T$, the equilibrium point is described by $(q, b^{\max}, u, p) = (0, b_0^{\max}, u_0, 0)$. The system around this point becomes

$$\begin{cases} f_1(x,u) = \dot{q} = u - u_0 \\ f_2(x,u) = \dot{b}^{\max} = 0 \\ g(x,u) = p(x,u) = \frac{q}{T_H b^{\max}} \end{cases}$$

and the partial derivates evaluated in the equilibrium point are

$$\begin{pmatrix} \frac{\partial f_1}{\partial q} & \frac{\partial f_1}{\partial b^{\max}} \\ \frac{\partial f_2}{\partial q} & \frac{\partial f_2}{\partial b^{\max}} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_2}{\partial u} \end{pmatrix}^T = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$$

$$\begin{pmatrix} \frac{\partial g}{\partial q} & \frac{\partial g}{\partial b^{\max}} \end{pmatrix} = \begin{pmatrix} \frac{1}{T_H b_0^{\max}} & \frac{-0}{T_H (b_0^{\max})^2} \end{pmatrix}.$$

Introducing the new variables, $\Delta q = q - q_0$, $\Delta b^{\max} = b^{\max} - b_0^{\max}$, $\Delta u = u - u_0$ and $\Delta p = p - p_0$ the system can be expressed as

$$\begin{pmatrix} \Delta\dot{q} \\ \Delta\dot{b}^{\max} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}}_{\text{A}} \begin{pmatrix} \Delta q \\ \Delta b^{\max} \end{pmatrix} + \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\text{B}} u$$

$$\Delta p = \underbrace{\begin{pmatrix} \frac{1}{T_H b_0^{\max}} & 0 \end{pmatrix}}_{\text{C}} \begin{pmatrix} \Delta q \\ \Delta b^{\max} \end{pmatrix}$$

which yields the following controllability and observability matrices

$$\zeta = \begin{pmatrix} \frac{1}{T_H b_0^{\max}} & 0 \\ 0 & 0 \end{pmatrix}$$

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

Both of them have rank 1, where $q$ is the only controllable and observable state. Due to having an empty queue, $b^{\max}$ is no longer observable which aligns with the intuition behind Equation (3.5). By operating in the $q(t) = 0$ branch, there is no information about $b^{\max}$ as long as $u < b^{\max}$.

### Time-varying and non-linear difficulties

As previously mentioned, $b^{\max}(t)$ makes the system time-varying and more difficult to analyze compared to time-invariant systems. The two linearizations above assume that $b^{\max}(t)$ does not change and describe the system in steady-state well. To complement the linearizations and to understand how the system behaves when $b^{\max}(t)$ do change, reasoning about changes in $b^{\max}(t)$ is presented below.

If $u(t_0) = b^{\max}(t_0)$ and the system has a certain queue $q_0 = q(t_0)$, an increase in $b^{\max}(t)$, while keeping $u(t) = u(t_0)$, will lead to an observable reduction in queue latency. The closer $q_0$ is to 0 the harder it will be to observe the increase in $b^{\max}(t)$. A decrease in $b^{\max}(t)$ under the same conditions ($u(t_0) = b^{\max}(t_0)$ and $q_0 = q(t_0)$), will always result in an increase in queue latency which is possible to observe for the sender no matter the queue size $q_0$.

On the other hand, if $u(t_0) < b^{\max}(t_0)$ and the system is in steady-state, the queue will be empty. An increase in $b^{\max}(t)$ will not be possible to observe due to the empty queue, but a decrease in $b^{\max}(t)$ below $u(t_0)$ will lead to a queue latency that is possible to observe for the sender.

To summarize, due to the non-linearity and time-varying aspects of the system, whether the state $b^{\max}(t)$ is observable or not is dependent on the current state of the system and in which direction $b^{\max}(t)$ changes. An increase in $b^{\max}(t)$ is unobservable if there is no queue and observable if there is a queue. A decrease in $b^{\max}(t)$ to $b^{\max}_{\text{new}}(t)$ is unobservable if there is no queue and $u(t) < b^{\max}_{\text{new}}(t)$ and observable otherwise.

### Model analysis - summary

Although time delays were not included in the linearization and relaxations were made to $p(t)$, conclusions regarding the actual model can still be drawn. Detecting increases in available link capacity is only possible if the process queue is non-empty and decreases in available link capacity are generally possible to detect. For latency-critical applications, the queue should be as small as possible for minimum queue latency while from the perspective of observing increases in available link capacity, the queue should be as large as possible.

Since different BSs can implement both physical and virtual queues, the only approach to detect if congestion is present or not is to observe $p(t)$ instead of $RTT(t)$. Consequently, this has to be considered when designing the control algorithm for the model.

# 4

# Control

The model dynamics described in Section 3.3 introduce challenges due to the distinction between physical and virtual bottlenecks and as discussed in Section 3.4 the latency of the virtual queue is no longer captured in $RTT(t)$. This makes $RTT(t)$ difficult to use when controlling the system, whereas $p(t)$, which is latency dependent in both queue types, becomes a natural replacement. The current state-of-the-art algorithms that work in both queue types, tend to be complex and difficult to analyse. In this chapter, a control structure simple enough to analyse, yet with performance on par with the state-of-the-art algorithms is presented.

The proposed control structure, visualized in Figure 4.1, is presented in Section 4.1. Following this, in Section 4.2, a stability analysis of the system will be presented in three steps. Thereafter, an analysis of the error signal and its implications on the control strategy is performed in Section 4.3. Finally, the full controller with an adaptive gain and the resulting control law is presented in Section 4.4.

## 4.1 The semi-cascaded P controller

The goal of the control algorithm is to have a predictable and low end-to-end latency. To achieve this the algorithm needs to be conservative with regards to throughput in order to not fully fill the queue and respond to decreases in link capacity quickly. At the same time, applications demanding high throughput should also be satisfied and utilize any available link capacity. With this in mind, it would seem reasonable to control the system in a cascaded fashion, as shown in Figure 4.1, where the inner loop responds quickly to decreases in link capacity and the outer controller guarantees that any available link capacity is utilized.

The control strategy proposed in this master thesis is similar to cascaded control but instead of having two controllers, where one controller's output drives the set point of the other controller, this strategy only uses one controller but with two feedback loops. The outer loop drives the system towards a set target value of $p(t)$, denoted $p_{\mathsf{ref}}$. This makes sure that any available link capacity is utilized, no matter if the queue is physical or virtual. The inner loop takes care of the other problem -

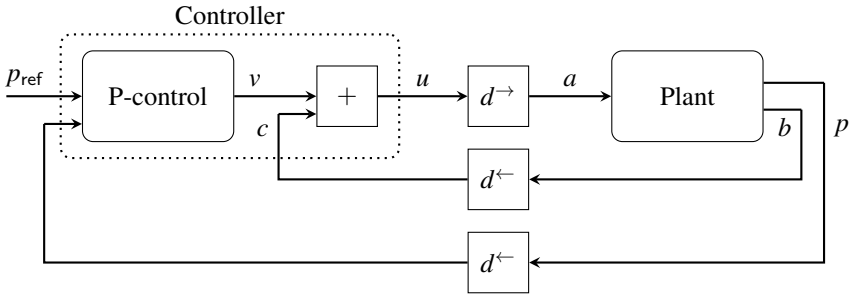



**Figure 4.1** Proposed control structure with the outer loop controlling the signal $v$, determining the desired growth rate of the queue and the inner loop adds $c$ to $v$ to get the incoming rate to the queue that corresponds to the calculated queue growth.

making sure that decreases in available link capacity are responded to quickly. By feeding back the rate that leaves the queue, $b(t)$, in the inner loop, it is possible to quickly detect decreases in available link capacity when a queue is present, since the rate that leaves the queue will decrease. This feedback also helps drive the system towards $p_{\text{ref}}$ when there is an increase in available link capacity and a queue is present since the outgoing rate will increase. If there is no queue when $b^{\max}(t)$ changes, $b(t)$ will not change and the controller will not have any immediate information about the capacity increase.

By forming the difference between the rate at which packets enter and leave the queue, $v(t) = a(t) - b(t)$, the queue can be viewed as a pure integrator, $\dot{q}(t) = v(t)$, similar to what has been done in [Nylander et al., 2018]. The idea is to view $v(t)$ as the control signal purely affecting the growth rate of the queue in the process. The bitrate that leaves the AS, $u(t)$, and ultimately $a(t)$ that enters the queue, is calculated by adding a time-delayed version of $b(t)$ denoted by $c(t)$ to $v(t)$. Since the process is now seen as a pure integrator, a P controller is proposed which will be able to follow reference values without any stationary errors as long as there are no disturbances acting on the output of the controller [Åström and Murray, 2008]. The control law is then given by

$$u(t) = c(t) + K\big(p_{\text{ref}} - p(t)\big) \tag{4.1}$$

where $K$ is the P controller gain. The control structure described above is visualized in Figure 4.1.

## 4.2 Stability analysis

There are many ways to analyze a control system and derive conditions on its stability. In this section, the stability analysis in continuous time will be presented in

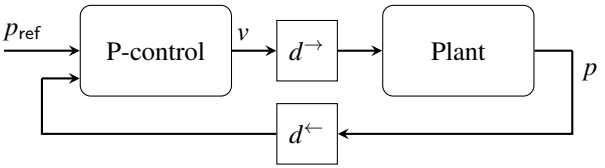



**Figure 4.2** The model structure used to analyse stability

three steps. The first part is a stability analysis of the system without any time delays and without saturations in $p(t)$. The second part will present a stability analysis with time delays but without saturations in $p(t)$. The third part analyses the stability of the system with delays and with saturations in $p(t)$.

The inner loop in Figure 4.1 exists so that the controller can output the signal $v$, determining the queue growth rate. Assuming that $c$ matches the output $b$ of the system, the inner loop is assumed to not alter the stability of the system and is ignored in the stability analysis. The system that is analysed in this section can be seen in Figure 4.2.

To derive the transfer function describing the plant, a few intermediate steps are presented for a better understanding of the final expression. The idea is to use $G_{pv}(s) = G_{pl}(s)G_{lq}(s)G_{qv}(s)$ and derive the three transfer functions separately and combine them afterwards. When a queue is present, the queue growth rate is described by

$$\dot{q}(t) = v(t - d^{\rightarrow})$$

and the plant can be seen as an integrator with the transfer function from $v$ to $q$ as

$$G_{qv}(s) = \frac{e^{-sd^{\rightarrow}}}{s}.$$

The queue latency is given by

$$l(t) = \frac{q(t)}{b^{\max}}$$

and the transfer function from $q$ to $l$ as

$$G_{lq}(s) = \frac{1}{b^{\max}}.$$

Before finding the transfer function between $p$ and $l$, two problems with $p(t)$ have to be addressed. As seen in Equation (3.3), $p(t)$ is bounded between 0 and 1 and contains an offset from 0. By introducing a non-saturated version of $p(t)$

$$\hat{p}(t) = \frac{l(t) - T_L}{T_H - T_L} = \frac{l(t) - T_L}{\Delta T}$$

and adding an offset to $\hat{p}(t)$ according to

$$\tilde{p}(t) = \hat{p}(t) + \frac{T_L}{\Delta T} = \hat{p}(t) + p_{\text{offset}}$$

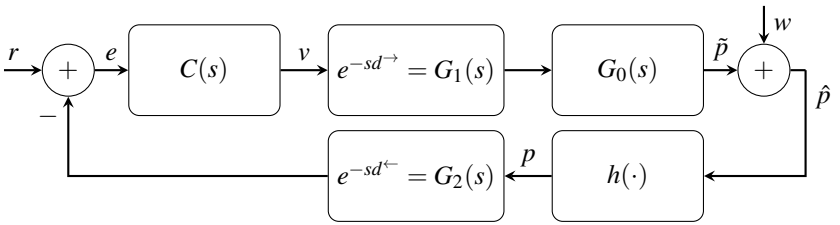



**Figure 4.3** Block diagram suitable for stability analysis where $r = p_{\text{ref}}$ and $w = -p_{\text{offset}}$ is introduced to aid in notation.

with a saturation function, $h(\hat{p}(t))$ as

$$p(t) = h(\hat{p}(t)) = [\hat{p}(t)]_0^1$$

the system dynamics are unchanged and the transfer function from $v$ to $\tilde{p}$ can be written as

$$G_{\tilde{p}v}(s) = \frac{1}{b^{\max}\Delta T s} e^{-sd^{\rightarrow}} = G_0(s)e^{-sd^{\rightarrow}}.$$

A block diagram of the system can now be illustrated as in Figure 4.3 and used to derive transfer functions suitable for analysis. Since a P controller is used, its transfer function is described by $C(s) = K$.

## System without time delays and no saturations

Lets start by investigating the stability of the system with no time delays and no saturations. This can be represented with $G_1(s) = G_2(s) = h(\cdot) = 1$ in Figure 4.3 and the loop transfer functions are then found by

$$P = W + G_0C(R-P) \Leftrightarrow P = \frac{1}{1+G_0C}W + \frac{G_0C}{1+G_0C}R \tag{4.2}$$

where

$$G_{pr}(s) = \frac{G_0(s)C(s)}{1+G_0(s)C(s)} = \frac{\frac{K}{b^{\max}\Delta T s}}{1+\frac{K}{b^{\max}\Delta T s}} = \frac{1}{1+\frac{b^{\max}\Delta T}{K}s} \tag{4.3}$$

and

$$G_{pw}(s) = \frac{1}{1+G_0(s)C(s)} = \frac{1}{1+\frac{K}{b^{\max}\Delta T s}} = \frac{\frac{b^{\max}\Delta T}{K}s}{1+\frac{b^{\max}\Delta T}{K}s}.$$

Since $G_{pr}$ and $G_{pw}$ have the same pole, $w$ will not be able to destabilize the system. For any value, $K > 0$, the pole of the closed-loop system will always lie in the left half-plane and result in a stable system. A suitable value of $K$ is, as expected, dependent on $b^{\max}$ and $\Delta T$.

### System with time delays and no saturations

Secondly, the system with time delays but without saturations is examined. This is represented with $h(\cdot) = 1$ in Figure 4.3. The closed-loop transfer functions can be found by

$$P = W + G_0G_1C(R - G_2P) \Leftrightarrow P = \frac{1}{1 + G_0G_1G_2C}W + \frac{G_0G_1C}{1 + G_0G_1G_2C}R \quad (4.4)$$

and similarly to the case without time delays, the transfer functions $G_{pw}$ and $G_{pr}$ have the same denominator. This means that the signal $w$ will, once again, not affect the stability of the system. Since delays are present in $G_1$ and $G_2$, it is no longer sufficient to analyse the stability with the closed-loop poles. The stability of the system can instead be analysed with the Nyquist stability criterion [Nyquist, 1932] by forming $G(s) = G_0(s)G_1(s)C(s)$ and $H(s) = G_2(s)$ and examining the open-loop transfer function $G_{\mathsf{ol}}(s) = G(s)H(s)$. In the following paragraph, an analytic upper bound on the control parameter $K$ is derived using the Nyquist criterion.

The Nyquist criterion states that conclusions regarding closed-loop stability can be drawn by examining the open-loop transfer function

$$G_{\mathsf{ol}}(i\omega) = G(i\omega)H(i\omega) = \frac{K}{b^{\mathsf{max}}\Delta T i\omega}e^{-i\omega d^{\rightarrow}}e^{-i\omega d^{\leftarrow}}$$

at two critical frequencies. At the cross over frequency $\omega_c$, where $|G_{\mathsf{ol}}(i\omega_c)| = 1$ it must hold that $\arg(G_{\mathsf{ol}}(i\omega_c)) > -\pi$ and for the frequency $\omega_0$ that results in $\arg(G_{\mathsf{ol}}(i\omega_0)) = -\pi$ it must hold that $|G_{\mathsf{ol}}(i\omega_0)| < 1$. The first step is to find the frequency $\omega_c$ at which $|G_{\mathsf{ol}}(i\omega_c)| = 1$ and by defining $D = d^{\rightarrow} + d^{\leftarrow}$, $\omega_c$ can be found as

$$|G_{\mathsf{ol}}(i\omega_c)| = \left|\frac{K}{b^{\mathsf{max}}\Delta T i\omega_c}e^{-i\omega_c D}\right| = \left|\frac{K}{b^{\mathsf{max}}\Delta T i\omega_c}\right| = \frac{K}{b^{\mathsf{max}}\Delta T\,\omega_c} = 1$$

which has the solution

$$\omega_c = \frac{K}{b^{\mathsf{max}}\Delta T}. \quad (4.5)$$

The second step is to make sure that $\arg(G_{\mathsf{ol}}(i\omega_c)) > -\pi$ for the cross over frequency $\omega_c$. The argument of $G_{\mathsf{ol}}(i\omega_c)$ is given by

$$\arg(G_{\mathsf{ol}}(i\omega_c)) = \arg\left(\frac{K}{b^{\mathsf{max}}\Delta T i\omega_c}e^{-i\omega_c D}\right) =$$
$$\arg\left(\frac{K}{b^{\mathsf{max}}\Delta T i\omega_c}\right) - \omega_c D = -\frac{\pi}{2} - \omega_c D.$$

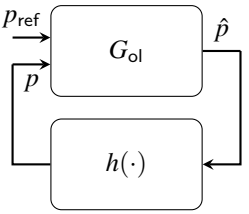



**Figure 4.4** Open-loop transfer function, $G_{\text{ol}}$, in feedback with non-linearity, $h(\cdot)$ suitable for analysis with Nyquist plots and Circle Criterion.

By inserting the $\omega_c$ found in Equation (4.5), the expression for stability becomes

$$\begin{aligned} -\frac{\pi}{2} - \omega_c D &> -\pi \\ \Leftrightarrow -\frac{\pi}{2} - \frac{K}{b^{\max}\Delta T} D &> -\pi \\ \Leftrightarrow \frac{\pi}{2} &> \frac{KD}{b^{\max}\Delta T} \end{aligned}$$

which yields the following condition

$$K < \frac{\pi}{2} \frac{b^{\max}\Delta T}{D} \tag{4.6}$$

needed to obtain a stable closed-loop system. The stability condition also aligns with the theory that the time delays in the system put an upper bound on the speed of the closed-loop system [Åström and Murray, 2008]. In essence Equation (4.6) shows that larger delays, $D$, require a lower $K$ and delays approaching 0 would, in theory, allow for an infinitely large $K$ similarly to the analysis made by investigating the closed-loop pole in Equation (4.3). The rate $b^{\max}$ that leaves the queue also needs to be considered when choosing a suitable $K$ together with the slope of the mark function determined by $\Delta T$. A $\Delta T$ very close to zero can be seen as turning the blue ramp mark function into the red step mark function in Figure 2.1 which naturally makes the system more difficult to control towards a certain reference value.

### System with time delays and saturation

Finally, the system with time delays and saturation is analysed to derive a condition on the control parameter $K$ that guarantees stability. Similarly to the previous analysis, the offset $w$ does not affect the stability of the system and can be ignored in this analysis. By placing $G_2(s)$ before $h(\cdot)$ in Figure 4.3, the block diagram can be redrawn as in Figure 4.4 which is suitable for analysis with the Circle Criterion [Shiriaev et al., 2004].

The Circle Criterion guarantees a globally asymptotically stable system if the non-linearity, $h(\cdot)$, can be sector bounded by two lines with slopes $k_1$ and $k_2$ and if

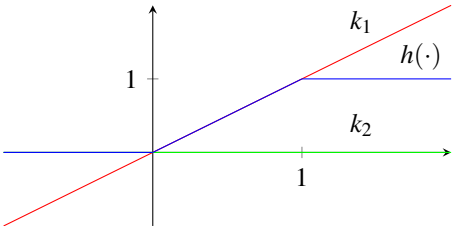



**Figure 4.5** The red, $k_1$, and green, $k_2$, lines sector bounds the blue non-linearity, $h(\cdot)$ caused by the saturations. The slopes are $k_1 = 1$ and $k_2 = 0$.

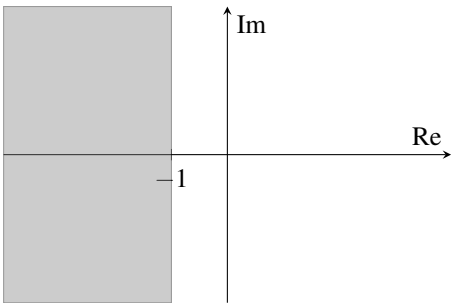



**Figure 4.6** In order to guarantee an asymptotically stable with the Circle Criterion, the Nyquist plot of $G_{\text{ol}}$ can not lie inside the shaded region determined by $k_1$ and $k_2$.

the Nyquist plot of the open-loop transfer function does not enter a circle with its diameter lying on the x-axis between $-1/k_1$ and $-1/k_2$. The non-linearity, $h(\cdot)$, is a saturation between 0 and 1 which is represented in Figure 4.5 with the blue line. The red and green lines, given by $k_1$ and $k_2$, sectors bounds $h(\cdot)$ and their slopes are trivially found as $k_1 = 1$ and $k_2 = 0$. Therefore, if the control parameter $K$ is chosen such that the Nyquist plot does not enter the shaded region in Figure 4.6, the system is guaranteed to be asymptotically stable.

Since $G_{\text{ol}}$ contains an integrator and delays, the gain at low frequencies will go towards infinity and the phase for low frequencies will be -90°. Therefore, the closest $G_{\text{ol}}$ will ever get to the shaded region in Figure 4.6 will be for $\omega_c = 0$. To derive a sufficient condition for stability, it must therefore hold that

$$\lim_{\omega \to 0^+} Re(G_{\text{ol}}(i\omega)) > -1. \tag{4.7}$$

Expanding $G_{\mathsf{ol}}(i\omega)$ leads to

$$G_{\mathsf{ol}}(i\omega) = \frac{K}{b^{\mathsf{max}}\Delta T i\omega} e^{-i\omega D} = \frac{K(\cos(-D\omega) + i\sin(-D\omega))}{b^{\mathsf{max}}\Delta T i\omega} = \frac{K(i\cos(-D\omega) - \sin(-D\omega))}{-b^{\mathsf{max}}\Delta T\omega}.$$

From this, the real part of $G_{\mathsf{ol}}(i\omega)$ can be extracted as

$$Re(G_{\mathsf{ol}}(i\omega) = \frac{K\sin(-D\omega)}{b^{\mathsf{max}}\Delta T\omega} = \frac{-K\sin(D\omega)}{b^{\mathsf{max}}\Delta T\omega}.$$

The limit in Equation (4.7) then gives

$$\lim_{\omega\to 0^+} Re(G_{\mathsf{ol}}(i\omega)) = \frac{-K\sin(D\omega)}{b^{\mathsf{max}}\Delta T\omega} = \frac{-KD}{b^{\mathsf{max}}\Delta T} > -1$$

which yields

$$K < \frac{b^{\mathsf{max}}\Delta T}{D}. \tag{4.8}$$

The derived condition is a sufficient but not a necessary condition for stability, meaning the system is guaranteed to be asymptotically stable if the condition is met and the system *might* lose its asymptotic stability if a larger K is chosen. The condition is similar to Equation (4.6) but more conservative which is the price one has to pay to guarantee asymptotic stability with saturations present in the system.

Finally, one should note that the analysis above is done in continuous time and that further analysis is needed to derive conditions on an implemented discretized controller. However, the derived conditions on stability in Equation (4.6) and (4.8) are still relevant and can be used as a guide in choosing control parameters.

## 4.3 Error analysis

In many classic control problems, particularly in industrial settings, there is commonly no bound on the magnitude of the error between process output and reference value. There is, however, usually a bound on the control actuation which can be limited by for example the range or speed of the actuator. This is not the case in this thesis. Since $p_{\mathsf{ref}}$ and $p(t)$ both are bounded by $[0,1]$, this means that the error $e(t)$ is bounded according to $e(t) \in [p_{\mathsf{ref}} - 1, p_{\mathsf{ref}}]$. For a P controller this means that also the control action, which is given by $v(t) = K \cdot e(t)$, is bounded since $e(t)$ is present in $v(t)$.

Therefore, the value of $p_{\mathsf{ref}}$ greatly impacts the way the controller adapts to increases or decreases in available link capacity since $p_{\mathsf{ref}} \cdot K$ determine the slope of the response. If $p_{\mathsf{ref}} < 0.5$ the controller responds faster to decreases and slower to increases in available link capacity. The opposite behaviour can be seen if $p_{\mathsf{ref}} > 0.5$.

Equal control response to both increases and decreases in available link capacity is achieved when $p_{\mathsf{ref}} = 0.5$.

Although the saturation of $p(t)$ limits the controller, the saturation also guarantees that the system output (with respect to $p(t)$) can not become unstable. The worst-case scenario is thus that $u(t)$ and $p(t)$ oscillate with constant magnitude instead of oscillating with increasing magnitude. However, as $l(t)$ is unbounded, the physical queue latency could in theory grow to infinity.

## 4.4 Adaptive P Congestion Controller

The prominent factor determining the stability condition of the system, according to Equation (4.8), is $b^{\mathsf{max}}(t)$. It is therefore important to choose a gain parameter $K$ such that the system is asymptotically stable even if $b^{\mathsf{max}}(t)$ changes. This implies that one either has to have good knowledge of which values $b^{\mathsf{max}}(t)$ will assume or one has to have good margins. However, a too small $K$ will make the controller slow which also is not desirable from a throughput perspective. Ideally, the gain parameter should adapt to the current $b^{\mathsf{max}}(t)$, which avoids the trade-off. One example of how to do this is by letting $K$ be determined through

$$K_{\mathsf{adapt}}(t) = \beta \frac{b^{\mathsf{max}}(t)\Delta T}{D} \tag{4.9}$$

where $b^{\mathsf{max}}(t)$ is estimated and $\beta$ is a percentage used to determine how close to the stability condition in Equation (4.8), $K_{\mathsf{adapt}}(t)$ is. A similar approach to adapt the gain that only uses signals available for the controller and therefore easier to implement is

$$K(t) = \beta \frac{c(t)\Delta T}{D}. \tag{4.10}$$

Combining the control law from Equation (4.1) with the adaptive $K(t)$ from Equation (4.10) results in the complete controller which in this thesis is called Adaptive P Congestion Controller (APCC) and described by the following control law

$$u(t) = c(t) + K(t)\big(p_{\mathsf{ref}} - p(t)\big). \tag{4.11}$$

## 4.5 Compensate for link delays

Controlling systems with persistent delays can be tricky and generally constrain the speed of the control actuation. This can also be seen in Equation (4.6) and (4.8) where a larger value of $D$ results in a smaller gain $K$ to obtain an asymptotically stable system.

Compensating for the link delays could be achieved with a Smith predictor [Smith, 1959] where the idea is to compensate for process delay. This would be

done by accurately modelling the system process, capturing its dynamics in a model with and without delays, letting the output signal from the model with delay cancel out the signal from the real process and only feed back the output from the model without delay for the controller to act upon. The predictor would perfectly compensate for the delays under the assumption that the model perfectly matches the system plant, however, in reality, such a perfect match would be very difficult to achieve. Designing a Smith predictor for the model in this thesis would be beneficial in steady-state and when the reference value changes and could in theory improve the performance. However, the longest latencies that occur in the system are those due to abrupt reductions in link capacity which would not be captured and a Smith predictor was therefore deemed unnecessary.

# 5

# Evaluation

To test the control structure presented in Section 4.1 and to evaluate the stability analysis made in Section 4.2, a simulation environment in Julia was developed. There are numerous open-source network simulators available which could be used for this purpose, however as Ericsson's state-of-the-art network simulator, later on, was used to test the algorithm in a real-world setting, there was no need to set up and learn these frameworks to perform the evaluation.

The developed Julia simulation environment is described in Section 5.1, the evaluation results are first presented and analysed in Section 5.2 with a fixed gain parameter, followed by an analysis of the results from the same simulation case but with an adaptive gain in Section 5.3. Finally, the results are related to a theoretical lower bound on the achievable queue latency, derived in Section 5.4.

## 5.1 Julia simulation

In order to test the designs made in this thesis and quickly iterate new versions of model and control structures, a self-built simulation environment has been implemented in Julia. The simulation is based on the Discrete Events package[1] available for Julia and the simulation has the following structure. Packets are transmitted from a sender to a queue node and then back again to the sender. Between these entities are static link delays. The packets are serviced in the queue with a deterministic service rate with added Gaussian noise to the service time for every packet. The service rate changes at determined timestamps in order to simulate changes in available link capacity. The sender estimates the outgoing rate of the queue by dividing the packet size by the time since the last packet arrived and is able to adapt its sending rate once for every arrived packet. The packets are saved for post-simulation plotting and analysis. The tunable parameters in the simulation are consequent with the ones presented in Section 3.3 and some of them are $b^{\mathsf{max}}(t)$, $T_L$, $T_H$, $d^{\rightarrow}$, $d^{\leftarrow}$, $p_{\mathsf{ref}}$, $\beta$ and $K$.

[1] `https://docs.juliahub.com/DiscreteEvents`

For the purpose of simplicity and to quickly develop a working model and control structure, the simulation is built to be quite simple and only information relevant to our model and control synthesis is simulated. Hence, no communication protocol, IP envelope or re-transmission policy is implemented. Multiple connected users, scheduling and priority of flows are also not included for the same purpose.

## 5.2 Stability evaluation

To get a good understanding of what control parameter $K$ that is suitable, the continuous-time stability conditions derived in Equation (4.6) and (4.8) are evaluated in the packet-based Julia simulation. To perform the evaluation, a simulation case with step and ramp changes to $b^{\max}(t)$ is used. Step changes are the most difficult to perform well on from a queue latency perspective and if the controller reacts well to these changes, the controller is likely to perform well in other scenarios as well.

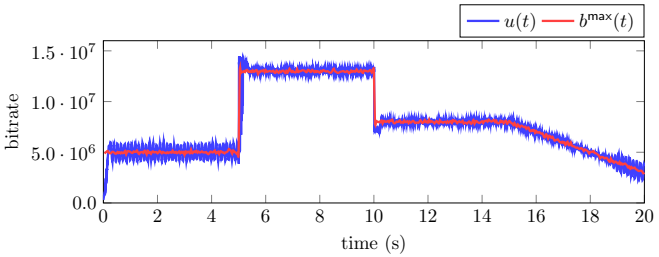
The simulation case is 20 seconds long and consists of four sections. At the beginning of the simulation, $b^{\max}$ starts of at $b^{\max} = 5$ Mbps until $t = 5$ s where $b^{\max}$ changes to 13 Mbps. At $t = 10$ s, $b^{\max}$ is lowered to 8 Mbps and at $t = 15$ s, $b^{\max}$ is ramped down with a slope of $-1$ Mbps/s. The Gaussian noise affecting the service rate, mark thresholds, link delays and reference value are kept constant during the simulation and their values are chosen as, $\sigma = 10^5$, $T_L = 0.008$ s, $T_H = 0.014$ s, $\Delta T = 0.006$ s, $D = 0.020$ s and $p_{\text{ref}} = 0.5$. Since the stability boundaries in Equation (4.6) and (4.8) is dependent on $b^{\max}$, $\Delta T$ and $D$, the boundaries will change as $b^{\max}$ changes in the simulation. The fixed control parameter, $K$, used during the entire simulation case is chosen to lie on the stability boundary determined by Equation (4.8) when $b^{\max} = 5$ Mbps and is calculated as

$$K = \frac{b^{\max}\Delta T}{D} = \frac{5 \cdot 10^6 \cdot 6 \cdot 10^{-3}}{20 \cdot 10^{-3}} = 1.5 \cdot 10^6$$

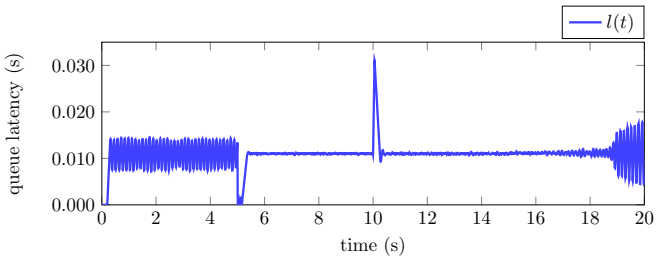
while the initial sending rate is set to 0.3 Mbps. The control law is described by Equation (4.1) and the results from the simulation case are presented in Figure 5.1 and contain four interesting behaviours that are highlighted below.

### Start-up

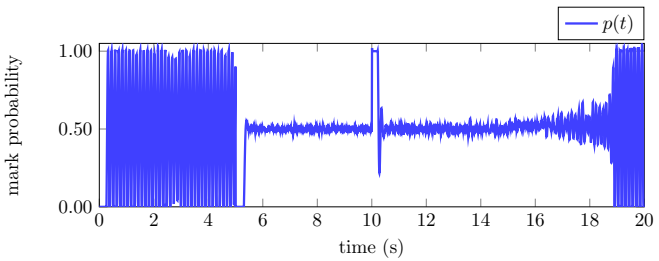
During the start-up phase when $b^{\max}(t)$ is unknown, the controller uses $v(t) = K \cdot (p_{\text{ref}} - p(t))$ to increase the outgoing rate of the queue and works as an integrator to reach $b^{\max}(t)$. Since $p(t) = 0$, the slope of the increase is determined by $K \cdot p_{\text{ref}}$ and as discussed in Section 4.3, the upper and lower bounds of $v(t)$ can be adjusted by changing $p_{\text{ref}}$ or $K$.




(a) Sending rate and available link capacity during the simulation case




(b) Experienced queue latency during the simulation case




(c) Mark probability during the simulation case

**Figure 5.1** Signals from the Julia simulation when $K = 1.5 \cdot 10^6$.

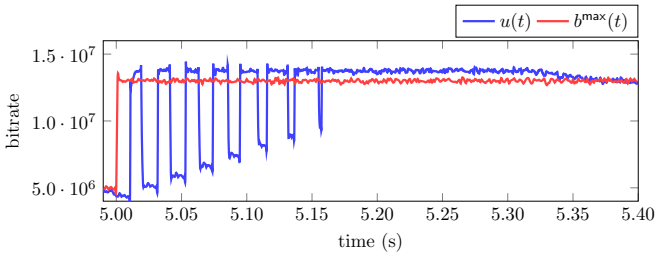



**Figure 5.2** Zoomed-in version of Figure 5.1(a) between $t = 4.99$ s and $t = 5.40$ s. The ringing occurs when the time delays in the system are larger than the time it takes to drain the queue.

## Stability condition

After the start-up phase, the system starts oscillating, indicating that the stability boundary has been surpassed. It could be the case that the system would oscillate for even smaller values of $K$, however the ramp down at the end of the simulation case confirms that the chosen $K$ does not generate an oscillatory system until $b^{\max}(t) \leq 5.0$ Mbps. This confirms that the continuous-time stability condition derived in Equation (4.8) holds for the discrete packet-based simulation if the sample period is kept short. The simulation case consists of 38 202 packets distributed over 20 seconds which on average results in a time of $h = {}^{20}/_{38202}$ s between the packets and a sample time of $h \approx 0.5$ ms. A discretized controller with a larger and fixed sample time $h$ will naturally influence the stability boundaries more than a controller acting on each packet.

## Link capacity increase

The third interesting behaviour is how the controller reacts to increases in $b^{\max}(t)$ after the start-up phase. Because a queue is present at $t = 5$ s, the inner loop in Figure 4.1 recognizes that $b(t)$ has increased and helps the controller utilize all available link capacity quickly. Because of the link delays, $d^{\rightarrow}$ and $d^{\leftarrow}$, the sending rate $u(t)$ might however temporarily start ringing. This can be seen in Figure 5.2 which is a zoomed-in version of Figure 5.1(a).

The ringing occurs when the sum of the link delays, $D$, is larger than the time it takes to empty the queue. This is the case in Figure 5.2 when $b^{\max}(t)$ increases from 5.0 Mbps to 13.0 Mbps and a queue is present. The outgoing rate of the queue becomes 13.0 Mbps until the queue is drained after which the outgoing rate is equal to the incoming rate. At $t = 5.0 + d^{\leftarrow}$ s, the increase in $b(t)$ is visible to the controller and it will quickly send at the newly found capacity thanks to the inner loop in Figure 4.1. When the queue has been drained, it outputs at the same rate that enters it which is the rate the controller sent $d^{\rightarrow}$ seconds ago. If the outer control loop would not exist, the sending rate $u(t)$ would oscillate between 5.0 and 13.0 Mbps

forever. Since $p(t) = 0$, the outer loop will constantly add $v(t) = K \cdot p_{\text{ref}}$ to $b(t)$ to form $u(t)$ and eventually stop the ringing. When the ringing stops, $u(t)$ is kept at a constant level above $b^{\text{max}}(t)$ until the load disturbance caused by $w$ in Figure 4.3 is rejected. By that time, $p(t)$ increases and $v(t)$ decreases until $p(t) = p_{\text{ref}}$ is achieved.

The ringing can be mitigated by introducing an estimator that estimates the new capacity and feeds that back to the controller. However, this is left for future work and discussed in Section 8.3.

### Link capacity decrease

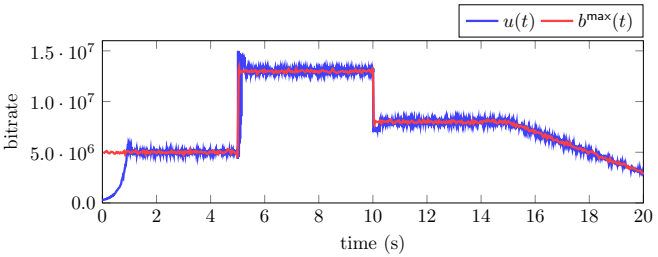
The last behaviour that is highlighted is how the controller reacts to decreases in $b^{\text{max}}(t)$ which is seen at $t = 10$ s in Figure 5.1. The inner loop of the controller quickly realizes that $b(t)$ is changed from 13.0 to 8.0 Mbps and the increase in queue latency results in $p(t) = 1.0$ which makes the outer loop output a negative $v(t)$. This makes $u(t) < b^{\text{max}}(t)$ and causes $l(t)$ and $p(t)$ to decrease and eventually reach their reference values. Ideally, there would be no queue latency increase but because of the delays, $d^{\rightarrow}$ and $d^{\leftarrow}$, in the system, it is not possible to react to link capacity decreases immediately and a queue latency increase is inevitable.

## 5.3 Simulation with adaptive controller gain

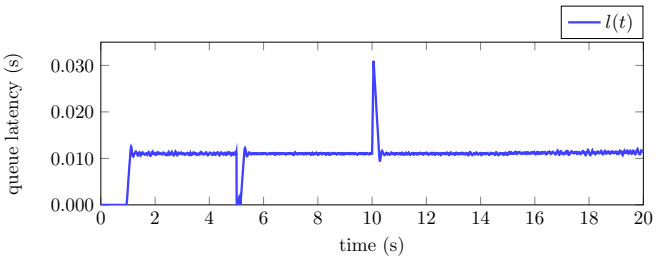
Having a fixed control parameter as in Section 5.2 when changes to $b^{\text{max}}(t)$ drastically influence the performance of the system, poses challenges. To get a stable system, the control parameter $K$ has to be chosen with respect to the lowest $b^{\text{max}}(t)$ that will occur during operation. Adapting $K$ to a low minimum $b^{\text{max}}(t)$ will make the system behave poorly in conditions when $b^{\text{max}}(t)$ is much larger than the minimum bound. A change in $b^{\text{max}}(t)$ can be viewed as a change to the closed-loop pole of the system whereas having an adaptive $K$ yields a constant pole placement for the closed-loop system and thus a constant stability margin regardless of changes in $b^{\text{max}}(t)$.

The implementation of the adaptive gain in this simulation uses APCC and the control law is therefore given by Equation(4.11). A satisfactory value of $\beta$ for the same simulation case that was described in Section 5.2 was experimentally found in the Julia simulation to be $\beta = 0.6$. The results from the simulation case with APCC are presented in Figure 5.3 and contain three behaviours that are particularly interesting and highlighted below.

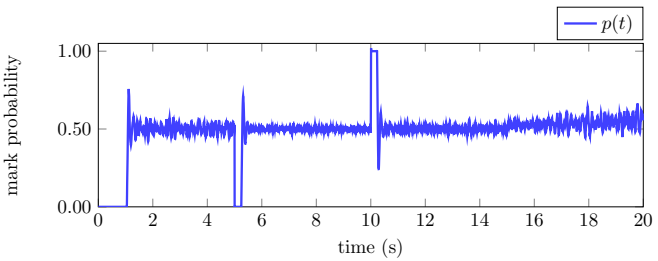
First of all, the start-up phase is different with an adaptive $K(t)$ compared to the fixed $K$ in Figure 5.1. Since the simulation case starts of with $u(t) = 0.3$ Mbps, the output $b(t)$ will also be 0.3 Mbps and result in a small $K(t)$. When $b(t)$ increases thanks to the outer loop, the gain also increases and $u(t) = b^{\text{max}}(t)$ after approximately 1 second. At this stage, the gain is below the stability boundary and no oscillations occur.



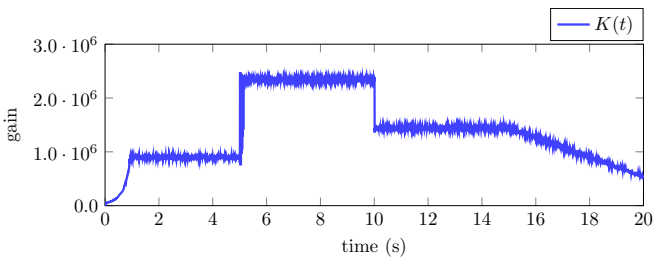

(a) Sending rate and available link capacity during the simulation case




(b) Experienced queue latency during the simulation case




(c) Mark probability during the simulation case




(d) Adaptive gain during the simulation case

**Figure 5.3** Signals from the Julia simulation when $K(t)$ is adaptive and $\beta = 0.6$.

Secondly, the same ringing in throughput occurs when $b^{\max}(t)$ increases with an adaptive $K(t)$ as with a fixed $K$. This will also induce a ringing in $K(t)$ and as explained in Section 5.2, a larger $K$ and smaller delays make the ringing disappear faster.

The third and last thing to note is the queue latency when $b^{\max}(t)$ decreases at $t = 10$ s. By comparing the queue latency obtained with a fixed $K$ in Figure 5.1(b) and the queue latency with an adaptive $K(t)$ in Figure 5.3(b) it is clear that there is no noticeable difference between the two. This is the case due to the adaptive $K(t)$ being close to the fixed $K$ after the step decrease in $b^{\max}(t)$ and the queue latency will therefore behave equally. If a lower fixed $K$ would have been chosen, the slope of $l(t)$ in Figure 5.1(b) at $t = 10$ s would have been smaller and resulted in a longer time to reach the reference value but the peak value of $l(t)$ would be the same. Analysing the lower bound of $l(t)$ when delays are present between the sender and queue and $b^{\max}(t)$ decreases in a step is interesting and examined in the coming section.

## 5.4 Best case queue latency

The delays, $d^{\rightarrow}$ and $d^{\leftarrow}$, in the system cause the signals between the controller and the queue to be delayed and introduce theoretical limitations on how fast the controller can adapt the incoming rate to the queue when $b^{\max}(t)$ changes and also what queue latencies that is possible to achieve. If the system is in steady state with $u(t) = b^{\max}(t)$ and $q(t) = q_{\mathsf{ss}}$ when $b^{\max}(t)$ changes from the steady state value $b_{\mathsf{ss}}^{\max}$ to the new value $b_{\mathsf{new}}^{\max}$ at time $t_c$, the controller will not receive this information until $d^{\leftarrow}$ time units later. When the controller receives the information about $b_{\mathsf{new}}^{\max}$, the controller is able to adapt $u(t)$ but because of the link delay it takes $d^{\rightarrow}$ time units before the new rate reaches the queue. By denoting $D = d^{\rightarrow} + d^{\leftarrow}$ and $\Delta b = b_{\mathsf{ss}}^{\max} - b_{\mathsf{new}}^{\max}$, it is possible to express the amount of bits that the queue has grown by as $\Delta q = \Delta b \cdot D$. The new queue size that have occurred purely because of the link delays and no matter what control structure is used can be calculated as $q(t_c + D) = q_{\mathsf{ss}} + \Delta q$ and its queue latency as

$$l_{\mathsf{min}} = l(t_c + D) = \frac{q(t_c + D)}{b_{\mathsf{new}}} = \underbrace{\frac{q_{\mathsf{ss}}}{b_{\mathsf{new}}}}_{\substack{\text{time to process} \\ \text{old queue size}}} + \overbrace{\frac{\Delta q}{b_{\mathsf{new}}}}^{\substack{\text{time to process} \\ \text{additional queue}}}. \tag{5.1}$$

The minimum latency, $l_{\mathsf{min}}$, is a theoretical minimum and under the assumptions that $b_{\mathsf{new}}^{\max}$ is constant and $b_{\mathsf{new}}^{\max} < b_{\mathsf{ss}}^{\max}$, values below this limit is not achievable without having the sender predict future values of $b^{\max}(t)$ and react before $b_{\mathsf{new}}^{\max}$ is sent back from the queue to the sender.

To give some intuition to the equations above, an example is provided in the following section where the numbers from Figure 5.1 and 5.3 are used and inserted into Equation (5.1) to evaluate how close the controller is to the theoretical lower bound $l_{\mathsf{min}}$.

### Best case queue latency - example

The step decrease of $b^{\mathsf{max}}(t)$ in Figure 5.1 and 5.3 occurs at $t = 10$ s and hence $t_c = 10$ s. The time where the step decrease to $b^{\mathsf{max}}(t)$ happens in Figure 5.1 and 5.3 is $t = 10$ s and hence $t_c = 10$ s. The queue latency right before the step decrease is 0.011 s and with $b_{\mathsf{ss}}^{\mathsf{max}} = 13.0$ Mbps the amount of bits in the queue before the decrease is given by $q_{\mathsf{ss}} = 0.011 \cdot 13\ 000\ 000 = 143\ 000$ bits. At $t = 10$ s, $b^{\mathsf{max}}(t)$ changes from 13.0 to 8.0 Mbps and $\Delta b = 13.0 - 8.0 = 5.0$ Mbps. With $D = 20$ ms, the queue will grow with $\Delta q = \Delta b \cdot D = 5\ 000\ 000 \cdot 0.020 = 100\ 000$ bits before any changes to $u(t)$ reaches the queue. The new queue size is given by $q(t_c + D) = 143\ 000 + 100\ 000$ bits and the minimum queue latency can now be calculated according to Equation (5.1) as

$$l_{\mathsf{min}} = \frac{243\ 000\ \mathsf{bits}}{8\ 000\ 000\ \mathsf{bits/s}} = 0.030375\ \mathsf{s} = 30.375\ \mathsf{ms}.$$

Comparing $l_{\mathsf{min}}$ to the queue latency obtained in Figure 5.1(b) and 5.3(b) indicates that the controller is able to perform close to the theoretical limit caused by the delays. This is due to the inner loop that quickly lowers the sending rate and keeps the peak value at a minimum in combination with the outer loop that makes sure to output a negative $v(t)$ when $p(t) > p_{\mathsf{ref}}$.

# 6

# Results and validation

With the simplifications in the Julia simulation, explained in Section 5.1, in mind, there is a need to validate the obtained model and control law in a setting similar to a real cellular network. This is done in Ericsson's state-of-the-art network simulator, briefly described in Section 6.1. The control algorithm of this thesis will then be evaluated and compared to the state-of-the-art algorithms highlighted in Section 2.7. Due to the distinction between the two queue types in Section 3.3, the results will be presented in two steps. First in a system implementing a physical bottleneck without priority of users in Section 6.2 and secondly in a system where different users can be prioritized in a virtual bottleneck in Section 6.3. The main purpose of this chapter is not to closely study and dissect the details of the performance of each algorithm, but rather to see if the developed control algorithm can perform on a similar level as the state-of-the-art algorithms.

## 6.1 Ericsson simulation

Although the Julia simulation, explained in Section 5.1, captures the essential behaviour of APCC, it is not able to capture the behaviour of the control algorithm in a real cellular network. Therefore, Ericsson's state-of-the-art network simulator is used to verify the algorithm in a real-world setting. Due to reasons regarding Ericsson's intellectual property, any details of Ericsson's simulator will not be shared in this thesis work. The simulator is solely used to verify the results obtained from the Julia simulation and to compare different control algorithms in a real-world setting.

Instead of adding the structure necessary for a real implementation of APCC, the algorithm is inserted into SCReAM to be able to test it in a simulation of a cellular network in the given time frame. In practice, this means that the part of SCReAM that calculates the target bitrate for its media encoder is overridden by APCC. The results of this chapter are thus, apart from the control algorithm, also based on the implementation of SCReAM. Using an already existing implementation of SCReAM saves time but also comes with its drawbacks. Details of how SCReAM estimates $p(t)$ and $b(t)$ from acknowledgements are not customized for

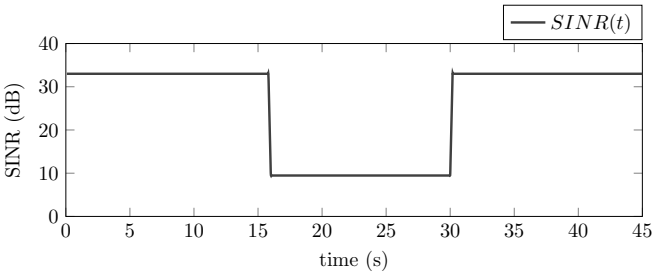



**Figure 6.1** Plot of the SINR in the scenario used in Ericsson's simulator.

APCC and to maximize the performance of the APCC, further modifications to SCReAM would be needed but are left for future work due to time constraints and discussed in Section 8.1.

Before moving on and presenting the results, it should be noted that the simulated DCTCP algorithm is a variant of both DCTCP and TCP Prague, it is essentially DCTCP but with an added fast-start feature from TCP Prague. For simplicity, this mix is referred to as "DCTCP", despite it being a variant of DCTCP and TCP Prague. Another important note is the following key difference between SCReAM, DCTCP and BBRv2 in Ericsson's simulator. SCReAM is implemented with an emulated media encoder while DCTCP and BBRv2 are not. This implies that the target bitrate set by DCTCP and BBRv2 is sent out immediately while SCReAM sets a target bitrate to a media encoder that tries to generate traffic corresponding to the target bitrate. Media encoders are not the focus of this thesis but in general, they cause fluctuations in the actual sending rate and are limited in how fast they can change their sending rate [Johansson et al., 2018]. This is good to keep in mind when comparing the results of APCC and SCReAM to DCTCP and BBRv2 in the coming sections.

## 6.2 Physical bottleneck

This section compares APCC to the state-of-the-art methods presented in Section 2.7 when the BS implements a physical queue as described in Section 3.3. To evaluate the algorithms, they were all run in a scenario consisting of background users sending as much traffic as possible, where the total capacity is equally shared between all users. The available link capacity in the simulation scenario is divided into three sections as described by the Signal-to-interference-plus-noise ratio (SINR) in Figure 6.1. A decrease in SINR is equivalent to a decrease in available link capacity and as discussed previously, a step decrease in available link capacity is a challenging task for any controller to perform well on. An increase in SINR is equivalent to an increase in available link capacity and is also challenging due to the

unobservability in certain states as discussed in Section 3.4. The scenario is simple to understand and implement but still exposes the algorithms to difficult conditions and if the algorithms perform well in this scenario, they are likely to perform well in other scenarios as well.

### APCC

The insertion of APCC into SCReAM results in a discretization of APCC and can be described in the following way. The target bitrate of the media encoder is updated with a sampling period of $h = 50$ ms and the control loop is executed for every new media frame generated by the media encoder. The adaptive $K(t)$ is updated every time the control loop executes and low-pass filtered according to

$$K(k) = (1-\alpha)K(k-1) + \alpha X$$

where $\alpha = 0.001$ and

$$X = \beta \frac{b(k)\Delta T}{D}$$

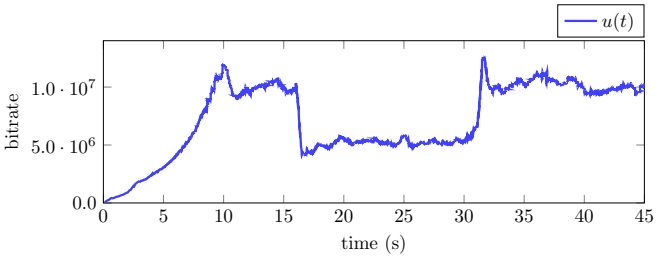
with

$$\beta \frac{\Delta T}{D} = 0.6.$$

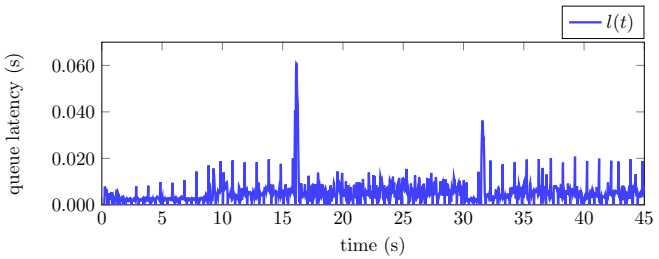
A satisfactory reference value was experimentally tuned and found as $p_{\mathsf{ref}} = 0.25$.

The mark probability, $p(t)$, is estimated once every RTT and calculated from the acknowledgements as the fraction between marked bytes and the total amount of bytes delivered this RTT. The return rate, $b(t)$, is also estimated from the acknowledgements but with a sample period of $h = 50$ ms and filtered by averaging the 8 last estimates. The results of running this controller in the SINR-scenario in Figure 6.1 are illustrated in Figure 6.2 and the most interesting behaviours are highlighted below.

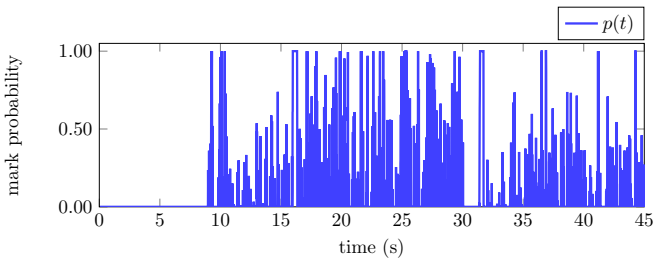
The start-up phase is similar to the Julia simulation with an adaptive $K(t)$ in Figure 5.3. APCC makes sure $u(t)$ is increased as long as $p(t) \neq p_{\mathsf{ref}}$, the gain $K$ is increased when $b(t)$ increases and the controller sends at $b^{\mathsf{max}}(t)$ around $t \approx 9$ s when $p(t) \neq 0$. An overshoot in $u(t)$ is present, similarly to the Julia simulation, to get rid of the constant load disturbance $w$ in Figure 4.3. At $t \approx 16$ s, $b^{\mathsf{max}}(t)$ decreases and the inner loop of APCC quickly decreases $u(t)$. The delays in $D$ give rise to a queue latency spike that the outer P controller of APCC removes by outputting a negative $v(t)$ since $p(t) > p_{\mathsf{ref}}$. When the queue latency is removed, APCC quickly sends at the new $b^{\mathsf{max}}(t)$ until $t \approx 30$ s where $b^{\mathsf{max}}(t)$ increases. Since $p(30) \neq 0$ and a queue therefore is present when $b^{\mathsf{max}}(t)$ increases, $b(t)$ increases as well and the new $b^{\mathsf{max}}(t)$ is found rapidly thanks to the inner loop but at the expense of an overshoot in $u(t)$ and $l(t)$. In the Julia simulation, an overshoot and ringing behaviour was present in $u(t)$ but it did not result in an overshoot in $l(t)$.



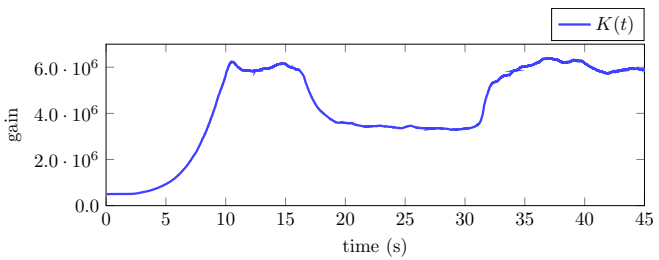

(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case



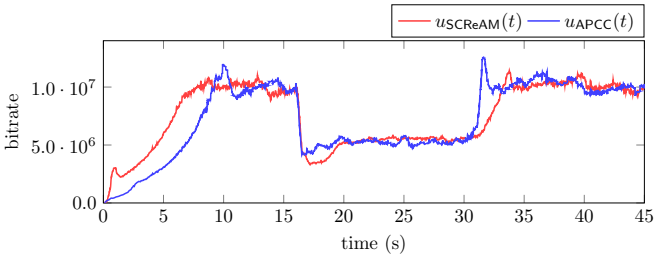

(d) Adaptive gain during the simulation case

**Figure 6.2** APCC in Ericsson's simulator and a physical bottleneck.

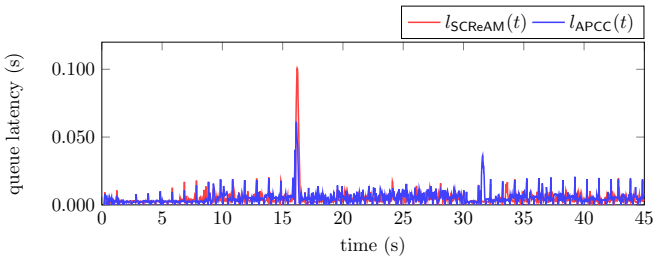
### SCReAM vs APCC

The comparison between SCReAM and APCC in Figure 6.3 shows interesting results throughout the simulation as discussed below.

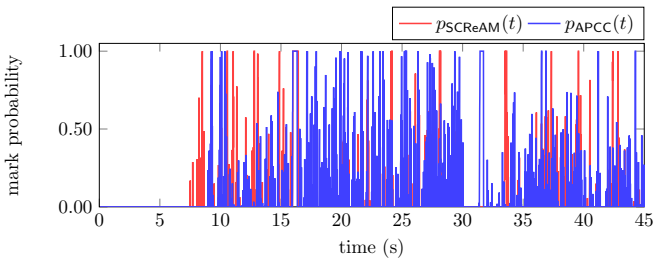
As can be seen in Figure 6.3(a), both SCReAM and APCC are relatively slow in the start-up phase, although SCReAM is slightly faster and does not suffer from an overshoot. This is likely due to the fact that SCReAM has a dedicated start-up mode that finds the available link capacity and then switches to a mode for steady-state as the initial marked packets are perceived. At around $t \approx 16$ s the available link capacity decreases, followed by a reduction of throughput for both algorithms where APCC is slightly faster than SCReAM. In turn, such a small difference in the speed of the reduction leads to a substantial difference in the queue latency as can be observed in Figure 6.3(b). After the decrease in available link capacity, both algorithms undershoot to allow the queue to be reduced before a steady state is found. At $t \approx 31$ s, APCC quickly finds the newly available link capacity at the expense of an overshoot, which in turn leads to a spike in $p(t)$ and thus a higher queue latency. SCReAM is slightly slower in this aspect as it probes for the new capacity, but a similar spike in $p(t)$ can be seen at $t \approx 34$ s without a significant increase in queue latency. There are many tunable parameters of SCReAM and finding a perfect combination of these for this simulation scenario is difficult. However, as APCC is based on SCReAM and everything except the target bitrate calculation is left unchanged, the comparison is considered fair and relevant.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case



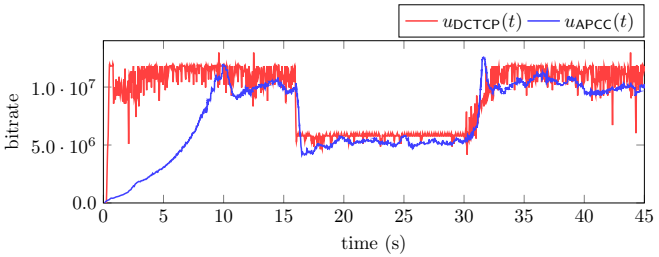

(c) Mark probability during the simulation case

**Figure 6.3** Comparison between APCC and SCReAM in Ericsson's simulator with a physical bottleneck.

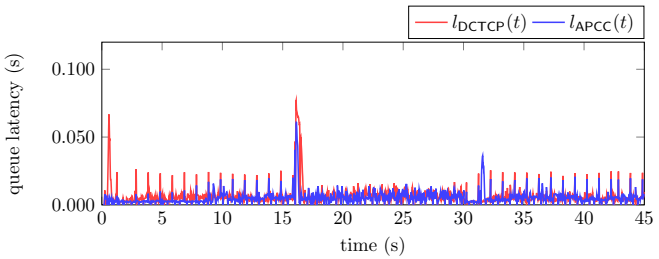
### DCTCP vs APCC

In this subsection, the performance of DCTCP and APCC is evaluated and a comparison can be seen in Figure 6.4.

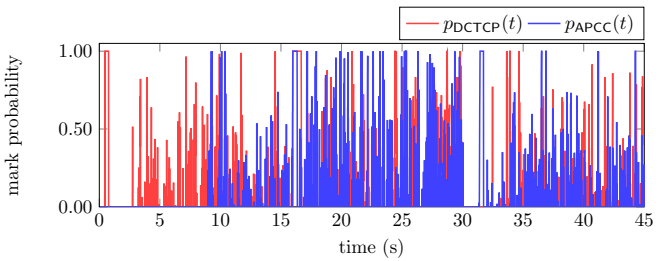
At the beginning of the simulation, DCTCP uses a specific start-up phase to find $b^{\mathsf{max}}(t)$ quickly at the expense of queue buildup as can be observed in Figure 6.4(a) and 6.4(b). Thereafter, DCTCP steadily utilizes the available capacity, but with a noisy signal. The noisy throughput could most likely be smoothened by a filter if necessary. DCTCP is able to transmit at a slightly higher rate than APCC throughout most of the simulation, this might be because of the media encoder implemented in SCReAM and thus also in APCC. The latency spike at $t \approx 16$ s in Figure 6.4(b) is particularly interesting as it would seem that the slower responding APCC would yield a higher spike compared to DCTCP. However, as the throughput of DCTCP is higher before the step decrease in $b^{\mathsf{max}}(t)$ a larger queue is formed compared to that of APCC, which results in higher latency when $b^{\mathsf{max}}(t)$ decreases. APCC is slightly faster in finding newly available capacity compared to DCTCP which gently probes to find it. This can most easily be seen in Figure 6.4(a) and 6.4(c) between $t \approx 30$ s and $t \approx 32$ s, as the mark probability is zero and the bitrate steadily increases. The results from this comparison should be considered with caution as DCTCP neither was intended for cellular systems nor fully tuned for this simulation nor uses a media encoder.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case

**Figure 6.4** Comparison between APCC and DCTCP in Ericsson's simulator with a physical bottleneck.

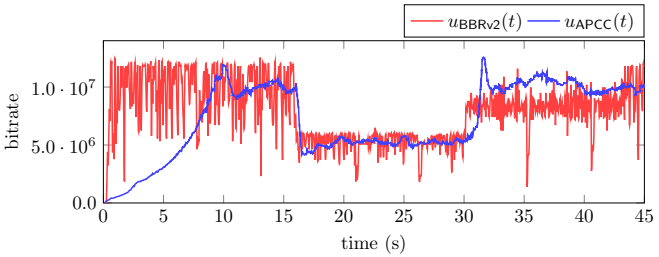
### BBRv2 vs APCC

A comparison between BBRv2 and APCC can be seen in Figure 6.5 and the behaviours differ a lot.

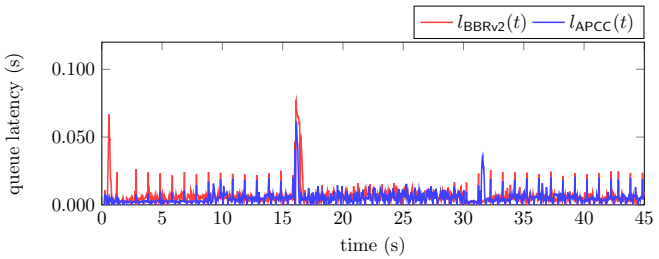
During the start-up phase, BBRv2 is in a start-up state and finds the maximum available link capacity quicker than APCC but at the cost of a large queue latency. When the maximum available link capacity is found, BBRv2 drains the queue and starts sending close to the rate at which the queue was built up. At this stage, BBRv2 cycles between a set of states and at $t \approx 16$ s, BBRv2 quickly reduces its sending rate. The value that the spike in queue latency assumes is dependent on what state BBRv2 was in right before the link capacity decreased. In Figure 6.5(b), the spike is approximately as large as for APCC but whether BBRv2 was in a state where it probed and tried to find more capacity or if it was in a state where it backed off and lowered its sending rate vastly affects the value of the spike.

At $t \approx 30$ s, where the available link capacity increases, BBRv2 finds some of the new capacity earlier than APCC but fails to find all of the capacity immediately. BBRv2 stays at a somewhat constant level below the maximum available link capacity until $t \approx 44$ s, while APCC utilizes more of the available link capacity after $t \approx 31$ s. The results from this comparison should also be considered with caution as BBRv2 neither was intended for cellular systems nor fully tuned for this simulation and does not include a model of a media encoder.

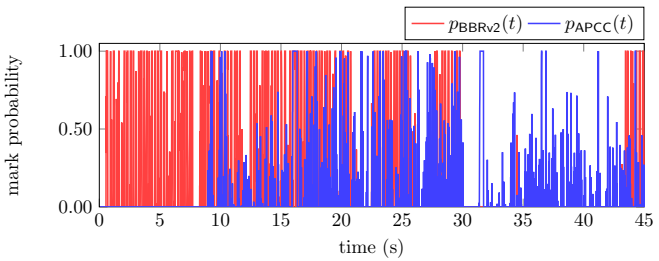
An important concluding remark is that the performance of BBRv2 heavily depends on the numerous tuneable parameters for the different states. Better performance could be achieved if BBRv2 was tuned further but the parameters used in this simulation still yield representable results for the behaviour of the algorithm.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case

**Figure 6.5** Comparison between APCC and BBRv2 in Ericsson's simulator with a physical bottleneck.

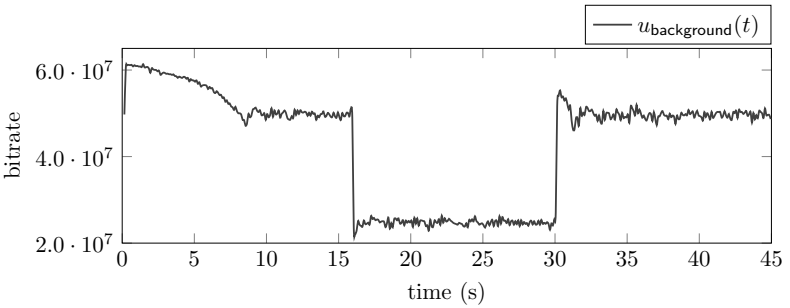



**Figure 6.6** Throughput plot of the background user in the simulation with the virtual bottleneck and one APCC user.

### Summary of physical bottleneck

All of the algorithms above are able to maintain a low queue latency in most parts of the simulation, apart from when the link capacity decreases drastically. Some algorithms handle this task better than others, but one has to keep in mind that the algorithms each were developed for different purposes that might not align with this simulation case. The speed at which the algorithms find and utilize newly available link capacity can differ and if the algorithms are tuned poorly, might be slow.

The spikes in queue latency align with the calculation in Equation (5.1) and never subceed this theoretical lower bound. However, if even lower latencies are required, a way to reduce queue latency is to have virtual queues with priority of flows.

## 6.3 Virtual bottleneck

With a virtual queue, the system dynamics change to the dynamics presented in Section 3.3 with a soft upper bound $b^{\mathsf{max}}_{\mathsf{virt}}(t)$ instead of the hard bound $b^{\mathsf{max}}(t)$. In combination with the ability to prioritize flows, this allows algorithms to achieve very low end-to-end latency without a substantial reduction of throughput. Although the stability of APCC has not been analysed in a virtual queue, it might still bring some insight to simulate and compare the algorithms in such a system.

The simulation case for the virtual bottleneck is the same as the one presented in Section 6.2 with the exception of the bottleneck being virtual and that bandwidth therefore can be borrowed from background users when flows are prioritized. This is what enables such a low queue latency for the prioritized flow. For SCReAM, DCTCP and APCC, the parameters that were used in the simulation with a physical bottleneck are also used in this simulation case. For BBRv2, the parameters had to be lightly tuned in order to achieve a reasonable comparison in this simulation.

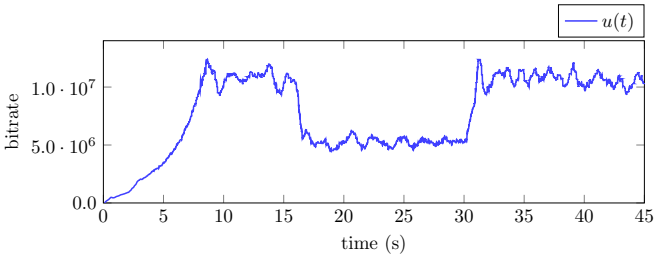
Figure 6.6 displays a plot of the sum of the background users' throughput. This

data comes from the same simulation as APCC and should therefore only be compared with the plots in Figure 6.7. The share of capacity can most easily be seen up to $t \approx 8$ s, where the throughput of the background users steadily decreases from 60 to 50 Mbps at the same time as the throughput of APCC increases from 0 to 10 Mbps in Figure 6.7(a). The effect of APCC being prioritized is best seen in the undershoot at $t \approx 16$ s in Figure 6.6, where the background users' throughput temporarily is reduced, in order to guarantee a low queue latency for APCC.

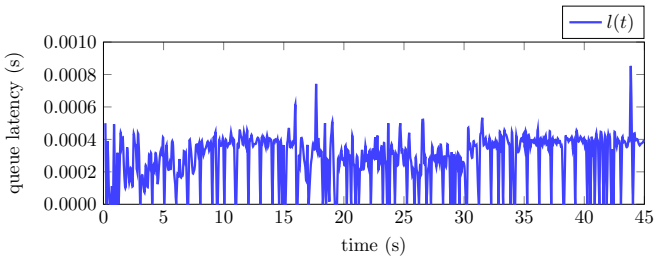
The results from the comparison with the different algorithms are presented below.

## APCC

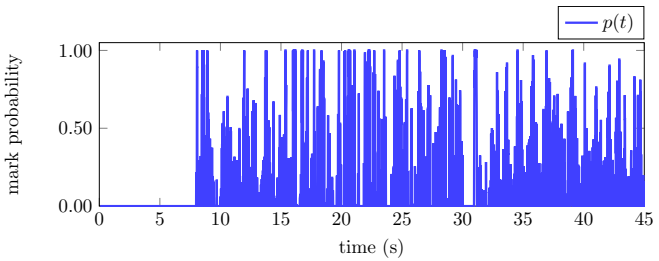
The simulation of APCC in a virtual bottleneck with priority can be seen in Figure 6.7 and behaves similarly to the physical bottleneck with some key distinctions highlighted below. The throughput seems to be slightly more oscillatory, which likely is due to no direct feedback of $b^{\mathsf{max}}(t)$. Since the flow of APCC is prioritized over the other competing flows of the background users, the queue latency in Figure 6.7(b) is substantially lower compared to the queue latency in a physical bottleneck. However, the same overshoot of throughput when detecting newly available link capacity can be seen at $t \approx 31$ s. Another thing to note is that the gain is higher between $t \approx 9-16$ s and $t \approx 32-45$ s, which might be due to achieving a lower queue latency and thus a lower RTT.



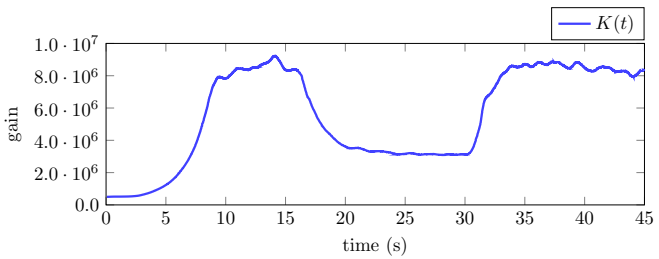

(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case



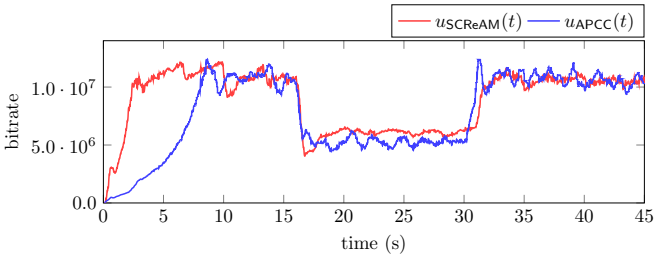

(d) Adaptive gain during the simulation case

**Figure 6.7** APCC in Ericsson's simulator and a virtual bottleneck.

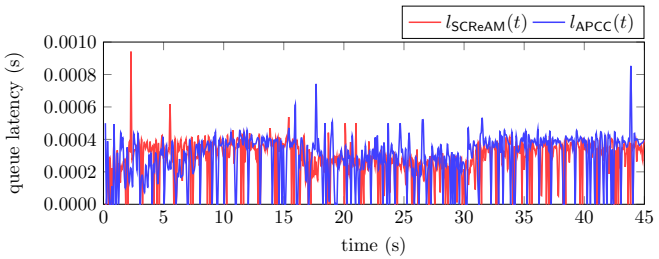
### SCReAM vs APCC

A comparison between SCReAM and APCC in a virtual bottleneck with priority is shown in Figure 6.8 and the most interesting aspects are discussed below.

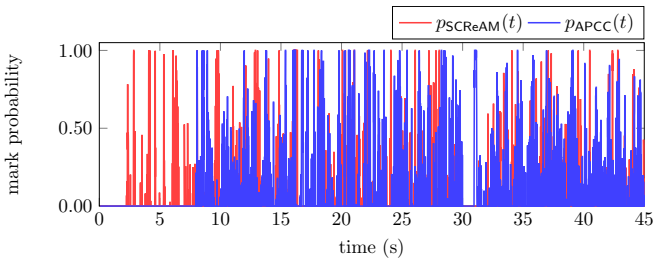
SCReAM is much faster in finding the maximum available link capacity compared to APCC during the start-up phase. Another thing to note is that SCReAM finds the maximum available link capacity at $t \approx 2.5$ s in the virtual bottleneck in Figure 6.8(a) while it takes approximately 7 s to find it in the physical bottleneck in Figure 6.3(a). APCC requires approximately 8.0 s in both cases. At $t \approx 16$ s, SCReAM undershoots a bit more compared to APCC but quickly settles at the new link capacity and sends at a close to constant rate until $t \approx 31$ s while APCC is oscillatory and constantly lower than SCReAM. In contrast to the physical bottleneck, a queue latency never occurs and is kept below 1 ms during the entire simulation for both algorithms. APCC reacts faster to the increase in link capacity at $t \approx 31$ s but overshoots while SCReAM gently finds the new capacity and oscillates less.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case

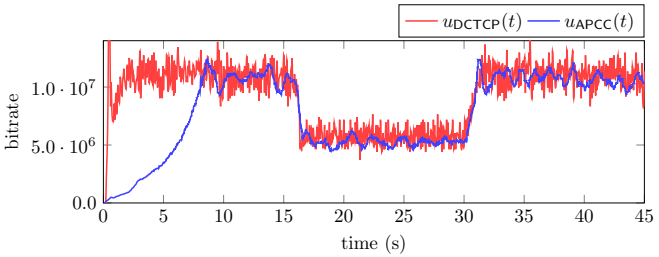



(c) Mark probability during the simulation case

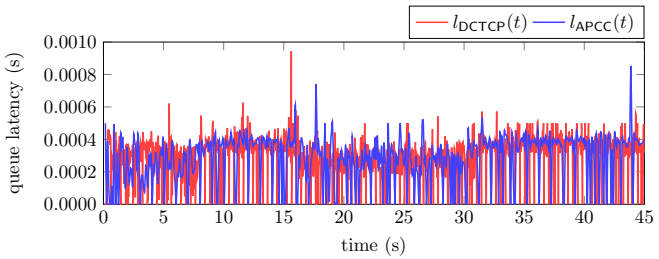
**Figure 6.8** Comparison between APCC and SCReAM in Ericsson's simulator with a virtual bottleneck.

### DCTCP vs APCC

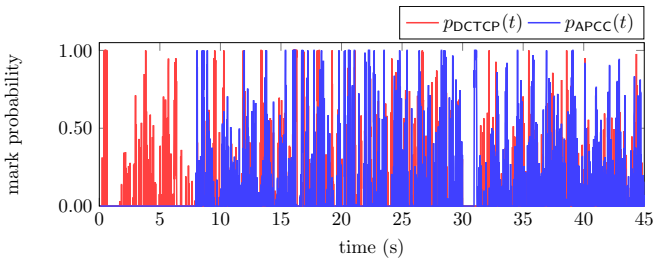
The results from the comparison between DCTCP and APCC, displayed in Figure 6.9, show similarities to the results in the physical bottleneck. The key areas of difference between the algorithms are in the start-up behaviour and the noise level of the throughput signal. Since the throughput of DCTCP is unfiltered, the noise level of the signal is greater than the throughput of APCC. Once again, since DCTCP has a specific start-up phase, it is much faster at utilizing the available capacity at the beginning of the simulation compared to APCC. DCTCP is able to transmit at a higher rate throughout most of the simulation, this is again most likely due to it not having an emulated media encoder as in the case with APCC. The latency spikes in Figure 6.9(b), seem to be more persistent and slightly higher in the DCTCP case, however, this behaviour is expected as DCTCP has a more noisy bitrate overall. Just as for the results in the physical bottleneck, with the same reasoning about their validity, the results in this comparison should be viewed with caution.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case

**Figure 6.9** Comparison between APCC and DCTCP in Ericsson's simulator with a virtual bottleneck.

### BBRv2 vs APCC

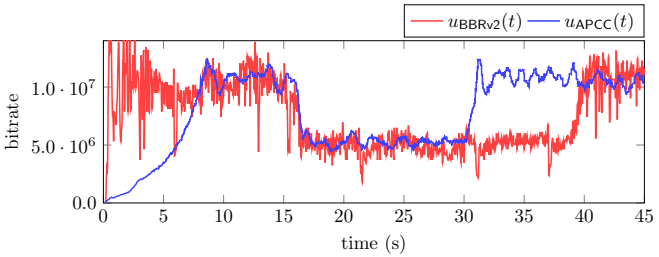
A comparison between BBRv2 and APCC in a virtual bottleneck with priority is shown in Figure 6.10 and discussed below.

Similarly to the physical bottleneck, BBRv2 uses a start-up state to quickly find the maximum available capacity. The BBRv2 throughput oscillates at the beginning followed by a section where $p(t) = 0$ which indicates that the maximum available link capacity is not utilized for either BBRv2 or APCC until $t \approx 8$ s. At this stage, BBRv2 varies more while APCC is smoother. When the link capacity decreases, BBRv2 obtains its largest queue latency while the queue latency of APCC remains low. Apart from this spike, the queue latency is low for both algorithms during the entire simulation. The link capacity increase at $t \approx 30$ s is utilized by APCC immediately while BBRv2 struggles to utilize it. It seems that BBRv2 enters a state where it does not recognize the increase until $t \approx 38$ s. The fact that BBRv2 has different states becomes clear at $t \approx 38$ s where BBRv2 starts looking for new capacity which it finds at $t \approx 40$ when $p(t) \neq 0$.

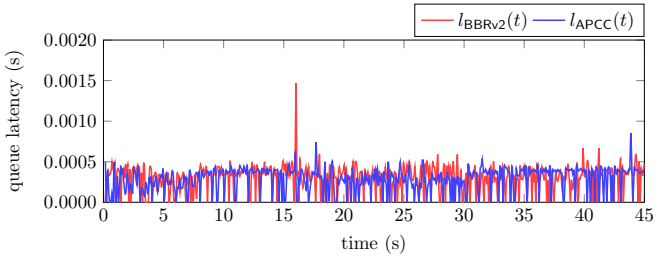
As stated in the comparison between BBRv2 and APCC in a physical bottleneck, the performance of BBRv2 varies a lot depending on the parameters used. This is also the case with a virtual bottleneck and it is possible to obtain better performance of BBRv2 with other parameters. Like in the physical bottleneck, the results of this comparison should be viewed with caution as the BBRv2 traffic in the simulation lacks a media encoder and uses non-optimal parameters which makes the comparison less fair. However, the trends and some typical behaviours of BBRv2 are still captured in Figure 6.10.

### Summary of virtual bottleneck

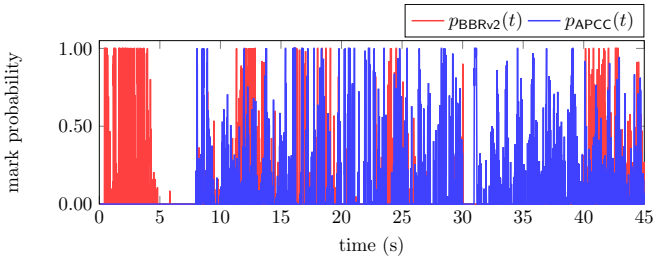
In the virtual bottleneck, all algorithms are able to achieve very low queue latency during the entire simulation despite the abrupt decrease in link capacity. This will always be the case as long as there exists resources that the prioritized users can borrow. Instead of an increase in queue latency for the prioritized users, the background users get a temporarily lower throughput while the prioritized users adapt to the new link capacity.




(a) Sending rate during the simulation case




(b) Queue latency during the simulation case




(c) Mark probability during the simulation case

**Figure 6.10** Comparison between APCC and BBRv2 in Ericsson's simulator with a virtual bottleneck.

# 7

# Discussion and Conclusion

This chapter discusses and analyses the results presented in the previous chapters along with the benefits of the derived model and proposed control structure in Section 7.1. Conclusions related to the problem formulation of the thesis are drawn in Section 7.2.

## 7.1 Discussion

When analyzing the results presented in Chapter 6, it is clear that the overall performance of all the compared algorithms is similar and that there is no clear best or worst performer in all aspects of the simulation. Best throughput performance, in both queue types, is achieved with DCTCP as it utilizes the available capacity throughout the simulation without high queue latency. The most notable difference in queue latency performance, in the physical queue, between the algorithms becomes evident when the available link capacity suddenly decreases, where APCC performs best with BBRv2 and DCTCP as the close second contenders. However, one has to keep in mind that the algorithms were developed for different purposes and that neither DCTCP nor BBRv2 is designed for conversational media services as APCC and SCReAM are. For this reason, both APCC and SCReAM use an emulated video-encoder which in other words creates a layer of ”inertia“ present in SCReAM and APCC that are not present in BBRv2 nor DCTCP.

Another note is that not enough time has been spent on tuning the algorithms to perform at their best in this specific simulation scenario, which also affects the fairness of the comparison. DCTCP and BBRv2 both implement a fast start phase of the algorithm which makes them outperform SCReAM and APCC in this regard. Implementing a fast start phase for APCC could drastically improve the throughput performance at the beginning of the simulation, however, due to time constraints such an implementation has been left for future work. This is further discussed in Section 8.1

In the virtual bottleneck, it is evident that all of the algorithms perform very well from a latency perspective and that prioritization of flows drastically reduces the

experienced queue latency. When looking at the end-to-end latency problem from a bigger picture, it becomes clear that not all performance is related to the congestion control algorithms. Although better latency performance can be achieved with a good congestion control algorithm, latency performance gains can for instance also be achieved in the prioritization of flows in the AQM and minimizing the link delays in a clever way. However, since the main focus of this thesis has been to develop a simple algorithm suitable for tractable analysis, most of the time spent has been in this area.

### Implementation drawbacks

Replacing the way SCReAM updates its sending rate with APCC, has allowed for testing of APCC in a simulated cellular network, but it also means that APCC is dependent on the way SCReAM estimates its incoming signal $b(t)$, $p(t)$ and $RTT(t)$. Since APCC has been developed from control theory, it is highly dependent on its feedback signals. In fact, the authors of this thesis have found that the performance of APCC substantially differs with different estimation techniques of $b(t)$, $p(t)$ and $RTT(t)$ and the frequency by which they are fed back. This is most notable in the way that SCReAM estimates $b(t)$, which likely is the cause of the overshoots in the simulation of APCC. This phenomenon is discussed as future work in Section 8.1.

### Effects of the mark probability function

As discussed in Section 2.4, the mark probability function $p(t)$ can vary in different BSs. This means that the stability condition in Equation (4.8) and particularly the slope, $\Delta T$ of the mark function $p(t)$ might vary with different BSs. In turn, this affects how the system behaves at different operating points, $p_{\mathsf{ref}}$, where it might be more difficult for the system to operate around a point in a steep slope compared to a moderate one, especially if a non-linear $p(t)$ is used.

### Model correctness

The behaviour of APCC in the Julia simulation as seen in Figure 5.3 is surprisingly similar to the behaviour observed in the Ericsson simulation despite the latter being a much more complex and ”complete“ simulator. Most notable is the start-up behaviour, the tendency to overshoot with increases in available capacity and the queue latency spike when the capacity drops, all of which can also be found in the Ericsson simulations. The similar behaviour indicates that the relatively simple model presented in Chapter 3 and which the Julia simulation is based on captures the dynamics of the cellular system quite well. The simplicity of the model is an important aspect of control design as it allows for tractable analysis in systems which otherwise would be very difficult to analyse and develop a control-based algorithm for.

### The benefit of simplicity

As mentioned above, being able to analyse a system is an important aspect, which in most cases enables prediction of algorithm behaviour and thus a reduction of testing time. The simplicity, analyzability and predictability of the algorithm behaviour is something that the authors of this thesis have been striving for when designing APCC. Comparing the methods used to analyze the system in this thesis with methods such as limit cycles and Poincaré maps used in the analysis of DCTCP in [Alizadeh et al., 2011], it is clear that it is far easier to analyze APCC.

## 7.2 Conclusion

In this master thesis, an adaptive control algorithm for low latency applications over the cellular network has been derived. The controller stems from classic control theory, enabled by modelling the communication system mathematically. By analysing the results in Chapter 6, it is possible to conclude that the simple adaptive P controller, APCC, presented in Section 4.1 and Section 5.3 is able to perform on par with the current state-of-the-art algorithms, yet simple enough to do tractable analysis on. Moreover, having many parameters in an algorithm allow for algorithm flexibility and can yield satisfactory results but be difficult to tune. On the other hand, an algorithm with fewer parameters limits the flexibility but allows for tractable analysis and performs sufficiently in the simulations of this thesis.

# 8

# Future work

Although the results of this thesis have been satisfactory, there are some areas that have not been explored due to time constraints. The goal of this chapter is to provide a guideline and present in which direction one should continue this work to make the performance of the controller even better. There are, of course, multiple ways of improving the controller and some of them are emphasized in the coming sections.

## 8.1 Signal and parameter estimation

The continuous-time signals, $u(t)$, $b(t)$, $b^{\max}(t)$ and $p(t)$, are theoretical constructions used in this thesis that allows the use of powerful analysis tools from control theory. In any actual implementation of the system, information about the continuous signals is instead present in discrete packages and how the continuous signals are estimated from discrete packages will influence the performance of the controller. This thesis uses the same estimation techniques as in SCReAM and a potential way of making APCC better would be to estimate the continuous signals differently.

### Estimating the mark probability

The mark probability function, $p(t)$, is used in the bottleneck to mark individual packets according to a certain probability based on the queue latency. Acknowledgements arriving back to the controller will either have their ECN bit set or not and how to estimate $p(t)$ from individual marked packets is not trivial. The way SCReAM does it, described in more detail in Section 6.2, is to calculate the fraction between marked bytes and delivered bytes this RTT without any memory of previous estimations. DCTCP, on the other hand, uses a similar technique with an added exponentially weighted moving average filter to have a memory of previous estimates which was shown in Section 2.7. The most suitable method of obtaining $p(t)$ from marked acknowledgements when using the controller in this thesis has not been evaluated and a different method for estimating $p(t)$ might improve the performance.

### Estimating the slope of the mark probability

APCC is from an implementation aspect quite simple. The choice of $K(t)$ is as stated in Equation (4.10) dependent on $\beta$ that the AS is free to choose, $b(t)$ which is constructed from acknowledgements, $D$ which can be inferred from the RTT and finally $\Delta T$ which is not measurable. The good thing is that, if a rough estimation of $\Delta T$ is known, the uncertainty in $\Delta T$ can be compensated for by adjusting the $\beta$ parameter. If a more accurate estimate of $\Delta T$ is necessary for implementation purposes, it should be possible to apply system identification methods to identify and estimate $\Delta T$.

### Estimating the outgoing rate of the bottleneck

As mentioned in Section 7.1, the outgoing rate estimate of the bottleneck, $b(t)$, is likely what causes the overshoots when the link capacity increases in Figure 6.2 and 6.7. Similar to the mark probability above, $b(t)$ also has to be estimated from the acknowledgements that might have a varying time gap between them. The fact that not all packets are acknowledged immediately in an attempt to send less data, makes the task even more challenging [Sarker et al., 2021]. The way SCReAM estimates $b(t)$, described in Section 6.2, shows that it is possible to estimate $b(t)$ and allows for the simple control law in Equation (4.1). An even better estimate is believed to be able to remove the overshoots when the available link capacity increases and result in better performance of APCC.

### Estimating available link capacity in start-up

An obvious area of improvement for APCC in Figure 6.2 and 6.7 is in the start-up phase where the available link capacity is unknown. Compared to DCTCP and BBRv2, which both have a certain start-up phase, APCC requires a much longer time to find and utilize the available link capacity. There are various possible techniques to do a quicker start-up and in essence, they all quickly build up a queue followed by a backoff in sending rate. Paced chirping [Misund and Briscoe, 2019], used in DCTCP, is one of these methods where a chirp signal of packets is sent with a decreasing inter-packet gap spacing. The sender analyses the acknowledgement-spacing, specifically from the point at which the gaps between acknowledgements become larger than the gaps between sent packages to determine the available capacity. The focus of this thesis has been on the link capacity changes in steady-state but an implementation of the controller in a real application would benefit from having a specific start-up phase and Paced chirping is a promising solution for that.

## 8.2 Alternative control structures

APCC performed well in the physical bottleneck while the throughput in the virtual bottleneck was more oscillatory. The oscillations occur because the system dynam-

ics are changed in a virtual bottleneck compared to a physical bottleneck. One possible way of reducing the oscillations and improving the steady-state behaviour in the virtual bottleneck would be to add derivative action to the controller. Derivative action comes with implementation trade-offs due to amplification of high-frequency noise if the measurement signal is not filtered. This would most likely not be an issue here since the measurement signal, $p(t)$, is a constructed software signal from individual packets and therefore not subject to noise from a sensor, which is often the case in control problems. Choosing two parameters, the proportional and the derivative gain is naturally more difficult than only choosing the proportional gain in the P controller. However, with a proper analysis of the virtual bottleneck, similar to the one presented in Section 3.4 for the physical bottleneck, it should be possible to derive stability conditions for the two parameters of a PD controller. Adapting the two parameters to have the same stability margins as $b(t)$ changes should also be feasible and theoretically result in a less oscillatory throughput.

## 8.3 Further analysis

The stability analysis of the P controller in Section 4.2 is based on the dynamics of a physical bottleneck, in the continuous-time domain and under the assumption that the inner loop of the controller does not alter the stability. All of these aspects could be analysed further to get a better understanding of the system and briefly discussed below.

### Analysis of virtual bottleneck

The dynamics of a virtual bottleneck are as described in Section 3.3, different to the dynamics of a physical bottleneck. This becomes evident when comparing the results of the controller in a physical and a virtual bottleneck where the latter yields a more oscillatory throughput. To understand what parameters affect the performance in the virtual bottleneck, similar analysis as the one of the physical bottleneck in Section 4.2 would be required. With a virtual bottleneck and its soft upper limit, $b_{\mathsf{virt}}^{\mathsf{max}}$, the assumption that the inner loop of the controller can be ignored during the analysis no longer holds. As a result, the virtual bottleneck could require analysis in the Single Input, Multiple Output (SIMO) domain where the single input is the throughput, $a(t)$, and the two outputs of the system would be $b(t)$ and $p(t)$ as in Figure 4.1. Naturally, this becomes a more challenging analysis to perform compared to the analysis of the physical bottleneck but would also resemble the dynamics of the virtual bottleneck better.

An alternative way, that would allow the stability analysis for the physical bottleneck to also be valid for the virtual bottleneck, would be to focus on creating an estimator of $b^{\mathsf{max}}(t)$ in the physical bottleneck and $b_{\mathsf{virt}}^{\mathsf{max}}(t)$ in the virtual bottleneck. By feeding the inputs and outputs of the system, $u(t)$, $b(t)$ and $p(t)$, into an estimator that outputs either $b^{\mathsf{max}}(t)$ or $b_{\mathsf{virt}}^{\mathsf{max}}(t)$ to the controller, depending on the

bottleneck type, the proposed P controller and its stability analysis would still be useful. The difference would be that $c(t)$ in Equation (4.1) would be replaced with the estimation of $b^{\max}(t)$ or $b^{\max}_{\text{virt}}(t)$. The problem to solve would instead be how to estimate $b^{\max}(t)$ or $b^{\max}_{\text{virt}}(t)$ from $u(t)$, $b(t)$ and $p(t)$, but might be a better alternative to the SIMO analysis above.

### Analysis of discretized system

Similarly to most control loops today, APCC will be implemented in a discrete environment and a useful addition to the stability analysis in Section 4.2 would therefore be to discretize the system. This would make it possible to, for example, discuss what sampling times are necessary to obtain good performance and if any new behaviours arise when discretizing the system.

# Bibliography


Alizadeh, M., A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan (2010). "Data center tcp (dctcp)". *SIGCOMM Comput. Commun. Rev.* **40**:4, pp. 63–74. ISSN: 0146-4833. DOI: 10.1145/1851275.1851192. URL: https://doi.org/10.1145/1851275.1851192.

Alizadeh, M., A. Javanmard, and B. Prabhakar (2011). "Analysis of dctcp: stability, convergence, and fairness". In: vol. 39, pp. 73–84. DOI: 10.1145/1993744.1993753.

Åström, K. and R. Murray (2008). *Feedback Systems : An Introduction for Scientists and Engineers*. English. Princeton University Press. ISBN: 9781400828739.

Bensley, S., D. Thaler, P. Balasubramanian, L. Eggert, and G. Judd (2017). *Data center tcp (dctcp): tcp congestion control for data centers*. RFC 8257. DOI: 10.17487/RFC8257. URL: https://www.rfc-editor.org/info/rfc8257.

Briscoe, B. (2019). "The native AQM for L4S traffic". *CoRR*. arXiv: 1904.07079. URL: http://arxiv.org/abs/1904.07079.

Briscoe, B., M. Kühlewind, and R. Scheffenegger (2022a). *More Accurate ECN Feedback in TCP*. Internet-Draft draft-ietf-tcpm-accurate-ecn-18. Work in Progress. Internet Engineering Task Force. 61 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-18.

Briscoe, B., K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Täht (2018). "Implementing the ' prague requirements ' for low latency low loss scalable throughput ( l 4 s )". In:

Briscoe, B., K. D. Schepper, M. Bagnulo, and G. White (2022b). *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture*. Internet-Draft draft-ietf-tsvwg-l4s-arch-16. Work in Progress. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-16.

Cardwell, N., Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson (2016). "Bbr: congestion-based congestion control: measuring bottleneck bandwidth and round-trip propagation time". *Queue* **14**:5, pp. 20–53. ISSN: 1542-7730. DOI: 10.1145/3012426.3022184. URL: https://doi.org/10.1145/3012426.3022184.

Cheng, Y., N. Cardwell, N. Dukkipati, and P. Jha (2021). *The RACK-TLP Loss Detection Algorithm for TCP*. RFC 8985. DOI: 10.17487/RFC8985. URL: https://www.rfc-editor.org/info/rfc8985.

Floyd, S. and V. Jacobson (1993). "Random early detection gateways for congestion avoidance". *IEEE/ACM Transactions on Networking* **1**:4, pp. 397–413. DOI: 10.1109/90.251892.

Ha, S., I. Rhee, and L. Xu (2008). "Cubic: a new tcp-friendly high-speed tcp variant". *Operating Systems Review* **42**, pp. 64–74. DOI: 10.1145/1400097.1400105.

Hock, M., R. Bless, and M. Zitterbart (2017). "Experimental evaluation of bbr congestion control". In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10. DOI: 10.1109/ICNP.2017.8117540.

Jain, V., V. Mittal, and M. P. Tahiliani (2018). "Design and implementation of tcp bbr in ns-3". In: *Proceedings of the 10th Workshop on Ns-3*. WNS3 '18. Association for Computing Machinery, Surathkal, India, pp. 16–22. ISBN: 9781450364133. DOI: 10.1145/3199902.3199911. URL: https://doi.org/10.1145/3199902.3199911.

Johansson, I., S. Dadhich, U. Bodin, and T. Jönsson (2018). "Adaptive video with SCReAM over LTE for remote-operated working machines". *Wireless Communications and Mobile Computing*. ISSN: 1530-8669. DOI: 10.1155/2018/3142496. URL: https://doi.org/10.1155/2018/3142496.

Johansson, I. and Z. Sarker (2017). *Self-clocked rate adaptation for multimedia*. RFC 8298. DOI: 10.17487/RFC8298. URL: https://www.rfc-editor.org/info/rfc8298.

Leung, I.-K. and J. Muppala (2001). "Packet marking strategies for explicit congestion notification (ecn)". In: *Conference Proceedings of the 2001 IEEE International Performance, Computing, and Communications Conference (Cat. No.01CH37210)*, pp. 17–23. DOI: 10.1109/IPCCC.2001.918631.

Ma, S., J. Jiang, W. Wang, and B. Li (2017). *Fairness of congestion-based congestion control: experimental evaluation and analysis*. DOI: 10.48550/ARXIV.1706.09115. URL: https://arxiv.org/abs/1706.09115.

Misund, J. and B. Briscoe (2019). "Paced chirping: rapid flow start with very low queuing delay". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 798–804. DOI: 10.1109/INFCOMW.2019.8845072.

Nylander, T., C. Klein, K.-E. Årzén, and M. Maggio (2018). "Brownoutcc: cascaded control for bounding the response times of cloud applications". English. In: *Proceedings of the American Control Conference*. Vol. 2018-June. American Control Conference 2018, ACC 2018 ; Conference date: 27-06-2018 Through 29-06-2018. IEEE - Institute of Electrical and Electronics Engineers Inc., United States, pp. 3354–3361. ISBN: 978-153865428-6. DOI: 10.23919/ACC.2018.8431282. URL: http://acc2018.a2c2.org/.

Nyquist, H. (1932). "Regeneration theory". *The Bell System Technical Journal* **11**:1, pp. 126–147. DOI: 10.1002/j.1538-7305.1932.tb02344.x.

Pan, W., H. Tan, X. Li, J. Xu, and X. Li (2022). "Improvement of bbrv2 congestion control algorithm based on flow-aware ecn". *Security and Communication Networks* **2022**. ISSN: 1939-0114. DOI: 10.1155/2022/1218245. URL: https://doi.org/10.1155/2022/1218245.

Sarker, Z., C. Perkins, V. Singh, and M. A. Ramalho (2021). *RTP Control Protocol (RTCP) Feedback for Congestion Control*. RFC 8888. DOI: 10.17487/RFC8888. URL: https://www.rfc-editor.org/info/rfc8888.

Schepper, K. D., O. Tilmans, and B. Briscoe (2021). *Prague Congestion Control*. Internet-Draft draft-briscoe-iccrg-prague-congestion-control-00. Work in Progress. Internet Engineering Task Force. 30 pp. URL: https://datatracker.ietf.org/doc/html/draft-briscoe-iccrg-prague-congestion-control-00.

Scholz, D., B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle (2018). "Towards a deeper understanding of tcp bbr congestion control". In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1–9. DOI: 10.23919/IFIPNetworking.2018.8696830.

Schulz, P., M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch (2017). "Latency critical iot applications in 5g: perspective on the design of radio interface and network architecture". *IEEE Communications Magazine* **55**:2, pp. 70–78. DOI: 10.1109/MCOM.2017.1600435CM.

Shalunov, S., G. Hazel, J. Iyengar, and M. Kühlewind (2012). *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817. DOI: 10.17487/RFC6817. URL: https://www.rfc-editor.org/info/rfc6817.

Shiriaev, A., R. Johansson, and A. Robertsson (2004). "Sufficient conditions for dynamical output feedback stabilization via the circle criterion". In: vol. 5, 4682–4687 Vol.5. ISBN: 0-7803-7924-1. DOI: 10.1109/CDC.2003.1272310.

Smith, O. J. (1959). "A controller to overcome dead time". *ISA J.* **6**, pp. 28–33.

**Lund University**
**Department of Automatic Control**
**Box 118**
**SE-221 00 Lund Sweden**

*Document name*
MASTER'S THESIS

*Date of issue*
June 2022

*Document Number*
TFRT-6175

*Author(s)*
Johan Siwerson
Samuel Gunnarsson

*Supervisor*

Victor Millnert, Ericsson AB, Sweden
Johan Eker, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

*Title and subtitle*
Adaptive Server Control for Low-Latency Applications over the Cellular Network

*Abstract*

Queue latency is a fundamental part of end-to-end latency and ensuring low end-to-end latency is crucial for a good quality of service in the upcoming 5G and 6G applications. In this work we present a model capturing the essential dynamics of an end-to-end cellular system upon which we derive a control structure that is suitable for tractable analysis, seeking to mitigate queue latency and provide stability guarantees. We do this by analysing the model and leveraging classic control theory. The controller is evaluated both in a self-built Julia simulation framework and a state-of-the-art network simulator where it also is compared to state-of-the-art congestion control algorithms. The results are promising and show that the proposed control structure performs on par with the compared congestion controllers despite its simple nature. The most prominent area of improvement is in signal estimation, where we believe that the proposed controller could gain in performance and yield better results.

*Keywords*
Congestion control, Adaptive control, Control theory, Latency

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280-5316

*ISBN*

*Language*
English

*Number of pages*
1-79

*Recipient's notes*

*Security classification*

http://www.control.lth.se/publications/